Technical Report

Getting Started with DIAsDEM Workbench 2.1: A Case-Based Tutorial

Karsten Winkler

HHL – Leipzig Graduate School of Management
Department of E-Business
Jahnallee 59, D-04109 Leipzig, Germany
E-Mail: kwinkler@ebusiness.hhl.de

Copyright: 2003
All rights reserved.

HHL. . .
Leipzig Graduate School of Management

Contents

1 Introduction **4**
1.1 The DIAsDEM Framework . . . 4
1.2 Code Credits and Trademarks . . . 7
1.3 License of DIAsDEM Workbench 2.1 . . . 8
1.4 Typographical Conventions . . . 9

2 Installation **10**
2.1 Prerequisites . . . 10
2.2 Unix/Linux . . . 10
2.3 Windows . . . 11

3 Case Study **13**
3.1 Application Domain and Data Set . . . 13
3.2 Text Pre-Processing in the KDD Phase . . . 14
3.2.1 Starting the Batch Script Recorder . . . 16
3.2.2 Creating a New Project . . . 17
3.2.3 Creating a Document Collection . . . 19
3.2.4 Importing Plain Text Files . . . 21
3.2.5 Creating Text Units . . . 23
3.2.6 Tokenizing Text Units . . . 26
3.2.7 Replacing Named Entities with NEEX 2.1 . . . 29
3.2.8 Lemmatizing Text Units . . . 37
3.3 Iterative Clustering in the KDD Phase . . . 40
3.3.1 Computing Term Frequency Statistics . . . 40
3.3.2 Viewing Term Frequency Statistics . . . 43
3.3.3 Editing Domain-Specific Thesauri . . . 44
3.3.4 Vectorizing Text Units in Iteration 1 . . . 47
3.3.5 Clustering Text Unit Vectors in Iteration 1 . . . 51
3.3.6 Monitoring Cluster Quality in Iteration 1 . . . 55
3.3.7 Editing the Cluster Label File in Iteration 1 . . . 60
3.3.8 Tagging Text Units in Iteration 1 . . . 62
3.3.9 Summary of KDD Process Iteration 2 . . . 65

3.4 XML Tagging of Texts in the KDD Phase . . . 69
3.4.1 Deriving a Preliminary XML DTD . . . 69
3.4.2 Tagging Documents . . . 72
3.4.3 Evaluating the Tagging Quality . . . 75
3.4.4 Stopping the Batch Script Recorder . . . 78
3.5 Summary of the Application Phase . . . 78
3.5.1 Preparing the Application Phase . . . 80
3.5.2 Editing the Batch Script . . . 80
3.5.3 Executing the Batch Script . . . 83
3.6 Auxiliary Tasks . . . 85
3.6.1 Replacing Named Entities with NEEX 2.0 . . . 85
3.6.2 Removing Stopwords . . . 88
3.6.3 Establishing an Initial Thesaurus . . . 89

4 Technical Specification 91
4.1 DIAsDEM Documents . . . 91
4.2 DIAsDEM Batch Scripts . . . 92
4.3 Text Pre-Processing . . . 92
4.3.1 Module: Create Text Units . . . 92
4.3.2 Module: Tokenize Text Units . . . 93
4.3.3 Module: Replace Named Entities 2.1 . . . 94
4.3.4 Module: Replace Named Entities 2.0 . . . 102
4.3.5 Module: Remove Stopwords . . . 107
4.3.6 Module: Lemmatize Text Units . . . 107
4.4 Iterative Clustering . . . 109
4.4.1 Module: Vectorize Text Units . . . 109
4.4.2 Module: Cluster Text Unit Vectors (Weka) . . . 112
4.4.3 Module: Monitor Cluster Quality . . . 113
4.4.4 Module: Tag Text Units . . . 114
4.5 XML Tagging of Texts . . . 114
4.5.1 Module: Derive Preliminary DTD . . . 114
4.5.2 Module: Tag Documents . . . 115

List of Abbreviations 117

List of Relevant German Vocabulary 118

Bibliography 120

1 Introduction

Most organizations are not only "drowning" in data, they are also "struggling" to cope with huge amounts of text documents. Currently, up to 80% of a company's information is stored in unstructured textual documents [Sul01, p. 56]. Hence, capturing interesting and actionable knowledge from textual databases is a major challenge. Creating semantic markup is one form of providing explicit knowledge about text archives to facilitate searching and browsing or to enable information integration with related data sources. Unfortunately, most users are not willing to manually create meta-data due to the efforts and costs involved. Thus, text mining techniques are required that (semi-) automatically create semantic markup.

Using the Extensible Markup Language XML, semantic annotation of text archives results in application-specific, semantic meta-data in the form of XML tags and an archive-specific XML document type definition (DTD). Semantic meta-data can be utilized to facilitate for example knowledge management and information integration. Appropriate XML query languages could be employed to submit both content- and structure-based queries against XML archives. However, two main problems must be solved to semi-automatically create text annotations: Firstly, an appropriately structured, semantic DTD should be derived for each textual archive. Secondly, all text documents contained in an archive should be semantically tagged according to the previously derived document type definition.

In the next section, the DIAsDEM[1] framework for semantic tagging of large, domain-specific text archives is concisely introduced. The reader might refer to [GWS01, GSW01, WS01c, WS01a, GWS01] for a complete description of the DIAsDEM framework as well as a thorough discussion of related work. Figure 1.1 illustrates the user interface of DIAsDEM Workbench 2.1. This Java-based research prototype supports the entire framework including automated batch processing.

1.1 The DIAsDEM Framework

In DIAsDEM, the notion of semantic tagging refers to annotating texts with domain-specific XML tags that might contain additional attributes describing extracted named entities (e.g., names of persons). Rather than classifying entire documents or tagging

[1] The acronym DIAsDEM is the name of a research project funded by Deutsche Forschungsgemeinschaft (German Research Society, http://www.dfg.de), DFG grants: SP 572/4-1 and SP 572/4-3.

Figure 1.1: Java-based GUI of DIAsDEM Workbench 2.1

single terms, we aim at semantically annotating structural text units such as sentences or paragraphs in order to make their semantics explicit. The following example illustrates two tagged sentences contained in a German Commercial Register entry, whereas each sentence is a text unit:

> <`BusinessPurpose`> Der Betrieb von Spielhallen in Teltow und das Aufstellen von Geldspiel- und Unterhaltungsautomaten. <`/BusinessPurpose`>
> <`AppointmentManagingDirector Person="Balski; Pawel"`> Pawel Balski ist zum Geschäftsführer bestellt. <`/AppointmentManagingDirector`>

Semantic tagging in DIAsDEM is a two-phase process. We have designed a knowledge discovery in textual databases (KDT) process that constitutes the first phase in order to build clusters of semantically similar text units, to derive an XML DTD describing the archive and to tag documents in XML according to the results . The KDT process, which is depicted in Figure 1.2, results in a final set of clusters. Their semantic labels serve as DTD elements and XML tags, respectively. Huge amounts of new documents can be automatically converted into XML documents in the second, batch-oriented and productive phase of the DIAsDEM framework.

Besides the initial text documents to be tagged, the following domain knowledge constitutes input to the KDT process: A thesaurus [ISO86] containing a domain-specific taxonomy of terms and concepts, a preliminary UML schema of the domain and descriptions of specific named entities, e.g. persons and companies. The UML schema reflects the semantics of named entities and the relationships among them, as they are initially conceived by application experts. This schema serves as a reference for the DTD to be

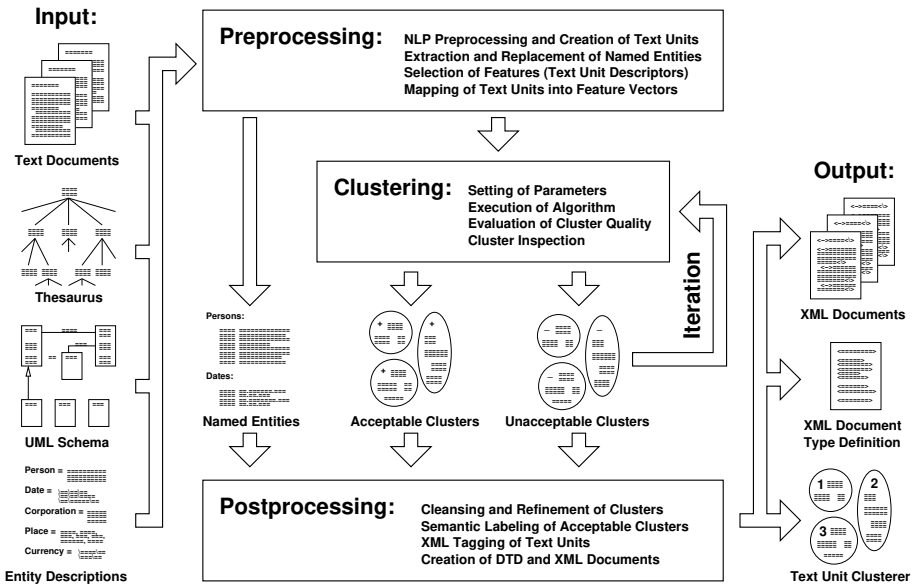

Figure 1.2: Iterative and interactive KDT process of the DIAsDEM framework

derived from discovered semantic tags. However, there is no guarantee that the final DTD will be contained in or will contain this schema.

Similarly to a conventional KDD process, the process starts with a pre-processing phase that includes basic NLP pre-processing tasks such as tokenization, normalization, and word stemming as well as named entity extraction. Instead of removing stop words, we establish a drastically reduced feature space by selecting a limited set of terms and concepts (so-called text unit descriptors) from the thesaurus and the UML schema. Text unit descriptors are currently chosen by the knowledge engineer, because they must reflect important concepts of the application domain. Text units are initially mapped onto Boolean vectors of this feature space. Thereafter, Boolean text unit vectors are further processed by applying the information retrieval TFxIDF weighting schema.

In the pattern discovery phase, all text unit vectors contained in the initial archive are clustered based on content similarity. The objective is to discover dense and homogeneous text unit clusters. Clustering is performed in multiple iterations. Each iteration outputs a set of clusters, which is partitioned into qualitatively "acceptable" and "unacceptable" ones according to our quality criteria. A cluster of text unit vectors is "acceptable", if and only if (i) its cardinality is large and the corresponding text units are (ii) homogeneous and (iii) can be semantically described by a small number of text unit descriptors. Members of "acceptable" cluster are subsequently removed from the data set for later labeling, whereas the remaining text unit vectors are input to the clus-

tering algorithm in the next iteration. In each iteration, the cluster similarity threshold value is stepwisely decreased such that "acceptable" clusters become progressively less specific in content. The KDT process is based on a plug-in and a plug-out concept that allows the execution of various clustering algorithms within DIAsDEM Workbench.

In the post-mining phase, qualitatively "acceptable" clusters are semi-automatically assigned a semantic label. DIAsDEM Workbench suggests default cluster labels for "acceptable" clusters that are derived from prevailing feature space dimensions (i.e., text unit descriptors) in each "acceptable" cluster. Cluster labels actually correspond to XML tags that are subsequently used to annotate cluster members. Thereafter, a preliminary XML DTD is derived that coarsely describes the semantic structure of the XML collection by enumerating discovered XML tags. Finally, original documents are annotated by valid XML tags with respect to the derived XML DTD. The archive-specific XML tags might include attributes reflecting previously extracted named entities and their values. The following DTD excerpt was created in a recent case study [WS01c]:

```
<!ELEMENT CommercialRegisterEntry ( #PCDATA | BusinessPurpose | ShareCapital |
SupervisoryBoard | AppointmentManagingDirector | (...) | Owner |
FoundationPartnership )* >  <!ELEMENT BusinessPurpose (#PCDATA)> (...)
<!ELEMENT FoundationPartnership (#PCDATA)>
```

1.2 Code Credits and Trademarks

Markus Banach, Martin Christian, Henner Graubitz and Karsten Winkler are code contributors. The research project DIAsDEM is funded by Deutsche Forschungsgemeinschaft (German Research Foundation), DFG grants SP 572/4-1 and SP 572/4-3. Information about Deutsche Forschungsgemeinschaft is available at http://www.dfg.de.

DIAsDEM Workbench 2.1 utilizes software developed by the JDOM Project (http://www.jdom.org/). Copyright (C) 2000-2003 Jason Hunter & Brett McLaughlin. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact <license AT jdom DOT org>.
4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management <pm AT jdom DOT org>.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the JDOM Project and was originally created by Jason Hunter <jhunter AT jdom DOT org> and Brett McLaughlin <brett AT jdom DOT org>. For more information on the JDOM Project, please see <http://www.jdom.org/>.

DIAsDEM Workbench 2.1 utilizes WEKA 3.3.3, 28 June 2002, Java Programs for Machine Learning. Copyright (C) 1998, 1999, 2000, 2001, 2002 Eibe Frank, Leonard Trigg, Mark Hall, Richard Kirkby. WEKA is distributed under the GNU public license available at http://www.opensource.org/licenses/gpl-license.php.

DIAsDEM Workbench 2.1 utilizes GNU Regular Expressions for Java 1.0.8 (http://www.cacas.org/java/gnu/regexp/). GNU Regular Expressions for Java 1.0.8 is distributed under the GNU Lesser General Public License available at http://www.opensource.org/licenses/lgpl-license.php.

Sun, Sun Microsystems, the Sun logo, Sun Microsystems, JavaSoft, JavaBeans, JDK, Java, the Java Coffee Cup logo, and Visual Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other tradenames, trademarks, and registered trademarks are the property of their respective owners.

1.3 License of DIAsDEM Workbench 2.1

Copyright (C) 2000-2003, Henner Graubitz, Myra Spiliopoulou, Karsten Winkler. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the research project DIAsDEM funded by Deutsche Forschungsgemeinschaft (German Research Foundation, DFG grants SP 572/4-1 and SP 572/4-3) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.4 Typographical Conventions

Italic is used for emphasis within text and to indicate selectable items in DIAsDEM Workbench windows and menus. *Italic* is also used to represent field names (i.e., parameters) in DIAsDEM Workbench dialogs and windows. `Courier` is used to refer to directories, file names and file extentions. Additionally, `Courier` is used for computer output, XML tags and contents of files (e.g., XML and text files). In the remainder of this case study, the following abbreviations indicate directories on your file system as listed below. Note, these four abbreviations do not correspond to environment variables.

- `${DIAsDEM_HOME}` denotes the local directory of DIAsDEM Workbench, e.g. `/home/kwinkler/diasdem/DIAsDEM.workbench21`.
- `${PARAMETER_HOME}` denotes the local subdirectory of `${DIAsDEM_HOME}` that contains default parameter files, i.e., `${DIAsDEM_HOME}/data/parameters`.
- `${SAMPLES_HOME}` denotes the local subdirectory of `${DIAsDEM_HOME}` that contains sample text files, i.e., `${DIAsDEM_HOME}/data/samples`.
- `${PROJECT_HOME}` denotes the local directory that contains all files related to a single project, e.g., `/home/kwinkler/diasdem/DIAsDEM.cases/tutorial`.

2 Installation

2.1 Prerequisites

The target machine must be equipped with at least 256 MB memory. Either the Java 2 Runtime Environment 1.4.0 (or higher) or the Java 2 Software Development Kit, Standard Edition, 1.4.0 (or higher) must be installed on the target machine. Visit the Web site http://java.sun.com to download the required Java release. Note, Java releases below 1.4 cannot be used to launch DIAsDEM Workbench, because it heavily depends on the support of regular expressions by the package `java.util.regex`. It is strongly recommended using the latest Java release (i.e., Java 1.4.2 as of August 2003) in order to profit from steady and noticeable performance improvements.

2.2 Unix/Linux

1. Visit the Web page http://www.hypknowsys.org/diasdem and download the compressed archive file `DIAsDEM.workbench21.tar.gz`.

2. Create a directory for DIAsDEM Workbench (e.g., `/home/kwinkler/diasdem`) and copy the file `DIAsDEM.workbench21.tar.gz` into this directory. Additionally, ensure that you have write permission in this new directory.

3. Make the DIAsDEM-specific directory (e.g., `/home/kwinkler/diasdem`) your current working directory and unzip the compressed file archive by submitting the following two commands at the prompt:

   ```
   /home/kwinkler/diasdem> gunzip DIAsDEM.workbench21.tar.gz
   /home/kwinkler/diasdem> tar -xf DIAsDEM.workbench21.tar
   ```

4. Using any common text editor, modify the environment variables `JAVA_HOME` and `DIAsDEM_HOME` in both shell scripts `DIAsDEM.workbench21/bin/diasdemgui` and `DIAsDEM.workbench21/bin/diasdembatch` (e.g., in the directory `/home/kwinkler/diasdem`) according to your system. For example, if Java is installed in the directory `/usr/lib/j2sdk1.4.0_01` and if the file `DIAsDEM.workbench21.tar.gz` was uncompressed in the directory `/home/kwinkler/diasdem`, these two environment variables have to be set as follows:

```
DIAsDEM_HOME=/home/kwinkler/diasdem/DIAsDEM.workbench21
JAVA_HOME=/usr/lib/j2sdk1.4.0_01
```

5. Make sure that both shell scripts `DIAsDEM.workbench21/bin/diasdemgui` and `DIAsDEM.workbench21/bin/diasdembatch` (e.g., in the directory `/home/kwinkler /diasdem`) are executable files:

```
/home/kwinkler/diasdem> chmod a+x
DIAsDEM.workbench21/bin/diasdemgui
/home/kwinkler/diasdem> chmod a+x
DIAsDEM.workbench21/bin/diasdembatch
```

6. Thereafter, the graphical user interface of DIAsDEM Workbench can be launched by executing the shell script `DIAsDEM.workbench21/bin/diasdemgui` (e.g., below the directory `/home/kwinkler/diasdem`).

```
/home/kwinkler/diasdem> DIAsDEM.workbench21/bin/diasdemgui
```

7. The command line batch script processor of DIAsDEM Workbench can be launched by executing the shell script `DIAsDEM.workbench21/bin/diasdembatch` (e.g., below the directory `/home/kwinkler/diasdem`). For example, the following statement has to be submitted at the command prompt to execute the DIAsDEM batch script `/home/kwinkler/diasdem/exampleScript.dsc` in verbose mode:

```
/home/kwinkler/diasdem> DIAsDEM.workbench21/bin/diasdembatch
/home/kwinkler/diasdem/exampleScript.dsc verbose
```

8. Finally, uncompress the text archive for case 1:

```
/home/kwinkler/diasdem> cd DIAsDEM.workbench21/data/samples/de/case1
/home/.../samples/de/case1> gunzip archive.tar.gz
/home/.../samples/de/case1> tar -xf archive.tar
```

2.3 Windows

1. Visit the Web page http://www.hypknowsys.org/diasdem and download the compressed archive file `DIAsDEM.workbench21.zip`. Note, this zipped archive includes exactly the same files and directories as `DIAsDEM.workbench21.tar.gz`.

2. Create a directory for DIAsDEM Workbench (e.g., `C:\Programs\diasdem`) and copy the file `DIAsDEM.workbench21.zip` into this directory.

3. Using for example WinZip that is available at http://www.winzip.com, extract the compressed file into the DIAsDEM-specific directory (e.g., `C:\Programs\diasdem`).

4. Using any common text editor, modify the two environment variables `JAVA_HOME` and `DIAsDEM_HOME` in both batch files `DIAsDEM.workbench21\bin\diasdemgui.bat` and `DIAsDEM.workbench21\bin\diasdembatch.bat` (e.g., below the directory `C:\Programs\diasdem`) according to your system. For example, if Java is installed in `C:\Programs\Java\j2re1.4.0_01` and if the file `DIAsDEM.workbench2.zip` was extracted in the directory `C:\Programs\diasdem`, these two environment variables have to be set as follows:

   ```
   DIAsDEM_HOME=C:\Programs\diasdem\DIAsDEM.workbench21
   JAVA_HOME=C:\Programs\Java\j2re1.4.0_01
   ```

5. Thereafter, DIAsDEM Workbench can be launched by opening Windows Explorer and double-clicking the batch file `DIAsDEM.workbench21\bin\diasdemgui.bat` (e.g., below the directory `C:\Programs\diasdem`).

6. The command line batch script processor of DIAsDEM Workbench can be launched by executing the batch file `DIAsDEM.workbench21\bin\diasdembatch.bat` (e.g., below the directory `C:\ Programs\diasdem`). For example, the following statement has to be submitted at the MS-DOS prompt to execute the DIAsDEM batch script `C:\Programs\diasdem\exampleScript.dsc` in verbose mode:

   ```
   C:\Programs\diasdem> DIAsDEM.workbench21\bin\diasdembatch.bat
   C:\Programs\diasdem\exampleScript.dsc verbose
   ```

7. Using Windows95 or Windows98 with standard system configurations, double-clicking `diasdemgui.bat` is likely to produce an "environment out of memory" error. In this case, the MS-DOS environment must be allocated more memory. Open Windows Explorer, right-click the icon of `diasdemgui.bat`, select *Properties*, click on the *Memory* tab and adjust *Initial Environment* from *Auto* to *2048*. After clicking on *OK* to commit the change, a PIF file (i.e., `diasdemgui.pif`) is created, which should afterwards be double-clicked to start DIAsDEM Workbench. The environment memory allocated to the batch file `diasdembatch.bat` must also be increased. Thereafter, DIAsDEM batch script files can be executed by launching the respective PIF file `diasdembatch.pif` instead of `diasdembatch.bat`:

   ```
   C:\Programs\diasdem> DIAsDEM.workbench21\bin\diasdembatch.pif
   C:\Programs\diasdem\exampleScript.dsc verbose
   ```

8. Finally, uncompress the text archive for case 1 using for example WinZip. The file `archive.tar.gz` is located in `DIAsDEM.workbench21\data\samples\de\case1`.

3 Case Study

3.1 Application Domain and Data Set

In Germany, each district court maintains a Commercial Register that contains important information about the companies in the court's district. According to law, many company activities like the establishment of branch offices, changes of share capital or mergers and acquisitions must be reported to the respective Register. Knowledge of these entries is indispensable for business activities, as they have both a right-confirmation and a right-generating effect according to the German Commercial Code.

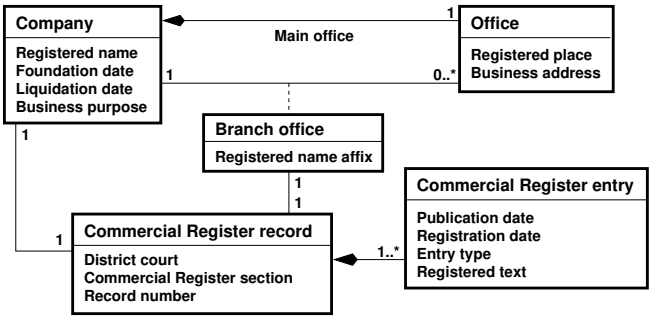

Figure 3.1: Simplified application domain (UML class diagram)

Commercial Register entries are made available to the public, since up-to-date knowledge about a company's affairs is essential to its (prospective) stakeholders. Three main categories of Commercial Register entries can be distinguished: foundation entries of new companies, update entries (e.g., changes in the managerial head of a company) and entries announcing that a company closes. The conceptual model the application domain is partly depicted as UML class diagrams in Figure 3.1 and Figure 3.2, respectively.

The directory `${SAMPLES_HOME}/de/case1` contains 1146 German Commercial Register entries published by the district court Potsdam in 1999 via its Web site (http://www.amtsgericht-potsdam.org). Each entry announces the foundation of a new company in the Potsdam district. Table 3.1 illustrates the contents of the file `${SAMPLES_HOME}/de/case1/file10780.training.txt`. Altogether, 985 text files (`*.training.txt`) are input to the first, interactive and iterative KDD phase of the DIAsDEM framework. The remaining 161 files having the extension `.application.txt` are automatically tagged in

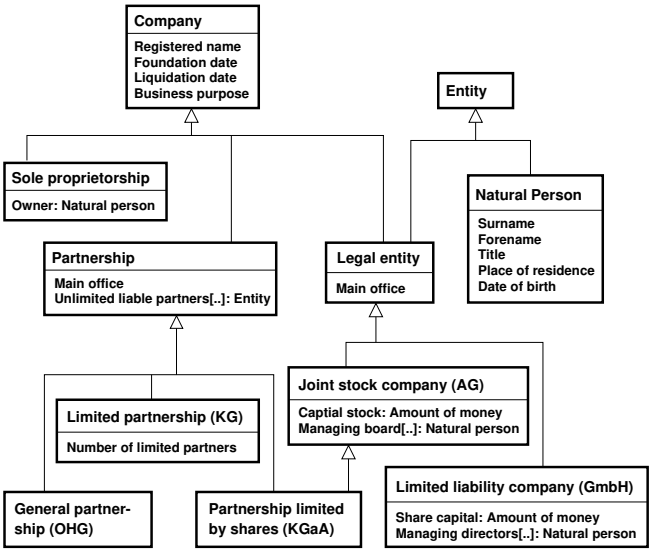

Figure 3.2: Simplified taxonomy of German companies (UML class diagram)

the second, batch-oriented application phase by applying the previously trained text unit clusterer. All text files are ISO-8856-1 encoded and have Unix-like line feeds. However, each entry is stored in a single line to avoid line feed related problems. Note, a concise list of relevant German vocabulary based on [PDV00] is available on page 118.

3.2 Text Pre-Processing in the KDD Phase

In the following screen-shots of this case study, the directory `/home/kwinkler/diasdem/DIAsDEM.workbench21` corresponds to `${DIAsDEM_HOME}`. `${PROJECT_HOME}` corresponds to `/home/kwinkler/diasdem/DIAsDEM.cases/tutorial/trainingProject` in the sections describing the interactive and iterative KDD phase of the DIAsDEM framework. All file and directory names are Unix/Linux-based in this case study.

Create a new local directory `${PROJECT_HOME}` on your machine, which can be used to store all files related to this case. Additionally, copy the entire directory template for KDD phase or so-called training projects into the directory `${PROJECT_HOME}`:

```
DIAsDEM.cases/tutorial> pwd
/home/kwinkler/diasdem/DIAsDEM.cases/tutorial
DIAsDEM.cases/tutorial> cp -R ../../DIAsDEM.workbench21/data/templates/trainingProject .
DIAsDEM.cases/tutorial> ls
trainingProject
```

Der Handel mit Waren aller Art sowie Import und Export. Der Dienstleistungsbereich bezieht sich auf Vermittlung, Beratung und Schulungen. Stammkapital: 50.000 DM. Gesellschaft mit beschränkter Haftung. Der Gesellschaftsvertrag ist am 18. April 1994 abgeschlossen und am 04. Dezember 1997 / 27. Mai 1998 abgeändert in §1 (Firma), §2 (Gegenstand) und §4 (Geschäftsführer). Durch Beschluss der Gesellschafterversammlung vom 17. November 1998 ist der Sitz der Gesellschaft von Maintal nach Damsdorf verlegt und der Gesellschaftsvertrag geändert in §1 (Firma und Sitz). Ist nur ein Geschäftsführer bestellt, so vertritt er die Gesellschaft allein. Sind mehrere Geschäftsführer bestellt, so wird die Gesellschaft durch zwei Geschäftsführer oder durch einen Geschäftsführer in Gemeinschaft mit einem Prokuristen vertreten. Einzelvertretungsbefugnis kann erteilt werden. Marion Marcella Adolph geb. Priester, 22.03.1957, Offenbach, ist zur Geschäftsführerin bestellt. Sie ist befugt, Rechtsgeschäfte mit sich selbst oder mit sich als Vertreter Dritter abzuschließen. Nicht eingetragen: Die Bekanntmachungen der Gesellschaft erfolgen im Bundesanzeiger.

Table 3.1: A German Commercial Register entry

The template for KDD phase projects comprises empty directories that are populated during this case study. For example, intermediate DIAsDEM documents are stored in the subdirectory `${PROJECT_HOME}/inputCollection`. At the end of this case study, semantically annotated XML documents that correspond to the training text documents are copied into the subdirectory `${PROJECT_HOME}/outputXmlDocuments`.

Make `${DIAsDEM_HOME}` your current working directory and start DIAsDEM Workbench by executing the shell script `${DIAsDEM_HOME}/bin/diasdemgui` on a Unix/Linux system. If you are working on a Windows machine, simply double-click the batch file `diasdemgui.bat`. Figure 1.2 on page 6 depicts DIAsDEM Workbench after startup.

The file `DIAsDEM.plugins` in `${DIAsDEM_HOME}` is a list of Java class names. Each class name corresponds to a DIAsDEM Workbench plug-in that is initialized during the start-up of DIAsDEM Workbench. `DIAsDEM.config` is another file created by DIAsDEM Workbench to store various, project-independent settings. Select *Tools* → *Options* to inspect and alter these settings, respectively. For example, you can choose your preferred Web browser as well as your preferred file editor in the tab *External Programs*. Be cautious when editing *GUI Properties*, because inappropriate parameter values might cause DIAsDEM Workbench to terminate abnormally.

```
DIAsDEM.cases/tutorial> ls trainingProject
applicationParameters kddProcessIteration1 outputGateDocuments outputSqlScripts
batchScripts         kddProcessIteration2 outputNeex21Files outputXmlDocuments
inputCollection      kddProcessIteration3 outputSampleFiles
DIAsDEM.cases/tutorial> cd /home/kwinkler/diasdem/DIAsDEM.workbench21
diasdem/DIAsDEM.workbench21> ls
DIAsDEM.plugins bin data doc lib src
diasdem/DIAsDEM.workbench21> bin/diasdemgui
diasdem/DIAsDEM.workbench21> ls
DIAsDEM.config DIAsDEM.plugins bin data doc lib src
diasdem/DIAsDEM.workbench21>
```

3.2.1 Starting the Batch Script Recorder

The entire interactive KDD phase should be recorded in a DIAsDEM batch script to enable subsequent automated semantic tagging. After slight adjustments, this recorded script can be employed in the application phase to automatically annotate new Commercial Register entries without time-consuming human intervention. DIAsDEM batch scripts are XML documents conforming to the XML document type definition which is listed in section 4.2 on page 92. Recording a batch script corresponds to saving all performed scriptable tasks along with their parameter settings in a file for subsequent task automation. After recording a script, it can be modified either in the dedicated *Batch Script Editor* (*Solutions* → *Batch Script Processing* → *Edit Batch Script*) or in any common text editor. Using the *Batch Script Editor*, new scripts can also be created from scratch as well. DIAsDEM Workbench allows executing batch scripts within the graphical user interface (*Solutions* → *Batch Script Processing* → *Execute Batch Script*) and on the Unix/Linux or MS-DOS command prompt.

Start recording all tasks performed hereafter by selecting *Solutions* → *Batch Script Processing* → *Record Batch Script* and thereafter clicking the button *Start*. Alternatively, tasks could be appended to an existing batch script by clicking the button *Open* and subsequently choosing the desired file. Figure 3.3 depicts DIAsDEM Workbench while recording a batch script. Stopping the current recording session as well as saving, editing and executing the recorded batch script is explained in sections 3.4.4 and 3.5.

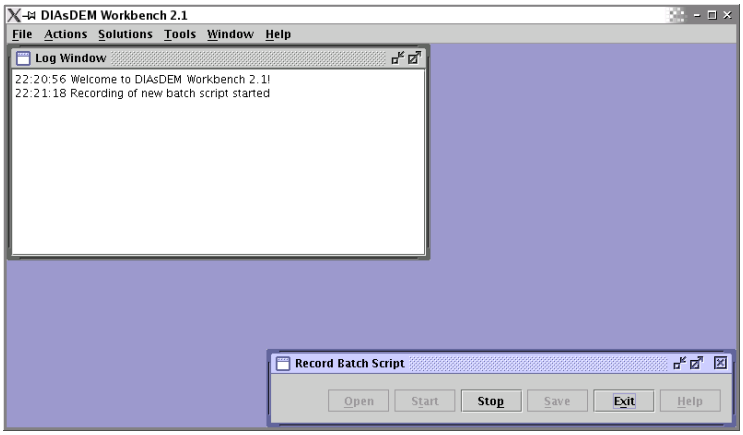

Figure 3.3: DIAsDEM Workbench 2.1 after starting the batch script recorder

3.2.2 Creating a New Project

Training a text unit clusterer for Commercial Register entries in the KDD phase of our framework constitutes an independent project. All input, intermediate and output files associated with a project should be stored in a single directory (i.e., in `${PROJECT_HOME}`) or its subdirectories to ensure a properly organized file system. Additionally, DIAsDEM Workbench stores project-related data such as most recently used parameter values in a DIAsDEM project file with the extension `.dpr` in the directory `${PROJECT_HOME}`. A project must thus either be created or opened before any DIAsDEM task from the menu *Actions* such as importing texts or clustering text unit vectors can be performed at all. Consequently, select *File* → *New Project* to create a new DIAsDEM project and enter the following parameters:

Parameter	Value
Project Name	`Tutorial - KDD Phase`
Project File Name	`${PROJECT_HOME}/project.dpr`
Project Directory	`${PROJECT_HOME}`
Parameter Directory	`${PARAMETER_HOME}`

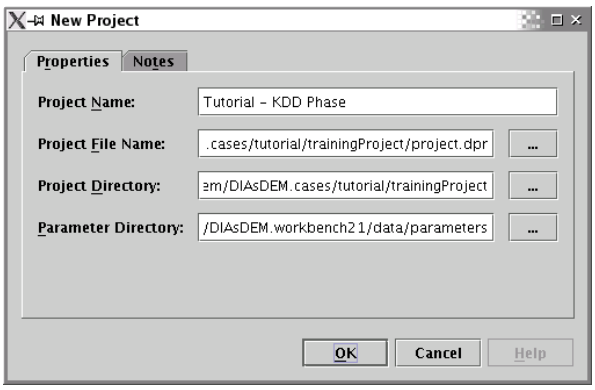

Figure 3.4: *New Project* window

Make sure to replace the abbreviations `${PROJECT_HOME}` and `${PARAMETER_HOME}` with the corresponding directories according to your individual installation of DIAsDEM Workbench. Recall, the abbreviation `${PARAMETER_HOME}` denotes the directory `${DIAsDEM_HOME}/data/parameters` in this case study. Therefore, it depends on the installation directory `${DIAsDEM_HOME}` of DIAsDEM Workbench on your machine. Files and directories can always be chosen by clicking the button "..." beside the respective text field and afterwards selecting the desired file from the file dialog. The Java-based file dialog can also be used to create new and rename existing directories, respectively.

Figure 3.4 shows the *New Project* window before clicking on *OK*. After clicking the *OK* button, a new project labeled "Tutorial - KDD Phase" is created and immediately opened. Note, the title bar of DIAsDEM Workbench window always indicates the actually opened project.

New Project: Summary

Module: *File* → *New Project* or *Actions* → *Project Management* → *New Project*

Use Case: The user wants to create a new DIAsDEM project in order to semantically annotate text documents.

Prerequisites: A dedicated local *Project Directory* must have been created for storing project-related files and subdirectories. Template directories for *Project Directory* are provided in `${DIAsDEM_HOME}/data/templates`.

Result: A file *Project File Name* that contains project-related meta-data is created in *Project Directory*. Additionally, the project properties *Project Name*, *Project Notes*, *Absolute File Name of Project File*, *Absolute File Name of Project Directory* and *Absolute File Name of Parameter Directory* are set.

Remarks: The new project can be opened by selecting *File* → *Open Project* and thereafter choosing *Project File Name*. Opened projects can be closed by selecting *File* → *Close Project*.

New Project: Parameters

Project Name: Name of the new DIAsDEM project that might include blank spaces; default value: <DefaultProjectName>

Project File Name: Valid local file name of project file to be created in *Project Directory*; file extension: `.dpr`

Project Directory: Existing local directory that should contain *Project File Name* and all project-related files; corresponds to `${PROJECT_HOME}` in this case study

Parameter Directory: Existing local directory whose subdirectories contain default parameter files of default DIAsDEM tasks; corresponds to the directory `${PARAMETER_HOME}` in this case study

3.2.3 Creating a Document Collection

Text documents are considered a collection of related documents, if they belong to the same application domain and if they should be semantically tagged. DIAsDEM Workbench is only capable of processing one collection at a time. Each document collection is identified by a specific collection file with extension `.dcf`. This *Collection File* contains meta-data about the corresponding archive as well as references to all DIAsDEM documents constituting the collection. Before texts can be imported into a collection for subsequent processing, a new collection must be created. Therefore, select *Actions* → *Prepare Data Set* → *Create Document Collection* and input the following parameters:

Parameter	Value
Collection Name	`Tutorial (Training Documents)`
Collection File	`${PROJECT_HOME}/collection.dcf`
Collection Directory	`${PROJECT_HOME}/inputCollection`
Documents Per Volume	`1`

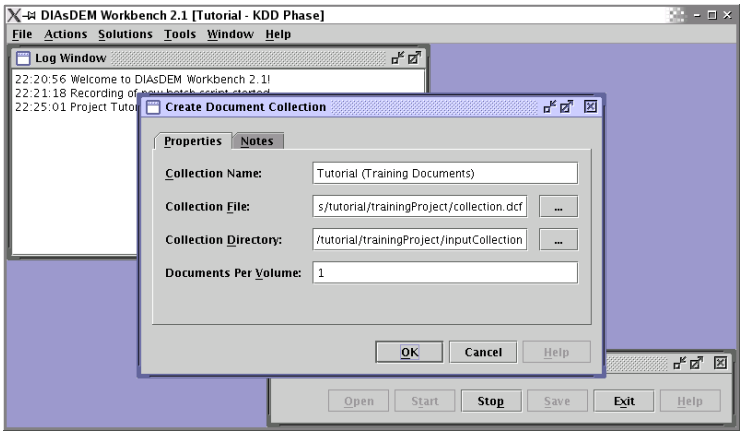

Figure 3.5: *Create Document Collection* dialog

Click on *OK* to create a new collection file. Thereafter, the directory `${PROJECT_HOME}` contains the files `collection.dcf` and `collection.dcf.files`. In the remainder of this case study, `collection.dcf` is referred to as *Collection File*. It uniquely identifies the corresponding archive of Commercial Register entries and contains relevant meta-data. Each *Collection File* is accompanied by an auxiliary file with extention `.dcf.files`, e.g. `collection.dcf.files`. This file only contains absolute file names of DIAsDEM document volume files comprising the collection. Both *Collection File* and its auxiliary file should neither be modified nor deleted manually.

The default implementation of DIAsDEM Workbench is file-based. Each DIAsDEM collection consists of multiple so-called DIAsDEM volumes that in turn include one or

more DIAsDEM documents. In this case study, one document is stored per volume file to improve clarity. Each DIAsDEM document contains the initially imported text, both original and processed text units (e.g., sentences) as well as possibly rollback text units and structured meta-data. In the default implementation, DIAsDEM volumes are XML documents that conform to the XML document type definition listed in section 4.1 on page 91.

Create Document Collection: Summary

Module: *Actions → Prepare Data Set → Create Document Collection*

Use Case: The user wants to create a new DIAsDEM document collection in order to semantically annotate text documents.

Prerequisites: A dedicated local *Collection Directory* must have been created for storing DIAsDEM documents in DIAsDEM volumes.

Result: A new DIAsDEM collection is created in *Collection Directory*. It is labeled *Collection Name* and can be referenced by *Collection File*. Additionally, the project properties *Default Collection File* and *Default Collection Directory* are set.

Remarks: All subsequent processing modules of DIAsDEM Workbench require a specific *Collection File* as an input parameter. Text documents can hereafter be imported into the new collection by selecting *Actions → Prepare Data Set → Import Plain Text Files*.

Create Document Collection: Parameters

Collection Name: Name of the new document collection that might include blank spaces

Collection File: Valid local file name of new collection file; file extension: `.dcf`; proposed value: `${PROJECT_HOME}/collection.dcf`

Collection Directory: Existing local directory that subsequently contains DIAsDEM volume files; default value: project property *Default Collection Directory*

Documents Per Volume: Number of DIAsDEM documents stored in a single DIAsDEM volume file; default value: 10

3.2.4 Importing Plain Text Files

Commercial Register entries provided for case 1 are plain text files. They can be imported into the new, up to now empty document collection by selecting *Actions* → *Prepare Data Set* → *Import Plain Text Files*. Please provide the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
Text File Directory	`${SAMPLES_HOME}/de/case1`
	`Disabled: Include Subdirectories`
File Name Extension	`.training.txt`

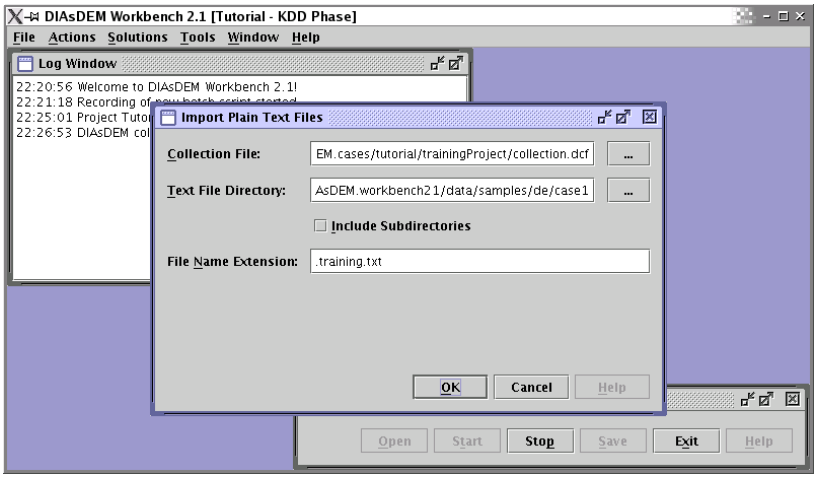

Figure 3.6: *Import Plain Text Files* dialog

Click the *OK* button to start importing altogether 985 text files with *File Name Extension* from *Text File Directory*. These text files are imported into the new DIAsDEM collection referenced by *Collection File*. Check the contents of the subdirectory `${PROJECT_HOME}/inputCollection`:

```
DIAsDEM.cases/tutorial> ls -l inputCollection | more
-rw-r--r--   1 kwinkler users      1503 2003-08-19 19:32 DefaultDIAsDEMvolume.dtd
-rw-r--r--   1 kwinkler users      1688 2003-08-19 19:32 volume100000.xml
-rw-r--r--   1 kwinkler users      2035 2003-08-19 19:32 volume100001.xml
...
-rw-r--r--   1 kwinkler users      2035 2003-08-19 19:32 volume100984.xml
```

`DefaultDIAsDEMvolume.dtd` is the XML document type definition of volume files. For example, the DIAsDEM document that corresponds to the Commercial Register

entry listed in Table 3.1 (i.e., ${SAMPLES_HOME}/de/case1/file10780.training.txt) is stored in volume file ${PROJECT_HOME}/inputCollection/volume100878.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0">
    <MetaData>
      <Name>DiasdemDocumentID</Name>
      <Content>/home/kwinkler/.../inputCollection/volume100878.xml:0</Content>
    </MetaData>
    <MetaData>
      <Name>SourceFile</Name>
      <Content>/home/kwinkler/.../de/case1/file10780.training.txt</Content>
    </MetaData>
    <OriginalText>Der Handel mit Waren aller Art sowie Import und Export. Der
    Dienstleistungsbereich bezieht sich auf ... Bundesanzeiger.</OriginalText>
  </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

In the default implementation of DIAsDEM Workbench, the meta-data attribute DiasdemDocumentID is created by concatenating the absolute file name of the volume file, a colon and the index of the respective document (0, 1, ...) within its volume. Note, the original text will not be modified by any subsequently performed tasks.

Import Plain Text Files: Summary

Module: *Actions → Prepare Data Set → Import Plain Text Files*

Use Case: The user wants to employ DIAsDEM Workbench to semantically annotate text documents that are stored in plain text files within a single local directory or its subdirectories.

Prerequisites: Each text file contains exactly one text document.

Result: Files in *Text File Directory* whose file names end with *File Name Extension* are imported into *Collection File* and stored as DIAsDEM volumes conforming to XML DTD DefaultDIAsDEMvolume.dtd described in section 4.1 on page 91. Additionally, the project properties *Default Collection File* and *Default Text File Directory* are set.

Remarks: Texts can be imported from one directory after the other. Alternatively, the user might implement a specialized DIAsDEM Workbench plug-in to import text documents into a collection. In this case, additional structured meta-data can be included in DIAsDEM documents as well.

Import Plain Text Files: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Text File Directory: Existing local directory that contains text files to be imported; if *Include Subdirectories* is enabled, text files stored in subdirectories of *Text File Directory* are also imported; default value: project property *Default Text File Directory*

File Name Extension: Names of text files to be imported must have this file name extension; default value: `.txt`

3.2.5 Creating Text Units

After importing texts into a collection file, the data pre-processing phase starts with identifying and separating text units. In this case study, sentences of Commercial Register entries correspond to text units. Hence, only sentences are semantically annotated by DIAsDEM Workbench in the course of this case study. To proceed, select *Actions* → *Prepare Data Set* → *Create Text Units* and type in the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
Text Unit Algorithm	`Heuristic Sentence Identifier`
Abbreviations File	`${PARAMETER_HOME}/createTextUnits/de/AbbreviationsDE.txt`
Full Stop Regex File	`${PARAMETER_HOME}/createTextUnits/de/FullStopRegexDE.txt`
Replaced Full Stops	`Keep Asterisks for Tokenization`
Text Units Layer	`Create or Replace Default Text Units Layer`

Click the *OK* button to identify and separate altogether 9,254 sentences in DIAsDEM documents. The Heuristic Sentence Identifier first replaces full stops in abbreviations (e.g., "z.B.") listed in *Abbreviations File* with asterisks. However, only abbreviations are replaced that either occur at the beginning of the text or that follow one of certain special characters (i.e., blank space and `(),;:/-'"`). Thereafter, regular expressions contained in *Full Stop Regex File* are matched against the original text. These regular expressions match full stops that are no sentence boundaries (e.g., "01.01.2002") and replace all matches with asterisks as well. Both textual parameter files can be edited in order to include additional domain knowledge.

Using for example the built-in, but rather simple *Tools* → *Miscellaneous* → *XML Document Viewer*, have a look at the contents of DIAsDEM volume file `${PROJECT_HOME}/inputCollection/file10878.txt.xml`. Its DIAsDEM document has been extended by the new section `<TextUnitsLayer>`. This section consists of two subsections, namely `<OriginalTextUnits>` whose elements `<textUnit>` mark up single, original sentences

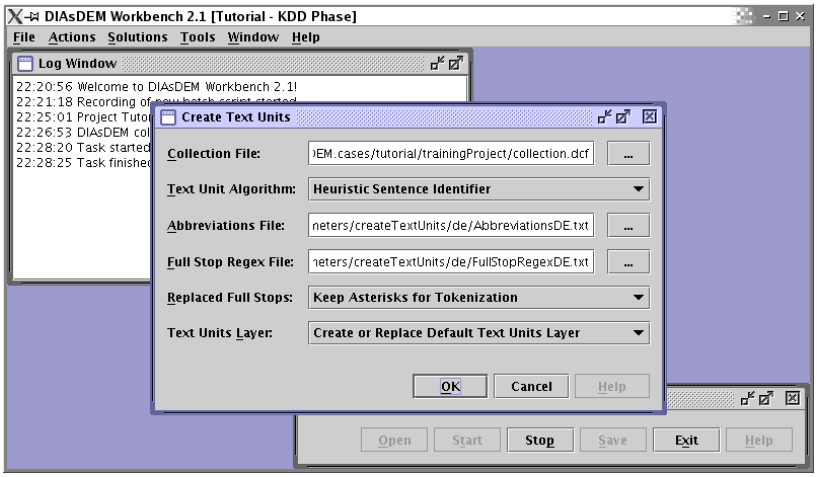

Figure 3.7: *Create Text Units* dialog

and `<ProcessedTextUnits>` whose elements `<textUnit>` mark up single, processed sentences. All sentences of the former subsection are retained unmodified, whereas the contents of the latter subsection are processed by tasks performed hereafter.

In general, the DIAsDEM framework supports multiple structural views on texts by introducing the notion of "text units layers". For example, one might define three text unit layers in order to semantically annotate each document at the text level, at the paragraph level, and at sentence level. However, DIAsDEM Workbench 2.1 does not fully support nested semantic tagging of text documents. Instead, plug-ins always process text units associated with the default layer and output accordingly tagged XML documents only. In this case study, the default text units layer 0 provides a structural view on sentences of each document. The index of the default text units layer is determined by the project property *Index of Default Active Text Units Layer in DIAsDEM Documents*. It can be modified in the *Project Properties* tab of the *Tools* → *Options* dialog.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0"> ...
    <TextUnitsLayer TextUnitsLayerID="0" TextUnitsDescription="Algorithm:
    HEURISTIC_SENTENCE_IDENTIFIER">
      <OriginalTextUnits>
        <OriginalTextUnit TextUnitID="0" BeginIndex="0" EndIndex="55">Der
        Handel mit Waren aller Art sowie Import und Export.</OriginalTextUnit>
        ... <OriginalTextUnit TextUnitID="2" BeginIndex="138" EndIndex="162">
        Stammkapital: 50.000 DM.</OriginalTextUnit> ... <OriginalTextUnit
        TextUnitID="9" BeginIndex="867" EndIndex="963"> Marion Marcella Adolph
```

```
      geb. Priester, 22.03.1957, Offenbach, ist zur Geschäftsführerin
      bestellt.</OriginalTextUnit> ...
    </OriginalTextUnits>
    <ProcessedTextUnits>
      <ProcessedTextUnit TextUnitID="0">Der Handel mit Waren aller Art
      sowie Import und Export.</ProcessedTextUnit> ... <ProcessedTextUnit
      TextUnitID="2">Stammkapital: 50*000 DM.</ProcessedTextUnit> ...
      <ProcessedTextUnit TextUnitID="9">Marion Marcella Adolph geb* Priester,
      22*03*1957, Offenbach, ist zur Geschäftsführerin bestellt.
      </ProcessedTextUnit> ...
    </ProcessedTextUnits>
  </TextUnitsLayer>
 </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

Asterisks that occur within the section `<ProcessedTextUnits>` either replace full stops in an abbreviation listed in *Abbreviations File* (e.g., `geb.`) or full stops matched by a regular expression in *Full Stop Regex File*. For example, the date literal `22.03.1957` is matched by the expression `([0-9]{1,2})\.([\ ]*[0-9]{1,2})\.([\ ]*[0-9]{2,4})`. Therefore, the original date literal `22.03.1957` has been replaced by the corresponding replacement string `$1\*$2\*$3` which results in `22*03*1957`.

Create Text Units: Summary

Module: *Actions* → *Prepare Data Set* → *Create Text Units*

Use Case: The user must pre-process imported texts as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives. Creating text units is pre-processing phase 1 of 4.

Prerequisites: Texts must have been imported into the DIAsDEM collection.

Result: The specified text units layer of each document is created or replaced. Elements of the sections `<OriginalTextUnits>` and `<ProcessedTextUnits>` mark up either single sentences (*Heuristic Sentence Identifier*) or the entire text (*Text as a Single Text Unit*). If *Create or Replace Default Text Units Layer* is enabled, an existing default layer are completely replaced. Additionally, the project properties *Default Collection File*, *Default Abbreviations File* and *Default Full Stop Regex File* are set and updated, respectively.

Remarks: Creating text units is a prerequisite for the remaining pre-processing phases 2 (i.e., tokenization) through 4 (i.e., lemmatization). Take the following side effect into consideration: All asterisks that occur in imported texts are eventually replaced by full stops.

Create Text Units: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Text Unit Algorithm: If the recommended option *Heuristic Sentence Identifier* is enabled, this module heuristically identifies sentences terminated by full stops for subsequent semantic annotation. If the option *Text as a Single Text Unit* is enabled, the entire text is marked up as a single text unit. In the latter case, the text is annotated by one XML tag only.

Abbreviations File: Valid local file name of existing file that contains known abbreviations in the format described in section 4.3.1 on page 92; file extension: `.txt`; default value: project property *Default Abbreviations File*

Full Stop Regex File: Valid local file name of existing file that contains regular expressions in the format described in section 4.3.1 on page 92; file extension: `.txt`; default value: project property *Default Full Stop Regex File*

Replaced Full Stops: If the recommended option *Keep Asterisks for Tokenization* is enabled, asterisks that replace full stops are retained for usage in the subsequent tokenization phase. Otherwise, asterisks are replaced by full stops before this module terminates.

Text Units Layer: If the recommended option *Create or Replace Default Text Units Layer* is enabled, the default layer is created and replaced, respectively. Otherwise, an additional text units layer is added to each document. Note, the default text unit layer index is determined by the project property *Index of Default Active Text Units Layer in DIAsDEM Documents*.

3.2.6 Tokenizing Text Units

After creating text units, tokenizing them constitutes the second pre-processing phase. During tokenization, text units are decomposed into individual words and tokens, respectively. In addition, text units are normalized in order to map for example date literals appearing in many formats (e.g., "1 Jan 2003" and "1.1.2003") onto a canonical representation (e.g., "01.01.2003"). Moreover, multi-token terms that contain blank spaces (e.g., "for example") are identified to subsequently process them as single tokens. Select *Actions* → *Prepare Data Set* → *Tokenize Text Units* and input the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
Tokenize Regex File	`${PARAMETER_HOME}/createTextUnits/de/TokenizeRegexDE.txt`
Normalize Regex File	`${PARAMETER_HOME}/createTextUnits/de/NormalizeRegexDE.txt`
Multi Token Words File	`${PARAMETER_HOME}/createTextUnits/de /neex21/MultiTokenWordsDE.txt`

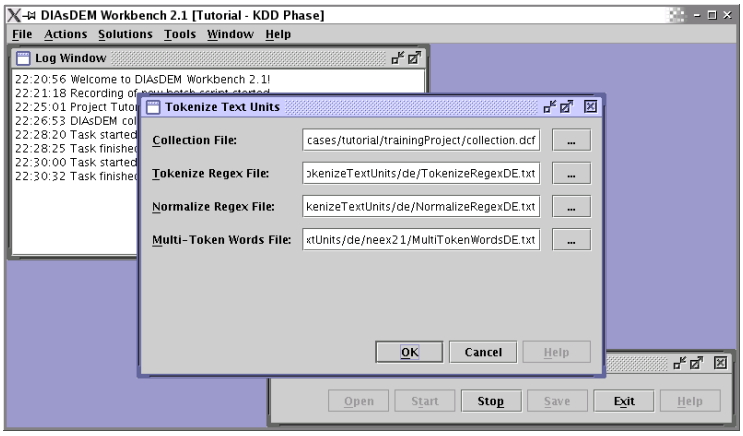

Figure 3.8: *Tokenize Text Units* dialog

Click on *OK* to tokenize and normalize text units as well as to identify and replace multi-token terms. All three processing steps can be fully parameterized by editing regular expressions or multi-token terms in the respective parameter files. The heuristic normalization algorithm does not separate asterisks from their surrounding characters, because asterisks correspond to previously replaced full stops. Therefore, *Tokenize Regex File* should not include regular expressions matching asterisks. Text units are normalized by applying regular expressions and substituting matching sequences with the corresponding replacement string. Furthermore, blank spaces in multi-token terms listed in *Multi Token Words File* (e.g., "for example") are replaced by underscores (e.g., "for_example") to create a single token. Finally, all asterisks that occur in the section `<ProcessedTextUnits>` are replaced by full stops.

Check the contents of file `${PROJECT_HOME}/inputCollection/volume100878.xml`: Firstly, the section `<ProcessedTextUnits>` of this DIAsDEM document has been updated. After tokenization, its elements `<ProcessedTextUnit>` mark up single, tokenized and normalized sentences. Secondly, a section `<RollbackTextUnits>` has been added to the default text units layer of each document. By executing *Actions* → *Miscellaneous* → *Rollback Processed Text Units*, the contents of `<RollbackTextUnits>` can be copied into `<ProcessedTextUnits>` in order to undo the effects of one preceding task. Note, DIAsDEM Workbench 2.1 supports three different rollback options: No backup at all (0), rollback of the immediately preceding task (1, default) and rollback of any preceding task (2). The active rollback policy is determined by the project property *Rollback Option (0, 1, 2) for ProcessedTextUnits in DIAsDEM Documents*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
```

```
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0"> ...
    <TextUnitsLayer TextUnitsLayerID="0" TextUnitsDescription="Algorithm:
    HEURISTIC_SENTENCE_IDENTIFIER"> ...
      <ProcessedTextUnits>
        <ProcessedTextUnit TextUnitID="0">Der Handel mit Waren aller Art sowie
        Import und Export .</ProcessedTextUnit> ... <ProcessedTextUnit
        TextUnitID="2">Stammkapital : 50000 DEM .</ProcessedTextUnit>
        <ProcessedTextUnit TextUnitID="3">Gesellschaft_mit_beschränkter_Haftung
        .</ProcessedTextUnit> ... <ProcessedTextUnit TextUnitID="9">Marion
        Marcella Adolph geb. Priester , 22.03.1957 , Offenbach , ist zur
        Geschäftsführerin bestellt .</ProcessedTextUnit> ...
      </ProcessedTextUnits>
      <RollbackTextUnits RollbackID="0">
        <ProcessedTextUnit TextUnitID="0">Der Handel mit Waren aller Art
        sowie Import und Export.</ProcessedTextUnit> ... <ProcessedTextUnit
        TextUnitID="2">Stammkapital: 50*000 DM.</ProcessedTextUnit> ...
        <ProcessedTextUnit TextUnitID="9">Marion Marcella Adolph geb* Priester,
        22*03*1957, Offenbach, ist zur Geschäftsführerin bestellt.
        </ProcessedTextUnit> ...
      </RollbackTextUnits>
    </TextUnitsLayer>
  </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

The module works as follows: Firstly, regular expressions listed in *Tokenization Regex File* are matched against each processed text unit. For example, the character subsequence "`e.`" of the string "`This is a sentence. There`" is matched by the regular expression `(\S)(\.|\!|\?)`. This matching character subsequence is thus substituted by the replacement string `$1\ $2`, which results in the following tokenized text: "`This is a sentence . There`". Secondly, regular expressions listed in *Normalization Regex File* are matched against processed text units. Analogously, matching character subsequences are substituted by the corresponding replacement string. Thirdly, multi-token terms included in *Multi Token Words File* are looked up in text units. Identified multi-token terms are reduced to single tokens by replacing their inner blank spaces with underscores.

Tokenize Text Units: Summary

Module: *Actions → Prepare Data Set → Tokenize Text Units*

Use Case: The user must pre-process all imported texts as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives. Tokenizing text units is pre-processing phase 2 of 4.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. Text units should have been created in the DIAsDEM collection.

Result: Elements of the section `<ProcessedTextUnits>` mark up tokenized and normalized text units. Inner blank spaces in multi-token terms have been replaced with underscores and all asterisks have been replaced with full stops. Additionally, the project properties *Default Collection File*, *Default Normalize Regex File*, *Default Tokenize Regex File* and *Default Multi Token Words File* are set and updated, respectively.

Tokenize Text Units: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Tokenize Regex File: Valid local file name of existing file that contains regular expressions in the format described in section 4.3.2 on page 93; file extension: `.txt`; default value: project property *Default Tokenize Regex File*;

Normalize Regex File: Valid local file name of existing file that contains regular expressions in the format described in section 4.3.2 on page 93; file extension: `.txt`; default value: project property *Default Normalize Regex File*

Multi Token Words File: Valid local file name of existing file that contains multi-token terms in the format described in section 4.3.2 on page 93; file extension: `.txt`; default value: project property *Default Multi Token Words File*

3.2.7 Replacing Named Entities with NEEX 2.1

After creating and tokenizing text units, identifying named entities and replacing them with placeholders constitutes the third pre-processing phase. Extracted named entities might thereafter serve as attribute values in semantic XML tags. For example, "Karsten Winkler" is an instance of named entity type "person" and "Leipzig" instantiates the NE type "place". Based on lists and regular expressions, the Named Entity Extractor of DIAsDEM Workbench 2.1 is capable of identifying named entities of the following types: "person", "company", "company_relocation", "number", "date", "time", "amount_of_money", "paragraph", "email", "url", "organization_id", "document_id", "court", "postal_code", "isin", and "wkn".

NEEX 2.1 is the successor of NEEX 2.0. The latter is still available and thus briefly described in section 3.6.1. Given appropriately customized parameter files, the new module should outperform its predecessor. Above all, NEEX 2.1 requires less handmade rules for instantiating composite named entities (e.g., persons) from basic named entities such as forename, surnames and academic titles. However, only NEEX 2.0 includes a heuristic algorithm for detecting canonical forms of persons and companies within one document. For example, the canonical form of a certain person replaces all occurrences

of named entity type "person" that probably refer to the same real-world person such as "Karsten Winkler" and "K. Winkler". Start using NEEX 2.1 by selecting *Actions* → *Prepare Data Set* → *Replace Named Entities 2.1* and typing in the following parameters:

Parameter	Value
Collection File	${PROJECT_HOME}/collection.dcf
Regex NE File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/Case1234RegexNE.txt
Organization Indicators File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/OrganizationIndicatorsDE.txt
Organization Suffixes File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/OrganizationSuffixesDE.txt
Organization Affixes File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/OrganizationAffixesDE.txt
Organizations File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/OrganizationsDE.txt
Organizations as Meta Data	
Place Indicators File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/PlaceIndicatorsDE.txt
Places File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/PlacesDE.txt
Place Affixes File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/PlaceAffixesDE.txt
Person Name Indicators File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/PersonNameIndicatorsDE.txt
Titles File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/TitlesDE.txt
Forenames File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/ForenamesDE.txt
Middle Initials File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/MiddleInitialsDE.txt
Surnames File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/SurnamesDE.txt
Surname Suffixes File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/SurnameSuffixesDE.txt
Name Affixes File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/NameAffixesDE.txt
Professions File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/ProfessionsDE.txt
Composite NE File	${PARAMETER_HOME}/replaceNamedEntities/de/neex21/Case1234CompositeNE.txt
Debugging Files Directory	${PROJECT_HOME}/outputNeex21Files

Click the *OK* button to identify and replace named entities in processed text units. Thereafter, open the file ${PROJECT_HOME}/inputCollection/volume100878.xml. The default text units layer of its DIAsDEM document has been extended by the new section <NamedEntities> whose elements <NamedEntity> mark up extracted named entities. Elements of the section <ProcessedTextUnits> now mark up tokenized and normalized sentences, which might contain named entity placeholders. Each placeholder tag

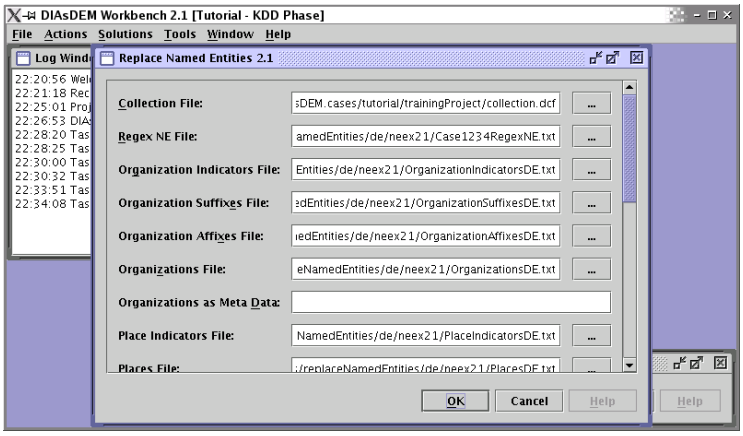

Figure 3.9: *Replace Named Entities 2.1* dialog

`<NeRef>` references its associated named entity in section `<NamedEntity>` via the attribute `NeID`. Note, the section `<RollbackTextUnits>` has also been updated. Hence, you might undo the effects of the performed named entity extraction task if necessary.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0"> ...
    <TextUnitsLayer TextUnitsLayerID="0" TextUnitsDescription="Algorithm:
    HEURISTIC_SENTENCE_IDENTIFIER"> ...
      <ProcessedTextUnits>
        <ProcessedTextUnit TextUnitID="0">Der Handel mit Waren aller Art sowie
        Import und Export .</ProcessedTextUnit> ... <ProcessedTextUnit
        TextUnitID="2">Stammkapital : <NeRef NeID="0" /> .</ProcessedTextUnit>
        <ProcessedTextUnit TextUnitID="3">Gesellschaft_mit_beschränkter_Haftung
        .</ProcessedTextUnit> ... <ProcessedTextUnit TextUnitID="9"><NeRef
        NeID="16" />, ist zur Geschäftsführerin bestellt .</ProcessedTextUnit> ...
      </ProcessedTextUnits> ...
      <NamedEntities>
        <NamedEntity NeID="0" NeType="amount_of_money">50000 DEM</NamedEntity> ...
        <NamedEntity NeID="12" NeType="date">22.03.1957</NamedEntity>
        <NamedEntity NeID="13" NeType="place">Offenbach</NamedEntity>
        <NamedEntity NeID="14" NeType="person_name">Marion Marcella Adolph
        </NamedEntity>
        <NamedEntity NeID="15" NeType="person_name">Priester</NamedEntity>
        <NamedEntity NeID="16" NeType="person">16|null|person|null|Marion
        Marcella Adolph|null|null|Priester|22.03.1957|null|null|null|Offenbach
        |null|null</NamedEntity>
      </NamedEntities>
```

```
    </TextUnitsLayer>
  </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

NEEX 2.1 can be fully parameterized by editing the corresponding parameter files. The heuristic Named Entity Extractor of DIAsDEM Workbench works as follows:

1. Firstly, regular expressions listed in *Regex NE File* are matched against processed text units to identify instances of basic named entities "number", "date", "time", "amount_of_money", "paragraph", "email", "url", "organization_id", "document_id", "court", "postal_code", "isin", and "wkn". For example, named entity 0 is an instance "50000 DEM" of named entity type "amount_of_money". This instance occurs in the tokenized and normalized sentence "Stammkapital : 50000 DEM ."

2. Secondly, instances of basic named entity type "organization" are identified by employing the parameter files *Organization Indicators File*, *Organization Suffixes File*, *Organization Affixes File*, and *Organizations File*. The latter file contains complete, but tokenized names of organizations, which are extracted in any case. Each document might optionally contain a single tokenized organization name as a metadata attribute value. If this attribute name is given as parameter *Organizations as Meta Data*, its value is always extracted as an organization in the corresponding text only. NEEX 2.1 searches organization suffixes and looks backwards for valid organization indicators to instantiate an "organization". Suffixes that are immediately preceded by an indicator are not instantiated as organizations. Thereafter, organization candidates are extended, if they are followed by an organization affix. For debugging purposes, extracted organizations are highlighted in the output file `Neex21_Organizations.html` located in `${PROJECT_HOME}/outputNeex21Files`.

3. Thirdly, terms instantiating the basic named entity type "place" are extracted using the parameter files *Place Indicators File*, *Places File*, and *Place Affixes File*. NEEX 2.1 looks up all tokens in the dictionary of known places and extends place candidates, if they are immediately followed by another place candidate or a place affix. However, place candidates are only instantiated as a "place", if they are directly preceded by a place indicator. For debugging purposes, identified places are highlighted in the output file `Neex21_Places.html`.

4. Thereafter, parameter files *Person Name Indicators File*, *Titles File*, *Forenames File*, *Middle Initials File*, *Surnames File*, *Surname Suffixes File* and *Name Affixes File* are used to discover instances of basic named entity "person_name". Each instance of "person_name" is a continuous sequence of tokens instantiating the following basic named entity types: academic "title", "forename", "middle_initial", "surname", and "name_affix". Composite forenames and surnames that include

the character "-" are considered as well. Person name candidates are extended, if they are immediately followed by a name affix or a capitalized term ending with a surname suffix listed in *Surname Suffixes File*. However, single-token person name candidates are only instantiated, if they are directly preceded by a person name indicator. For debugging purposes, all identified person names are highlighted in the output file `Neex21_PersonNames.html`. If the optional parameter *Professions File* is specified, instances of basic named entity type "profession" are identified in text units, which contain at least one instance of NE type "person_name". Discovered professions are highlighted in the output file `Neex21_Professions.html`.

In the example above, "Marion Marcella Adolph" and "Priester" are instances of named entity type "person_name". In general, each token might simultaneously instantiate various basic named entities. For example, the term "Adolph" could be either a surname or a forename. In contrast to its predecessor, NEEX 2.1 uses various heuristics and after all employs *Person Name Indicators File* and *Place Indicators File* at the respective stage to decide whether a certain token probably instantiates a "place" or is rather a partial "person_name".

5. Finally, NEEX-specific rules provided in *Composite NE File* are applied to processed text units that contain basic named entities. This step aims at identifying instances of composite named entities "person", "company", and "company_relocation". Each composite named entity consists of basic named entities that occur in a context described by rules in *Composite NE File*. For example, composite named entities of type "person" can be constructed from basic named entities such as "person_name", "date", and "place". If a composite named entity is identified, both textual contents and basic named entity placeholders matched by the rule are substituted by the corresponding composite named entity placeholder. For debugging purposes, remaining basic (green font) as well as identified composite (red font) named entities are highlighted in the output file `Neex21_CompositeNEs.html`.

   For example, the rather intuitive, NEEX-specific rule "<<person_name>> geb. <<person_name>> , <<date>> , <<place>>", which is listed in *Composite NE File*, triggers the instantiation of named entity 16. This named entity maps the token sequence "Marion Marcella Adolph geb. Priester , 22.03.1957 , Offenbach" onto an instance of composite named entity type "person".

Replace Named Entities 2.1: Summary

Module: *Actions → Prepare Data Set → Replace Named Entities 2.1*

Use Case: The user must pre-process all imported texts as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives. Identifying and replacing named entities is pre-processing phase 3 of 4.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. Elements `<ProcessedTextUnit>` must not contain previously inserted named entity references `<NeRef>`. Text units should have been created and tokenized in the DIAsDEM collection.

Result: Elements of section `<ProcessedTextUnits>` mark up text units containing placeholders for extracted named entities. The named entities are stored in elements `<NamedEntity>` of section `<NamedEntities>`. Additionally, the project properties *Default Collection File*, *NEEX 2.1: Default Regex NE File*, *NEEX 2.1: Default Organization Indicators File*, *NEEX 2.1: Default Organization Suffixes File*, *NEEX 2.1: Default Organization Affixes File*, *NEEX 2.1: Default Organizations File*, *NEEX 2.1: Default Organizations Meta Data Attribute in DIAsDEM Documents*, *NEEX 2.1: Default Place Indicators File*, *NEEX 2.1: Default Places File*, *NEEX 2.1: Default Place Affixes File*, *NEEX 2.1: Default Person Name Indicators File*, *NEEX 2.1: Default Titles File*, *NEEX 2.1: Default Forenames File*, *NEEX 2.1: Default Middle Initials File*, *NEEX 2.1: Default Surnames File*, *NEEX 2.1: Default Surname Suffixes File*, *NEEX 2.1: Default Name Affixes File*, *NEEX 2.1: Default Professions File*, *NEEX 2.1: Default Composite NE File* and *NEEX 2.1: Default Directory of Debugging HTML Files* are set and updated on request, respectively.

Replace Named Entities 2.1: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Regex NE File: Valid local file name of existing file that contains regular expressions for identifying basic named entities (e.g., of type "amount_of_money") in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Regex NE File*

Organization Indicators File: Valid local file name of existing file that contains terms that frequently precede organization names (e.g., "acquired") in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Organization Indicators File*

Organization Suffixes File: Valid local file name of existing file that contains a list of organizational abbreviations (e.g., "Corp." or "AG") in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Organization Suffixes File*

Organization Affixes File: Valid local file name of existing file that contains a list of terms that frequently follow organization suffixes as part of organization names (e.g., "Import and Export") in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Organization Affixes File*

Organizations File: Valid local file name of existing file that contains a list of complete, tokenized organization names (e.g., "Foo and Partners Ltd.") in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Organizations File*

Organizations as Meta Data: Valid name of meta-data attribute in DIAsDEM documents whose values store exactly one complete, tokenized organization name (e.g., "TokenizedNameOfPublishingCompany"); optional parameter; default value: project property *NEEX 2.1: Default Organizations Meta Data Attribute in DIAsDEM Documents*

Place Indicators File: Valid local file name of existing file that contains terms that frequently precede places (e.g., "in" or "to") in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Place Indicators File*

Places File: Valid local file name of existing file that contains a list of places (i.e., cities) in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Places File*

Place Affixes File: Valid local file name of existing file that contains a list of terms that frequently follow places (e.g., districts and names of rivers) as part of the place name in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Place Affixes File*

Person Name Indicators File: Valid local file name of existing file that contains terms that frequently precede person names (e.g., "Mr." or "with") in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Person Name Indicators File*

Titles File: Valid local file name of existing file that contains a list of academic and professional titles (e.g., "Prof." or "Prof. Dr.") in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Titles File*

Forenames File: Valid local file name of existing file that contains a list of forenames in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Forenames File*

Middle Initials File: Valid local file name of existing file that contains a list of middle initials (e.g., "von", "de la" or "A.") in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Middle Initials File*

Surnames File: Valid local file name of existing file that contains a list of surnames in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Surnames File*

Surname Suffixes File: Valid local file name of existing file that contains a list of frequent surname suffixes (e.g., "wicz" or "ova") in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Surname Suffixes File*

Name Affixes File: Valid local file name of existing file that contains a list of terms that frequently follow person names (e.g., "Ph.D." or "jr.") as part of the name in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Name Affixes File*

Professions File: Valid local file name of existing file that contains a list of professions (e.g., "CEO" or "President") that should be associated with person names in the format described in section 4.3.3 on page 94; file extension: `.txt`; optional parameter; default value: project property *NEEX 2.1: Default Professions File*

Composite NE File: Valid local file name of existing file that contains NEEX-specific rules for instantiating composite named entities of type "person", "companies", and "company_relocation" in the format described in section 4.3.3 on page 94; file extension: `.txt`; default value: project property *NEEX 2.1: Default Composite NE File*;

Debugging Files Directory: Valid local file name of existing directory for storing debugging files created by NEEX 2.1; optional parameter; default value: project property *NEEX 2.1: Default Directory of Debugging HTML Files*;

Advanced Options: The option *Determine Canonical Forms of Named Entities* cannot be enabled, because NEEX 2.1 does not support this operation.

3.2.8 Lemmatizing Text Units

Lemmatization of terms is the final text pre-processing step. During this step, grammatical roots of terms (i.e., their lemma forms) are determined and terms are replaced with their lemma forms. For example, inflected verb forms (e.g., "went") are mapped onto their respective infinite forms (e.g., "go"). This pre-processing step drastically reduces the number of distinct terms occurring in a collection. Hence, lemmatization also facilitates both the establishment and the usage of domain-specific thesauri that are required by DIAsDEM Workbench for controlled dimension reduction.

DIAsDEM Workbench supports two distinct methods of creating lemma forms. They can either be automatically determined by TreeTagger or each term can be looked up in a user-supplied list of known lemma forms. TreeTagger is a multilingual part-of-speech tagger developed by Helmut Schmid [Sch94]. As of August 2003, TreeTagger for Linux and Solaris can be used for research purposes free of charge. Using TreeTagger is the preferred method of lemmatization. However, the list-based method of determining lemma forms is applied in this case study to avoid any problems with installing the part-of-speech tagger. In contrast to 'real' part-of-speech tagging, this lexicon-based method has a main disadvantage: Lemma forms can only be determined for terms whose grammatical root forms are listed in the file of known lemma forms. Additionally, the syntactical context of term occurrences is not taken into consideration. Select *Actions → Prepare Data Set → Lemmatize Text Units* and type in the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
Lemmatization Algorithm	`Look Up Lemma Form in List`
TreeTagger Input File	
TreeTagger Output File	
Known Lemma Forms	`${PARAMETER_HOME}/lemmaForms/de/Case1LemmaForms.txt`
Unknown Lemma Forms	`${PARAMETER_HOME}/lemmaForms/de/NewLemmaForms.txt`
Advanced Options	`Disabled: Create New Known Lemma Forms File`

Click the *OK* button to start lemmatizing text units. Thereafter, check the contents of the file `${PROJECT_HOME}/inputCollection/volume100878.xml`: The section `<ProcessedTextUnits>` of this DIAsDEM document has been updated. After tokenization, its elements `<ProcessedTextUnit>` mark up lemma forms and named entity placeholders of identified text units. Furthermore, the section `<RollbackTextUnits>` has been updated to enable a rollback of this task.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0"> ...
    <TextUnitsLayer TextUnitsLayerID="0" TextUnitsDescription="Algorithm:
    HEURISTIC_SENTENCE_IDENTIFIER"> ...
```

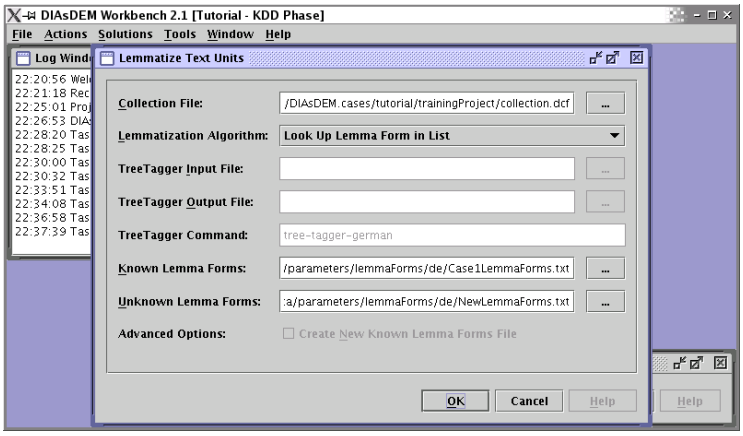

Figure 3.10: *Lemmatize Text Units* dialog

```
    <ProcessedTextUnits>
      <ProcessedTextUnit TextUnitID="0">d Handel mit Ware alle Art sowie
      Import und Export .</ProcessedTextUnit> ... <ProcessedTextUnit
      TextUnitID="2">Stammkapital : <NeRef NeID="0" /> .</ProcessedTextUnit>
      <ProcessedTextUnit TextUnitID="3">Gesellschaft_mit_beschränkter_Haftung
      .</ProcessedTextUnit> ... <ProcessedTextUnit TextUnitID="9"><NeRef
      NeID="16" />, sein zur Geschäftsführerin bestellen .</ProcessedTextUnit> ...
      <ProcessedTextUnit TextUnitID="11">nicht eintragen : d Bekanntmachung d
      Gesellschaft erfolgen im Bundesanzeiger .</ProcessedTextUnit>
    </ProcessedTextUnits> ...
  </TextUnitsLayer>
 </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

Note, the list of known lemma forms had been created using TreeTagger. In the file shown above, the inflected verb form “ist” now occurring in `<RollbackTextUnit>` has been mapped onto its infinitive form “sein” in the section `<ProcessedTextUnit>` . If TreeTagger is unable to determine the grammatical root form for a term, its lemma form simply equals the original term.

During the iterative clustering phase, text unit vectors are clustered based on similarity of their contents in order to discover semantic XML tags. Text unit vectors are created by mapping elements of the section `<ProcessedTextUnits>` onto vectors. Thereby, vector dimensions correspond to so-called text units descriptors that are defined in a domain-specific and case-sensitive thesaurus. Consequently, thesauri should contain case-sensitive lemma forms of descriptor and non-descriptor thesaurus terms, because they truly occur in the section `<ProcessedTextUnits>` only.

Lemmatize Text Units: Summary

Module: *Actions → Prepare Data Set → Lemmatize Text Units*

Use Case: The user must pre-process all imported texts as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives. Creating lemma forms is pre-processing phase 4 of 4.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. Text units should have been created and tokenized in the DIAsDEM collection and named entities should have been replaced with placeholders in all text units. If *Use TreeTagger to Determine Lemma Form* is enabled, the absolute file name of the respective TreeTagger start script (e.g., `/.../tree-tagger-german`) must be entered in the *External Programs* tab of the *Tools → Options* dialog.

Result: Elements of section `<ProcessedTextUnits>` mark up text units containing lemma forms and named entity placeholders. Additionally, the project properties *Default Collection File*, *Default TreeTagger Input File*, *Default TreeTagger Output File*, *Default Known Lemma Forms File* and *Default Unknown Lemma Forms File* are set and updated, respectively.

Lemmatize Text Units: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Lemmatization Algorithm: If *Use TreeTagger to Determine Lemma Form* is enabled, the external part-of-speech tagger is employed. In this case, TreeTagger must have been successfully installed and the absolute file name of the respective TreeTagger start script (e.g., `/.../tree-tagger-german`) must be entered in the *External Programs* tab of the *Tools → Options* dialog. If *Look Up Lemma Form in List* is enabled, a list of a priory known grammatical root forms (i.e., *Known Lemma Forms*) is utilized.

TreeTagger Input File: Must be set, if *Use TreeTagger to Determine Lemma Form* is enabled; valid local file name of new or existing file that is replaced; this temporary file is created by DIAsDEM Workbench and includes text to be POS-tagged by TreeTagger; file extension: `.txt`; default value: project property *Default TreeTagger Input File*

TreeTagger Output File: Must be set, if *Use TreeTagger to Determine Lemma Form* is enabled; valid local file name of new or existing file that is replaced;

this temporary file is created by TreeTagger and includes the results of POS-tagging; file extension: `.txt`; default value: project property *Default TreeTagger Output File*

Known Lemma Forms: Must be set, if *Look Up Lemma Form in List* is enabled; valid local file name of existing file that contains terms along with their lemma forms in the format described in section 4.3.6 on page 107; file extension: `.txt`; default value: project property *Default Known Lemma Forms File*

Unknown Lemma Forms: Must be set, if *Look Up Lemma Form in List* is enabled; valid local file name of existing file that is created or extended by DIAsDEM Workbench; includes terms occurring in the collection that are not listed in *Known Lemma Forms* as well as the context of their occurrence (i.e., the sentence); can be used to update *Known Lemma Forms*; format described in section 4.3.6 on page 107; file extension: `.txt`; default value: project property *Default Unknown Lemma Forms File*

Advanced Options: If *Create New Lemma Forms File* is enabled along with *Use TreeTagger to Determine Lemma Form*, all terms and the corresponding lemma forms determined by TreeTagger are saved for later usage as a file of *Known Lemma Forms*.

3.3 Iterative Clustering in the KDD Phase

3.3.1 Computing Term Frequency Statistics

During the clustering phase, DIAsDEM Workbench requires a controlled vocabulary in the form of a domain-specific thesaurus. Text units are mapped onto vectors whose dimensions correspond to thesaurus descriptors. Computing term frequency (TF) statistics for a collection is the first step in establishing or updating a thesaurus for subsequent use in clustering. TF statistics give an insight into the specific word frequency distribution prevalent in a certain document collection. Based on term frequency statistics, an initial thesaurus can either be created or an existing thesaurus can be updated by adding, editing or removing terms of interest. Although there exists a prepared thesaurus for this case study, creating and inspecting TF statistics is described in this tutorial for the sake of completeness. Therefore, select *Actions* → *Understand Domain* → *Compute Term Frequency Statistics* and provide the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
TF Statistics File	`${PROJECT_HOME}/termFrequencies.dtf`
Advanced Options	`Enabled: Export Original Texts in CSV Format`
	`Enabled: Export Term Frequency Statistics in CSV Format`
	`Enabled: Export Term Frequency Statistics in HTML Format`
	`Enabled: Exclude Numbers, Dates and NE Placeholders`

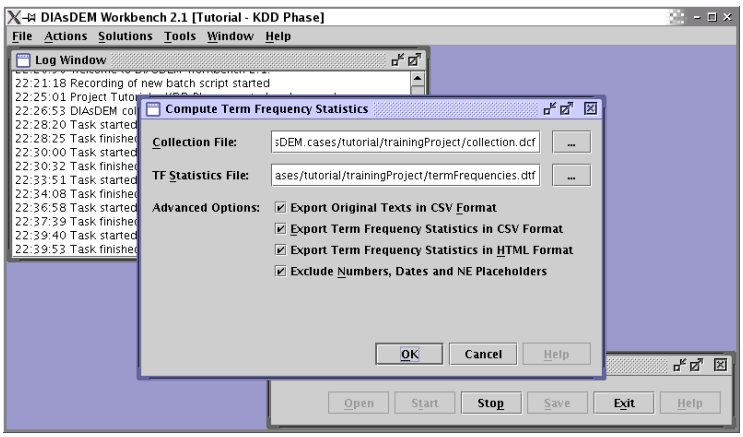

Figure 3.11: *Compute Term Frequency Statistics* dialog

Click on *OK* to compute term frequency statistics. Thereafter, check the contents of the directory `${PROJECT_HOME}`, which now contains the DIAsDEM-specific TF statistics file `termFrequencies.dtf`. It can be opened using *Tools* → *Term Frequency Statistics Viewer*. Moreover, this directory contains TF statistics in CSV (`termFrequencies.csv`) and HTML (`termFrequencies.html`) format, respectively. Due to the settings of advanced options, all original texts of this collection have also been exported into the CSV files `termFrequencies.orig.csv` and `termFrequencies.proc.csv`. The former contains file names of input documents and the original texts as stored in section `<OriginalText>`. In contrast, the latter file contains processed texts (i.e., the contents of the section `<ProcessedTextUnits>`) along with their input file names. Both export files can be input to further text analysis and text mining activities employing third-party software. According to the parameter settings, numbers, date literals, named entity placeholders, and other tokens that do not contain at least one letter are excluded from the TF statistics.

Compute Term Frequency Statistics: Summary

Module: *Actions* → *Understand Domain* → *Compute Term Frequency Statistics*

Use Case: The user wants to analyze the term frequency distribution prevalent in a document collection to get insight into the particularities of its specific vocabulary. Additionally, the user might want to create an initial, collection-specific thesaurus or may want to edit an existing thesaurus based on term frequencies.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. Text units should have been created and tokenized in the DIAsDEM collection and named entities should have been replaced with placeholders in all text units.

Result: *TF Statistics File* contains the absolute frequencies of single- and multi-token terms that occur in the section `<ProcessedTextUnits>` of DIAsDEM documents in collection *Collection File*. Additionally, the project properties *Default Collection File* and *Default Word Statistics File* are set and updated, respectively.

Remarks: The section `<ProcessedTextUnits>` should contain lemmatized text units in case of computing TF statistics for thesaurus establishment or update, because only lemmatized text units should be mapped onto vectors for subsequent clustering. Thesauri to be employed by DIAsDEM Workbench should include lemma forms of both descriptors and non-descriptors only.

Compute Term Frequency Statistics: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

TF Statistics File: Valid local file name of new or existing file that is created or replaced by DIAsDEM Workbench; file extension: `.dtf`; default value: project property *Default Word Statistics File*

Advanced Options: If *Export Original Texts in CSV Format* is enabled, two CSV files are created in `${PROJECT_HOME}` that contain the file name and the textual contents of each document. If *Export Term Frequency Statistics in CSV Format* is enabled, a CSV file is created in `${PROJECT_HOME}` that contains all terms and the respective absolute frequencies. If *Export Term Frequency Statistics in HTML Format* is enabled, an HTML file is created in `${PROJECT_HOME}` that lists terms and their absolute frequencies. Tokens that do not contain at least one letter are excluded from TF statistics, if *Exclude Numbers, Dates and NE Placeholders* is enabled.

3.3.2 Viewing Term Frequency Statistics

Select *Tools* → *Term Frequency Statistics Viewer* to open the new TF statistics file. Thus, click on *Open Statistics* and choose the file `${PROJECT_HOME}/termFrequencies.dtf`. After entering the minimum frequency of terms to be displayed (e.g., 5), its contents are shown in the left pane as illustrated in Figure 3.12. TF statistics can either be sorted by decreasing frequency or by ascending term. To sort the list of terms in the left pane, click the buttons *Sort by Freq.* and *Sort by Term*, respectively.

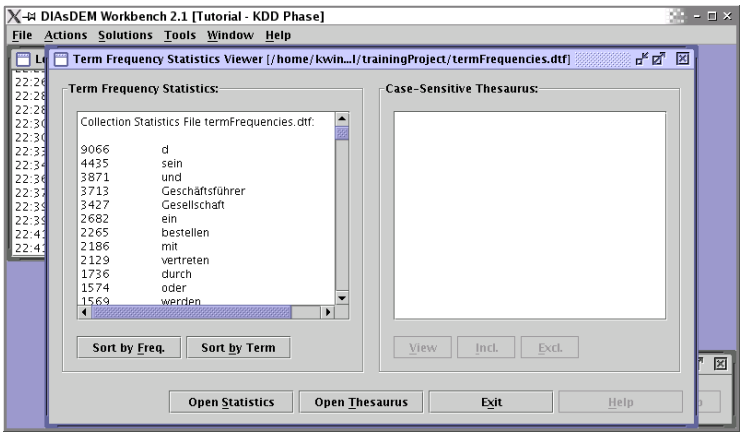

Figure 3.12: *Term Frequency Statistics Viewer* of DIAsDEM Workbench 2.1

Terms appearing in the left pane can be compared with an existing DIAsDEM-specific thesaurus file. To proceed, click the *Open Thesaurus* button and choose the thesaurus file `${PARAMETER_HOME}/thesauri/de/Case123Thesaurus.dth`. As illustrated in Figure 3.13, the entire thesaurus is initially displayed in the right pane. Each line corresponds to one thesaurus term that can either be a descriptor or a non-descriptor referencing an associated descriptor term.

Thesaurus entry `Ablehnung (D; Case2)` corresponds to the descriptor ("D") term "Ablehnung" that is a valid text unit descriptor in case study 2 only. Note again, valid text unit descriptors correspond to dimensions of text unit vectors to be clustered subsequently. According to thesaurus entry `<<person>> (D; Case1)`, named entity type "person" is a valid descriptor in case study 1. For each identified instance of named entity type "person" in a text unit, the respective descriptor counter is incremented. Finally, thesaurus entry `beginnen (N; Beginn)` states that the non-descriptor ("N") term "beginnen" is mapped onto its descriptor term "Beginn". If the term "beginnen" occurs in a text unit, the counter of its descriptor term "Beginn" is thus incremented. However, the descriptor `Beginn (D; Case1)` is a valid descriptor in case study 1 only.

Click the *Incl.* button in the right pane to filter TF statistics terms that are also

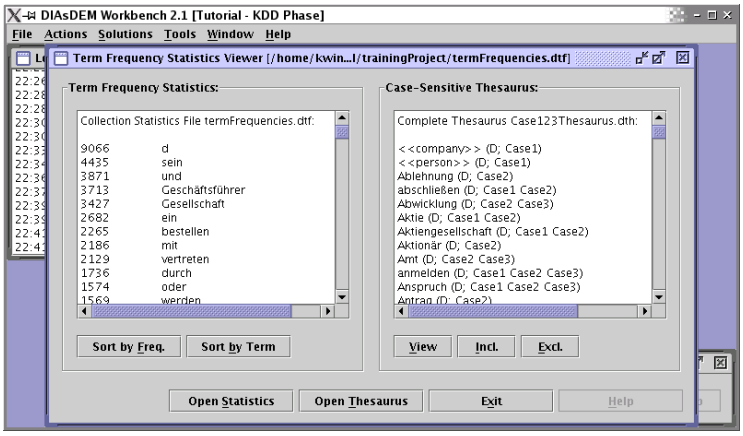

Figure 3.13: *Term Frequency Statistics Viewer* of DIAsDEM Workbench 2.1

descriptor or non-descriptor thesaurus terms. In this case, the collection-specific term frequency is also shown for each thesaurus term in the right pane. Click the *Excl.* button to filter TF statistics terms that are not contained in the thesaurus. Frequently occurring and semantically important terms, which are not listed in the thesaurus, are candidates for thesaurus updates. In contrast, infrequently occurring terms might be removed from the thesaurus to reduce the dimensionality of text units vectors. However, concepts such as "Tätigkeit" should not be deleted, if they are descriptor terms for less frequently occurring non-descriptors. Note, specific named entity placeholders (e.g., `<<0>>`) should not be included in any thesaurus, because they might replace instances of various named entity types such as "person" or "date".

After analyzing TF statistics, you might consider to add the frequently occurring term "Bundesanzeiger" as a descriptor to the thesaurus. Moreover, the rather frequent term "Bauvorhaben" should be a non-descriptor term pointing to the important descriptor "Tätigkeit". Additionally, thesaurus term "Aktionär" should be removed due to its infrequent occurrence in the collection. Thesaurus updates are explained in the next section. Therefore, do not close *Term Frequency Statistics Viewer* yet.

3.3.3 Editing Domain-Specific Thesauri

DIAsDEM Workbench includes two German thesauri in `${PARAMETER_HOME}/thesauri/de`. They contain application-specific vocabularies for case studies related to Commercial Register entries and corporate news, respectively. Thesauri, which should be employed in different application domains, can be created by *Actions* → *Understand Domain* → *Establish Initial Thesaurus* as described in section 3.6.3. However, the remainder of this section focuses on the process of updating an existing, DIAsDEM-specific

thesaurus by adding, editing and removing terms. Select *Tools* → *Thesaurus Editor*, click on *Open* and choose the thesaurus file `Case123Thesaurus.dth` in the directory `${PARAMETER_HOME}/thesauri/de`.

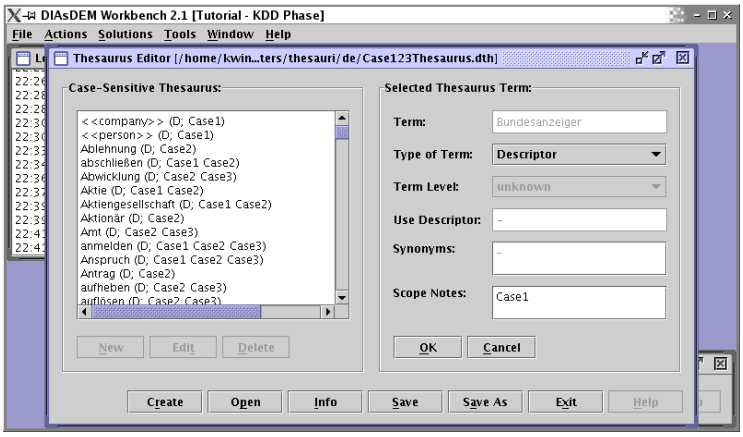

Figure 3.14: *Thesaurus Editor* of DIAsDEM Workbench 2.1

To add the first new term, click on *New* and enter "Bundesanzeiger" in the appearing dialog. This noun is thereafter displayed in the right editor pane as the selected and editable thesaurus term. Similarly to many Windows applications, you might use the system clipboard to transfer textual contents for example from *Term Frequency Statistics Viewer* to *Thesaurus Editor*. Using the mouse, select the term "Bundesanzeiger" in *Term Frequency Statistics Viewer*. The highlighted text can be copied into the clipboard by the keyboard shortcut *CTRL-C*. Afterwards, clipboard contents can be pasted into other documents by placing the cursor at the desired position and using the keyboard shortcut *CTRL-V*. Moreover, the keyboard shortcut *CTRL-X* can be used to cut (i.e., to remove) selected text from the source document after copying it to the clipboard.

Each thesaurus term must either be a descriptor or a non-descriptor that references an associated descriptor term. Due to its frequent occurrence and semantic importance, the German word "Bundesanzeiger" should be a text unit vector dimension and must thus be a descriptor. Therefore, change the attribute value *Type of Term* from *unknown* to *Descriptor*. The contents of the field *Scope Notes* can be used to limit the number of valid descriptors during the process of vectorizing text units. The term "Bundesanzeiger" should be a valid descriptor in the first case study only. Thus, case-sensitively enter "Case1" in the field *Scope Notes*. Finally, the new term is added to the current thesaurus by clicking on *OK*. Otherwise, click the *Cancel* button to discard any modifications of the selected term. Figure 3.14 depicts *Thesaurus Editor* before committing the insertion of the new term "Bundesanzeiger".

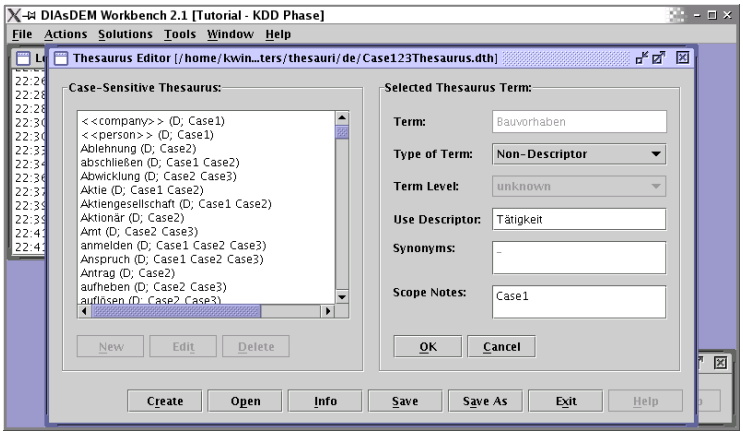

Figure 3.15: *Thesaurus Editor* of DIAsDEM Workbench 2.1

Insert the second new term "Bauvorhaben" into the thesaurus and change its *Type of Term* from *unknown* to *Non-Descriptor*. For each non-descriptor, an associated descriptor must be specified in the field *Use Descriptor*. Hence, type the descriptor term "Tätigkeit" in this field. Note, the *Use Descriptor* field must not contain other non-descriptors or terms that are not included in the same thesaurus. As before, input "Case1" in the field *Scope Notes* as well. Figure 3.15 illustrates *Thesaurus Editor* before committing the insertion of the term "Bauvorhaben". The fields *Term Level* and *Synonyms* are not used in the current version of DIAsDEM Workbench.

Existing terms such as "Bundesanzeiger" and "Bauvorhaben" can be modified by clicking the *Edit* button and afterwards entering the term of interest. Alternatively, a term listed in the left editor pane can be selected using the mouse. Thereafter, clicking on *Edit* automatically opens the corresponding term in the right editor pane for update. Analogously, terms can be removed from the thesaurus by clicking the *Delete* button. However, make sure not to delete descriptor terms that are referenced by remaining non-descriptors. Finally, remove the German term "Aktionär", because it occurs only once in the entire collection of Commercial Register entries.

Click the *Info* button and look at the brief thesaurus summary that lists the number of terms, descriptors and non-descriptors in the opened thesaurus. The case-specific thesaurus should now include 127 descriptor and 104 non-descriptor terms. Keep in mind that the number of descriptors should be kept as low as possible, because DIAsDEM Workbench does not employ uncontrolled techniques for dimensionality reduction (e.g., singular value decomposition). As a rule of thumb, the number of descriptors should not exceed 250 terms. Click the *Save* button to commit the previous thesaurus updates. After saving, inspect the contents of the directory `${PARAMETER_HOME}/thesauri/de`.

In addition to updating the DIAsDEM-specific thesaurus file, saving a thesaurus always results in the creation of thesaurus files in CSV and HTML format in the same directory. The latter contains information about all thesaurus terms and an explicit mapping of descriptors onto their associated non-descriptors. Finally, click the respective *Exit* buttons to close both *Thesaurus Editor* and *Term Frequency Statistics Viewer*.

3.3.4 Vectorizing Text Units in Iteration 1

Concerning the clustering of text unit vectors, DIAsDEM Workbench implements both a plug-in and a plug-out concept, which enables the usage of various clustering algorithms. Users can either employ one of three built-in Weka [WF99] clustering algorithms (i.e., k-means, Cobweb and EM) or utilize algorithms supplied by external data mining applications. To ensure this flexibility, DIAsDEM Workbench is capable of exporting text unit vector files in three different formats. As the k-means clustering algorithm provided by the Java-based data mining library Weka is employed is this case study, vectors are exported in the Weka-specific ARFF format only. However, all three formats are briefly described in section 4.4.1 on page 109. To export text unit vectors for the first clustering iteration, select *Actions* → *Prepare Data Set* → *Vectorize Text Units* and enter the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
KDD Process Iteration	`1`
Text Unit Vectors Format	`ARFF: Weka Data Mining Project`
Text Unit Vectors File	`${PROJECT_HOME}/1vectors.arff`
Thesaurus File	`${PARAMETER_HOME}/thesauri/de/Case123Thesaurus.dth`
Text Unit Descriptors	`Descriptors whose Scope Notes Contain String` `Case1`
Descriptor Frequency	`Boolean Descriptor Frequency`
Descriptor Weights	`Create Descriptor Weights File: IDF Weights` `${PROJECT_HOME}/1weights.ddw`
Advanced Options	`Disabled: Create File for Mining Descriptor Association Rules` `Enabled: Create Meta-Data File for Text Unit Vectors File`

Click the *OK* button to export text unit vectors according to these parameter settings. In the first clustering iteration, processed text units in section `<ProcessedTextUnits>` are mapped onto their vector representations. Let D be the set of descriptors. The dimensionality of text unit vectors corresponds to $|D| = 73$ descriptors in thesaurus file `Case123Thesaurus.dth` that contain the string "Case1" in their respective scope notes.

Mapping text units onto text unit vectors works as follows: Firstly, a boolean vector is created for each text unit. Each vector component $i = 1, \ldots, |D|$ represents the boolean term frequency of descriptor d_i in the text unit. Vector component i is 1, if descriptor d_i occurs in the corresponding text unit, or 0 otherwise. Secondly, boolean vectors are weighted by multiplying each vector component i and the inverse document frequency

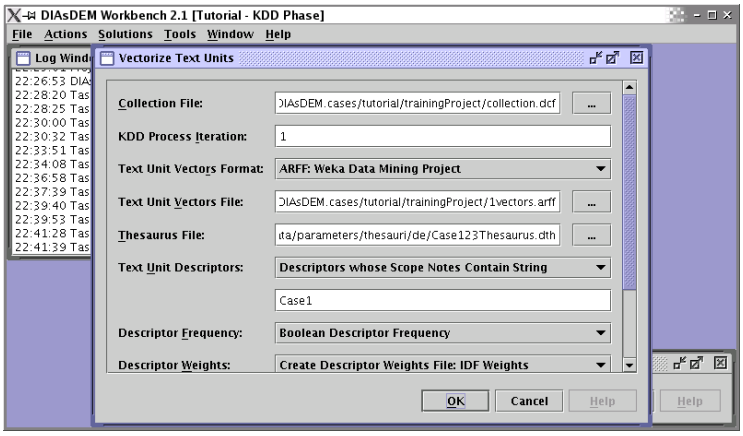

Figure 3.16: *Vectorize Text Units* dialog

of descriptor d_i. Let U be the set of text units in the collection and let $freq(d_i)$ be the absolute frequency of descriptor d_i in the same collection. The inverse document frequency of descriptor d_i is here defined as $log(|U|/freq(d_i))$. This weighting schema favors terms that occur in relatively few text units, because these terms have a higher discriminative power than terms occurring in almost all text units. To sum up, vector component i represents the product of boolean term frequency of descriptor d_i in the corresponding sentence and inverse document frequency of descriptor d_i within the entire collection. Open the meta-data file `${PROJECT_HOME}/1vectors.arff.meta` that lists the descriptor frequency and the inverse document frequency for each descriptor:

```
...
D1_Aktie = Aktie; Descriptor Frequency = 37; Descriptor Weight = 5.5
D2_Gesellschafter = Gesellschafter; Descriptor Frequency = 172; Descriptor Weight = 3.9
...
D73_Anspruch = Anspruch; Descriptor Frequency = 9; Descriptor Weight = 6.9
```

Note that descriptor term “Aktie” and the associated non-descriptors such as “Namensaktie” occur 37 times in the collection of Commercial Register entries. The term weight of “Aktie”, which equals its inverse document frequency, is greater than the term weight of “Gesellschafter”, because “Aktie” occurs less frequently in this collection. According to the applied IDF weighting schema, “Aktie” has a greater discriminative power than “Gesellschafter” due to its relatively infrequent occurrence in the collection. Note, this meta-data file has a purely informative character. In contrast, the file `${PROJECT_HOME}/1weights.ddw` contains the same descriptor weights for usage in the first clustering iteration of the application phase.

The text unit vector file ${PROJECT_HOME}/vectors1.arff is input to the first clustering iteration, which is described in the next section. This file contains 9,254 vectors to be clustered in the Weka-specific ARFF format [WF99]. ARFF-files include metadata about the relation and its attributes (i.e., their names and domains) as well as the actual data below @data. For example, the second vector depicted below corresponds to the second text unit of file ${PROJECT_HOME}/inputCollection/volume100668.xml: "Persönlich haftende Gesellschafterin: AGE Glas Vertrieb GmbH, Sitz: Garbsen." The descriptor "Gesellschafter" occurs in this text unit in the form of its associated non-descriptor "Gesellschafterin". Hence the second vector component represents a term weight greater than zero. The first component of the second vector equals zero, because neither the descriptor term "Aktie" nor a related non-descriptor occurs in this sentence.

```
@relation 'DIAsDEM'
@attribute DocumentType string
@attribute Document string
@attribute TextUnit string
@attribute D1_Aktie real
@attribute D2_Gesellschafter real
...
@attribute D73_Anspruch real
@data
...
null,/home/.../volume100668.xml:0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
null,/home/.../volume100668.xml:0,1,0,3.98531,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
...
```

Vectorize Text Units: Summary

Module: *Actions → Prepare Data Set → Vectorize Text Units*

Use Case: The user wants to cluster pre-processed text units of imported texts as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives. Vectorizing text units precedes the clustering step.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section <ProcessedTextUnits>. Text units should have been created, tokenized, and lemmatized in the DIAsDEM collection and named entities should have been replaced with placeholders in all text units.

Remarks: In the KDD phase of the DIAsDEM framework, an iteration-specific descriptor weights file must be created for usage in the application phase.

Vectorize Text Units: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

KDD Process Iteration: If `1` is input to indicate the first iteration, text unit vectors are created for all text units in section `<ProcessedTextUnits>`. If a number greater than `1` is input to indicate subsequent iterations, text unit vectors are created for text units in section `<ProcessedTextUnits>` that have not been semantically named in a previous clustering iteration (i.e., `ClusterLabel="-"`). These vectors have been assigned to qualitatively unacceptable clusters in all preceding iterations as explained in section 1. Default value: project property *Default Iteration (1, 2, ...)*

Text Unit Vectors Format: Choice of vector file format as described in section 4.4.1 on page 109 between comma separated values (CSV file), fixed width values (TXT file) and the Weka-specific ARFF file format; default value: project property *Default Vector File Format*

Text Unit Vectors File: Valid local file name of file to be created or replaced by DIAsDEM Workbench; file extension depends on choice of *Text Unit Vectors Format*: `.csv`, `.txt` or `.arff`; default value: project property *Default Text Unit Vectors File*

Thesaurus File: Valid local file name of existing DIAsDEM-specific thesaurus file as described in section 4.4.1 on page 109; file extension: `.dth`; default value: project property *Default Thesaurus File*

Text Unit Descriptors: If *All Descriptors in Thesaurus* is enabled, all descriptor terms in *Thesaurus File* are vector dimensions. If *Descriptors whose Scope Note Contain String* is enabled, only descriptor terms in *Thesaurus File* are valid whose scope notes contain the complete string entered below. If *Descriptors whose Scope Note Don't Contain String* is enabled, only descriptor terms in *Thesaurus File* are valid whose scope notes do not contain the string entered below.

Descriptor Frequency: If *Raw Descriptor Frequency* is enabled, term frequency of valid descriptor d in text unit u equals the number of times d occurs in u. If *Boolean Descriptor Frequency* is enabled, term frequency of d in u is 1 if d occurs in u and 0 otherwise.

Descriptor Weights: In the KDD phase, either the option *Create Descriptor Weights File: Equal Weights* or *Create Descriptor Weights File: IDF Weights* has to be enabled. In the former case, the term frequency of valid descriptor

d in text unit u is not weighted at all and all descriptor weights thus equal 1. If the latter option is enabled, the term frequency of valid descriptor d in text unit u is multiplied by the inverse document frequency of d in the entire collection. The specified weights file (file extension: `.ddw`; default value: project property *Default Descriptor Weights File*) is created or replaced in the KDD phase. In contrast, the option *Apply Existing Weights File* must be enabled in the application phase. In this phase, an existing, iteration-specific descriptor weights file must be specified.

Advanced Options: If *Create File for Mining Descriptor Association Rules* is enabled, an additional file named analogously to *Text Unit Vectors File* but suffixed `.assoc` is created. It can be used for discovering association rules between descriptor terms in text units. If *Create Meta-Data File for Text Unit Vectors File* is enabled, an additional file is created that is named analogously to *Text Unit Vectors File*, but suffixed `.meta`. This meta-data file contains mappings of abbreviated attribute names onto their respective unabbreviated descriptors along with their descriptor frequencies and descriptor weights.

3.3.5 Clustering Text Unit Vectors in Iteration 1

DIAsDEM Workbench supports the export of text unit vectors into different, mostly standardized file formats. Hence, various external clustering algorithms could be employed to group vectors based on their contents for subsequent semantic labeling. This plug-out concept has been successfully tested in case studies employing commercial data mining applications such as IBM Intelligent Miner for Data and SAS Enterprise Miner, respectively [WS01c, WS02c]. Additionally, DIAsDEM Workbench implements a plug-in concept that wraps the Java-based data mining library Weka.

Along with various data pre-processing and machine learning algorithms, Weka includes three clustering algorithms (i.e., k-means, Cobweb, and EM), which have been integrated into DIAsDEM Workbench. Discussing these clustering algorithms is beyond the scope of this tutorial. See [WF99] for an excellent description of these algorithms, their parameters and their implementation in the open source Weka library. For moderate amounts of data, all three Weka algorithms should be capable of clustering text unit vectors without memory- or runtime-related problems. In this case study, the k-means clustering algorithm is employed only. To proceed, select *Actions* → *Discover Patterns* → *Cluster Text Unit Vectors (Weka)* and enter the following parameters:

Parameter	Value
Clustering Mode	`Clustering Phase (Create New Clustering Model)`
Collection File	`${PROJECT_HOME}/collection.dcf`

Text Unit Vectors File	`${PROJECT_HOME}/1vectors.arff`
Clustering Algorithm	`weka.clusterers.SimpleKMeans`

Clustering Parameters	`1) Number of Clusters = 100`
	`2) Acuity =`
	`3) Cutoff =`
	`4) Max. Iterations =`
	`5) Random Number Seed =`
	`6) Min. Std. Deviation =`

Clustering Results File	`${PARAMETER_HOME}/1results.csv`
Text Unit Clusterer File	`${PARAMETER_HOME}/1clusterer.wskm`

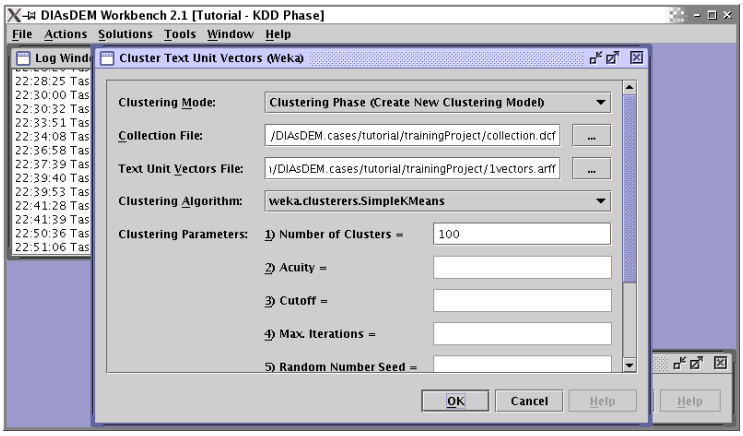

Figure 3.17: *Cluster Text Unit Vectors (Weka)* dialog

Click the *OK* button to start the first clustering iteration. According to the parameters, the simple k-means algorithm is executed to create exactly $k = 100$ text unit vector clusters, some of whom may of course remain empty. *Number of Clusters* is the only parameter of this algorithm, whereas *Acuity* and *Cutoff* are two parameters of the Cobweb clustering algorithm. The EM algorithm can be parameterized by *Max. Iterations*, *Random Number Seed*, and *Min. Std. Deviation*. All algorithms require text unit vector files that conform to the Weka-specific ARFF-format. Again, see [WF99] for a detailed discussion of these parameters. The progress of clustering cannot be displayed due to the missing support of progress measurement in Weka. Running Java 1.4.2 for Linux on a Notebook equipped with an 1.06 GHz Intel Mobile Celeron processor and 256 MB memory, clustering of 9,254 text unit vectors requires approx. eight minutes.

DIAsDEM Workbench post-processes the proprietary output file generated by Weka clusterers (e.g., `${PROJECT_HOME}/1results.csv.temp`). Clustering results are converted into CSV files, which can easily be processed by the tasks *Actions* → *Postprocess Patterns* → *Monitor Cluster Quality* and *Actions* → *Postprocess Patterns* →

Tag Text Units, respectively. After clustering has finished, inspect the *Clustering Results File* ${PROJECT_HOME}/1results.csv. Each line contains a DIAsDEM document ID, the respective text unit ID as the second attribute and the associated cluster ID as the third attribute. Consider for example the first and only DIAsDEM document in file ${PROJECT_HOME}/inputCollection/volume100668.xml: The first text unit is assigned to cluster 25, whereas the second one is assigned to cluster 39. Note, cluster 25 also contains the first text unit of the DIAsDEM document in file volume100669.xml.

```
...
/home/.../volume100668.xml:0,0,25
/home/.../volume100668.xml:0,1,39
/home/.../volume100668.xml:0,2,41
/home/.../volume100668.xml:0,3,57
/home/.../volume100668.xml:0,4,33
/home/.../volume100669.xml:0,0,25
...
```

As described in section 3.3.6, the contents of text units clusters can be visualized by *Actions → Postprocess Patterns → Monitor Cluster Quality*. The file ${PROJECT_HOME}/1clusterer.wskm is a serialized instance of Java class weka.clusterers.SimpleKMeans. This so-called text unit clusterer can be employed to cluster text unit vectors during the application phase of the DIAsDEM framework. In contrast to the clustering or KDD phase exemplified by this section, *Text Unit Clusterer File* is an input file during the application phase. However, running DIAsDEM Workbench in application mode can be simulated by applying ${PROJECT_HOME}/1clusterer.wskm to the same text unit vectors in ${PROJECT_HOME}/1vectors.arff using the following parameters:

Parameter	Value
Clustering Mode	Application Phase (Apply Existing Clustering Model)
Collection File	${PROJECT_HOME}/collection.dcf
Text Unit Vectors File	${PROJECT_HOME}/1vectors.arff
Clustering Algorithm	weka.clusterers.SimpleKMeans
Clustering Parameters	1) Number of Clusters =
	2) Acuity =
	3) Cutoff =
	4) Max. Iterations =
	5) Random Number Seed =
	6) Min. Std. Deviation =
Clustering Results File	${PARAMETER_HOME}/1results.csv
Text Unit Clusterer File	${PARAMETER_HOME}/1clusterer.wskm

Compared to training a text unit clusterer, a significant runtime improvement can be noticed in application mode. When applying an existing clusterer to text unit vectors, the parameters of the algorithm cannot be altered due to obvious reasons. After clustering text unit vectors, monitoring the cluster quality is the next step in this clustering or

KDD phase of the case study as described in the next section. In contrast, clustering is directly followed by text unit tagging, if DIAsDEM Workbench is running in application mode.

Cluster Text Unit Vectors (Weka): Summary

Module: *Actions → Discover Patterns → Cluster Text Unit Vectors (Weka)*

Use Case: The user wants to cluster text unit vectors as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives.

Prerequisites: Vectors to be clustered in *Text Unit Vectors File* must conform to the Weka-specific ARFF file format. This file format is exported by *Actions → Prepare Data Set → Vectorize Text Units* and is described in section 4.4.1 on page 109.

Result: In clustering mode, text unit vectors are clustered by the chosen algorithm. The resulting text unit clusterer is saved for subsequent usage in application mode. Given an existing text unit clusterer, text units can quickly be assigned to their respective clusters in application mode. Additionally, the project properties *Default Collection File*, *Default Text Unit Vectors File*, *Default Clustering Algorithm*, *Default Clustering Mode*, *Default Clustering Parameters*, *Default Clustering Results File* and *Default Text Unit Clusterer File* are set and updated, respectively.

Remarks: Instead of employing the Weka-based, internal clustering algorithms, any other algorithm might also be used, if the results can be exported or converted into a file format supported by DIAsDEM Workbench.

Cluster Text Unit Vectors (Weka): Parameters

Clustering Mode: If *Clustering Phase (Create New Clustering Model)* is enabled, a new text unit clusterer is trained according to the parameter settings and output as *Text Unit Clusterer File*. If *Application Phase (Apply Existing Clustering Model)* is enabled, the existing clusterer *Text Unit Clusterer File* is applied to the contents of *Text Unit Vectors File*.

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Text Unit Vectors File: Valid local file name of existing file; file extension: `.arff`; default value: project property *Default Text Unit Vectors File*

Clustering Algorithm: One of three algorithms supported by the Java-based Weka library [WF99] must be selected: *weka.clusterers.SimpleKMeans*, *weka.clusterers.Cobweb* or *weka.clusterers.EM*.

Clustering Parameters: If *Clustering Phase (Create New Clustering Model)* is enabled, the selected algorithm can be parameterized as follows [WF99]: *weka.clusterers.SimpleKMeans*: *Number of Clusters*; *weka.clusterers.Cobweb*: *Acuity* and *Cutoff*; *weka.clusterers.EM*: *Max. Iterations*, *Random Number Seed*, and *Min. Std. Deviation*.

Clustering Results File: Valid local file name of file to be created or replaced by DIAsDEM Workbench; file extension: `.csv` default value: project property *Default Clustering Results File*

Text Unit Clusterer File: Valid local file name of file to be created or replaced by DIAsDEM Workbench, if *Clustering Mode* is set to *Clustering Phase*; valid local file name of existing file, if *Clustering Mode* is set to *Application Phase*; file extension depends on clustering algorithm: `.wskm`, `.wcw`, or `.wem`; default value: project property *Default Text Unit Clusterer File*

3.3.6 Monitoring Cluster Quality in Iteration 1

As explained in the introduction, the set of text unit clusters discovered during clustering has to be analyzed to separate qualitatively "acceptable" clusters from "unacceptable" ones. Recall that members of the former are semi-automatically assigned a semantic label, whereas all text unit vectors assigned to qualitatively "unacceptable" clusters are re-clustered in the next iteration. A discussion of cluster quality criteria is beyond the scope of this tutorial. However, the cluster quality criteria are described in detail in [GSW01]. The Cluster Quality Monitor of DIAsDEM Workbench computes descriptive statistics for clusters, visualizes cluster contents in HTML files and creates a cluster label file. The latter contains default semantic labels for qualitatively "acceptable" clusters only. These default cluster labels are composed of text unit descriptors that prevail in the respective clusters. Start monitoring cluster quality by selecting *Actions* → *Postprocess Patterns* → *Monitor Cluster Quality* and submitting the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
KDD Process Iteration	`1`
Result File Format	`CSV: Comma Separated Values`
Cluster Result File	`${PROJECT_HOME}/1results.csv`
Cluster Directory	`${PROJECT_HOME}/kddProcessIteration1`
Cluster Label File	`${PROJECT_HOME}/1labels.dcl`

Max. Cluster ID	99
Thesaurus File	${PARAMETER_HOME}/thesauri/de/Case123Thesaurus.dth
Text Unit Descriptors	Descriptors whose Scope Notes Contain String Case1
Cluster Quality Criteria	1) Min. Cardinality = 50 2) Max. Distinct Ratio = 0.75 3) Min. Frequent Ratio = 0.25
Advanced Options	Disabled: Ignore First Line of Cluster Result File Enabled: Ignore Empty Clusters in Cluster Index HTML File Enabled: Launch Web Browser with Cluster Index HTML File Enabled: Launch Cluster Label Editor with Cluster Label File Enabled: Dump DIAsDEM Documents for Visualization

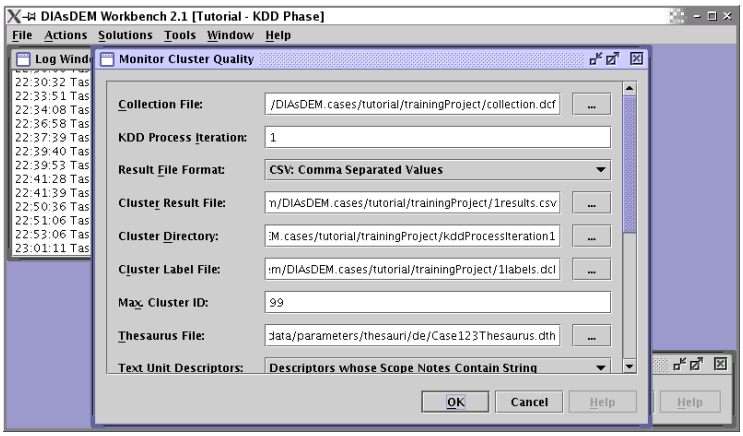

Figure 3.18: *Monitor Cluster Quality* dialog

For obvious reasons, the settings of *Thesaurus File* and *Text Unit Descriptors* must exactly correspond to the parameters entered in *Actions* → *Prepare Data Set* → *Vectorize Text Units* in the current clustering iteration. *Max. Cluster ID* must equal the greatest integer serving as a cluster identifier in the current clustering run. In this first iteration, the Weka simple k-means algorithm was parameterized to discover $k = 100$ clusters. However, *Max. Cluster ID* is 99 according to the output of *Actions* → *Discover Patterns* → *Cluster Text Unit Vectors (Weka)*. Please refer to [GSW01] for a detailed description of *Cluster Quality Criteria*. However, decreasing *Min. Cardinality* or *Min. Frequent Ratio* as well as increasing *Max. Distinct Ratio* tends to result in a greater number of qualitatively "acceptable" clusters which are automatically assigned a default label in *Cluster Result File*.

After monitoring cluster quality, your preferred Web browser pops up and displays the HTML file `${PROJECT_HOME}/kddProcessIteration1/index.html`. As illustrated in Figure 3.19, it references all non-empty cluster files in the same directory. If the

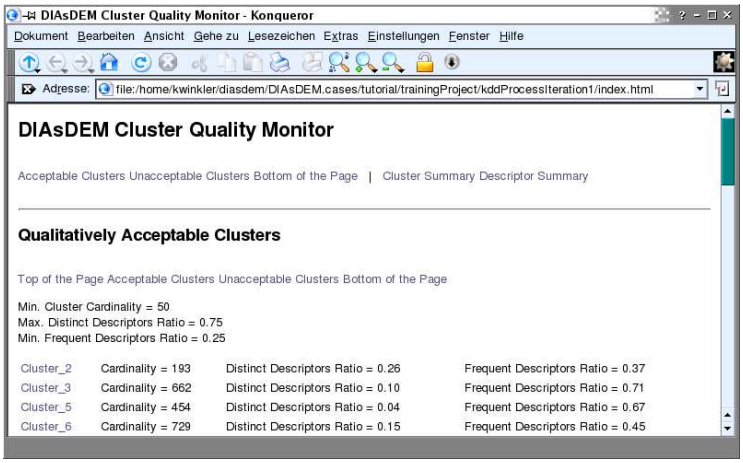

Figure 3.19: Cluster index file created by *Monitor Cluster Quality*

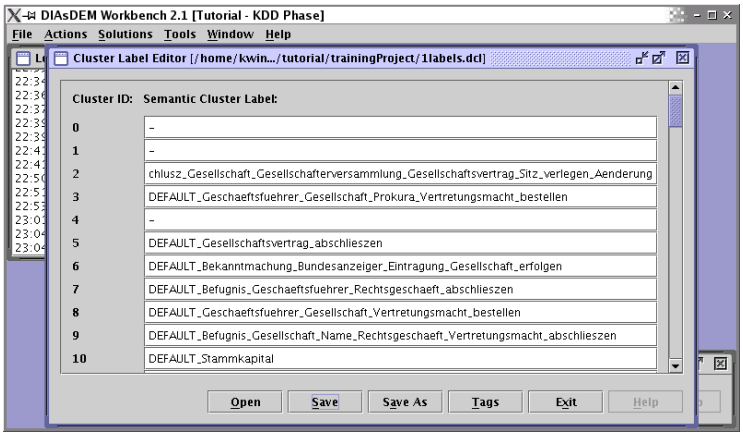

Figure 3.20: *Cluster Label Editor* of DIAsDEM Workbench 2.1

browser cannot be launched, check the current settings in the *External Programs* tab of the *Tools* → *Options* dialog. Figure 3.20 depicts *Cluster Label Editor*, which is also launched within DIAsDEM Workbench. This editor allows you to modify the automatically created cluster label file `${PROJECT_HOME}/1labels.dcl` by altering and deleting default cluster labels as well as semantically naming clusters without a default label. Using *Cluster Label Editor* to customize the file `${PROJECT_HOME}/1labels.dcl` is described in section 3.3.7. Nevertheless, close both *Cluster Label Editor* and the browser displaying `${PROJECT_HOME}/kddProcessIteration1/index.html`.

Monitor Cluster Quality: Summary

Module: *Actions → Postprocess Patterns → Monitor Cluster Quality*

Use Case: The user wants to separate qualitatively "acceptable" text unit vector clusters from "unacceptable" ones after clustering as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives.

Prerequisites: Clustering results in *Cluster Result File* must conform either to the DIAsDEM-specific CSV or to the DIAsDEM-specific TXT file format. Both formats are described in section 4.4.3 on page 113.

Result: The contents of all discovered text unit vector clusters are visualized as HTML files in *Clustering Directory*. Additionally, *Cluster Label File* contains default semantic labels for qualitatively "acceptable" clusters according to *Cluster Quality Criteria*. Additionally, the project properties *Default Collection File*, *Default Result File Format*, *Default Cluster Result File*, *Default Cluster Visualization Directory*, *Default Cluster Label File*, *Default Max. Cluster ID*, *Default Thesaurus File*, *Default Text Unit Descriptors*, *Default Min. Cardinality*, *Default Max. Distinct Ratio*, and *Default Min. Frequent Ratio* are set and updated, respectively.

Remarks: This module must only be executed in the KDD phase of the DIAsDEM framework as explained in section 1. In this phase, monitoring cluster quality and thereby creating *Cluster Label File* is a prerequisite for subsequently tagging text units using *Actions → Postprocess Patterns → Tag Text Units*. *Thesaurus File* and *Text Unit Descriptors* must exactly correspond to the parameters entered in *Actions → Prepare Data Set → Vectorize Text Units* in the same clustering iteration.

Monitor Cluster Quality: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project properties *Default Collection File*

KDD Process Iteration: Number of current KDD process iteration; default value: project property *Default Iteration (1, 2, ...)*

Result File Format: Choice of cluster result file format as described in section 4.4.3 on page 113 between comma separated values (CSV file) and fixed width values (TXT file); default value: project property *Default Result File Format*

Cluster Result File: Valid local file name of existing file; file extension depends on choice of *Result File Format*: `.csv` or `.txt`; default value: project property *Default Cluster Result File*

Cluster Directory: Valid local file name of existing directory or directory to be created by DIAsDEM Workbench; *Cluster Directory* should be empty; default value: project property *Default Cluster Visualization Directory*

Cluster Label File: Valid local file name of file to be created or replaced by DIAsDEM Workbench; file extension: `.dcl`; default value: project property *Default Cluster Label File*

Max. Cluster ID: integer greater than zero; corresponds to the greatest cluster identifier assigned by the clustering algorithm in the current iteration; default value: project property *Default Max. Cluster ID*

Thesaurus File: Valid local file name of existing DIAsDEM-specific thesaurus file as described in section 4.4.1 on page 109; file extension: `.dth`; default value: project property *Default Thesaurus File*

Text Unit Descriptors: If *All Descriptors in Thesaurus* is enabled, all descriptor terms in *Thesaurus File* are vector dimensions. If *Descriptors whose Scope Note Contain String* is enabled, only descriptor terms in *Thesaurus File* are valid whose scope notes contain the complete string entered below. If *Descriptors whose Scope Note Don't Contain String* is enabled, only descriptor terms in *Thesaurus File* are valid whose scope notes do not contain the string entered below.

Cluster Quality Criteria: One integer and two floating-point thresholds as described in [GSW01]; default values: project properties *Default Min. Cardinality*, *Default Max. Distinct Ratio* and *Default Min. Frequent Ratio*

Advanced Options: If the first line of *Cluster Result File* is a list of attribute names, *Ignore First Line of Cluster Result File* must be enabled. If *Ignore Empty Clusters in Cluster Index HTML File* is enabled, the index file in *Clustering Directory* does not contain links to HTML files of empty clusters. If *Launch Web Browser with Cluster Index HTML File* is enabled, the browser specified in *Tools* → *Options* is launched to display the index file of *Clustering Directory* after monitoring cluster quality. Analogously, if *Launch Cluster Label Editor with Cluster Label File* is enabled, *Tools* → *Cluster Label Editor* is launched to edit *Cluster Label File*. If *Dump DIAsDEM Documents for Visualization* is enabled, all documents comprising the collection are exported as XML files in subdirectories of *Cluster Directory*. In this case, HTML files that visualize cluster contents link each text unit to its original DIAsDEM document in order to allow a quick analysis of the text unit context.

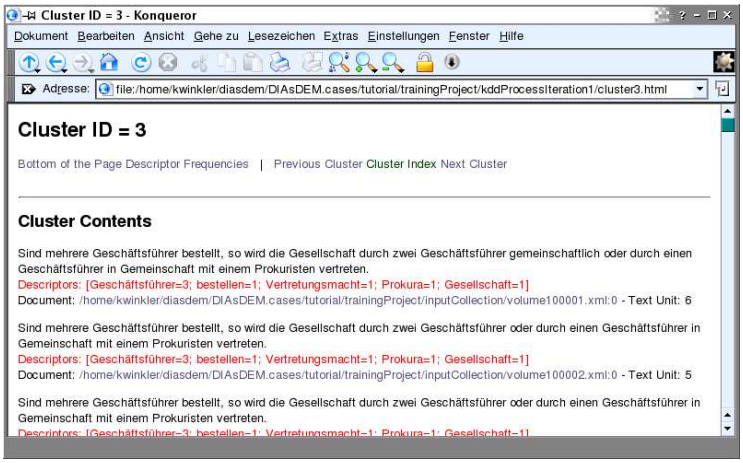

Figure 3.21: Top of HTML file visualizing the contents of cluster 3

3.3.7 Editing the Cluster Label File in Iteration 1

After clustering text unit vectors and monitoring cluster quality, the default *Cluster Label File* should be manually inspected by a domain specialist. The objective of this task is to assign each qualitatively "acceptable" cluster an appropriate semantic label. Semantic cluster labels should provide a concise and content-based description of the respective text units, because labels finally serve as elements of the XML document type definition to be derived. Text units whose vectors are assigned to semantically labeled clusters will be annotated by an XML tag that corresponds to the respective cluster label. The remaining text unit vectors are input to the clusterer in the next iteration.

Firstly, qualitatively "acceptable" clusters should be checked, which have been automatically assigned a default cluster label. In this case study, default German cluster labels have to be replaced by English labels manually. Furthermore, "acceptable" clusters may contain rather inhomogeneous text units according to the human sense of semantic similarity. For example, two opposite semantic concepts such as "to appoint" and "to dismiss" a managing director might be prevailing in the same cluster. In these cases, default labels should be deleted in *Cluster Label File* to enforce a re-clustering of the corresponding text unit vectors in the next KDD process iteration. Secondly, qualitatively "unacceptable" clusters should be inspected as well, because the applied cluster quality criteria cannot capture all cases of semantic similarity. For example, a cluster might contain text units that belong to a common semantic concept, although there are no statistically prevailing text unit descriptors.

Select *Tools* → *Cluster Label Editor* and open the file `${PROJECT_HOME}/1labels.dcl` by clicking on *Open* and choosing this cluster label file. As depicted in Figure 3.20, it

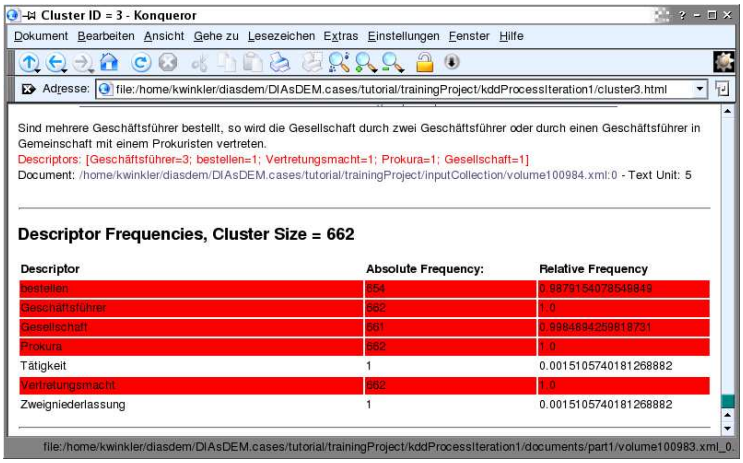

Figure 3.22: Descriptor frequencies at the bottom of cluster 3 HTML file

contains default cluster labels assigned during the first iteration. Additionally, open the index HTML file `${PROJECT_HOME}/kddProcessIteration1/index.html`. For each cluster, there exists an HTML file in the same directory that visualizes the cluster contents and that provides descriptive statistics of frequently occurring text unit descriptors.

Altogether, 16 qualitatively "acceptable" clusters and 34 non-empty "unacceptable" ones have been automatically discovered by the preceding task. For example, Figures 3.21 and 3.22 illustrate the HTML file visualizing the qualitatively "acceptable" cluster 3, which has been assigned the label "DEFAULT_Geschaeftsfuehrer_Gesellschaft_Prokura_Vertretungsmacht_bestellen". This default label has been created by concatenating text unit descriptors that prevail in cluster 3 and that are highlighted in Figure 3.22. Change the label of cluster 3 into its English equivalent "IfAppointmentOfManyManagingDirectors_JointPowerToRepresent" in *Cluster Label Editor* and click the *Save* button. Consider cluster 26, which is listed in the section "Qualitatively Unacceptable Clusters" of index file `${PROJECT_HOME}/kddProcessIteration1/index.html`. The German concept "Tätigkeit" occurs in all members of cluster 26. Therefore, this cluster can be manually labeled with its English equivalent "PurposeOfCompany". Inspect the remaining clusters and modify their respective semantic labels in *Cluster Label File* according to Table 3.2. Finally, close the Web browser, save *Cluster Label File* by clicking on *Save* and close *Cluster Label Editor* by clicking the *Exit* button.

Note, *Cluster Label Editor* of DIAsDEM Workbench 2.1 supports so-called tag proposal files to minimize human typing efforts. Tag proposal files can either be text files or DIAsDEM-specific files containing a previously derived, preliminary document type definition. For example, copy the cluster labels listed in Table 3.2 into an empty text file, whereas each line must exactly contain one label. After clicking the button *Tags*

Cluster ID	Semantic Cluster Label
2	ResolutionByShareholders_ChangeOfPlaceOfDomicile
3	IfAppointmentOfManyManagingDirectors_JointPowerToRepresent
5	ConclusionOfPartnershipAgreement
6	PublicationMediaOfCommercialRegisterEntries
8	IfAppointmentOfOneManagingDirector_SolePowerToRepresent
9	SolePowerToRepresent_PowerToContractWithOneself
10	ShareCapital
24	PowerToContractWithOneself
26	PurposeOfCompany
34	NumberOfLimitedPartners
35	ConclusionAndModificationOfPartnershipAgreement
42	LimitedPartnership
43	SolePowerToRepresent_PowerToContractWithOneself
49	LimitedLiabilityCompany
52	AppointmentOfManagingDirector

Table 3.2: Summary of semantic cluster labels in the first iteration

and selecting this new tag proposal file, *Cluster Label Editor* creates a drop-down menu beside each cluster label field. All drop-down menus comprise the same list of potentially useful cluster labels for the expert to choose from. Furthermore, tag proposal files can be imported into *Cluster Label Editor* one after the other.

3.3.8 Tagging Text Units in Iteration 1

After clustering text unit vectors (i.e., creating *Cluster Result File*), monitoring cluster quality and manually editing the resulting *Cluster Label File*, all intermediate DIAsDEM documents associated with *Collection File* have to be updated. Specifically, each text units whose vector has been input to the current clustering iteration should be annotated with the numerical identifier of the cluster it has been assigned to. In addition, members of qualitatively "acceptable" and thus labeled clusters have to be annotated with the respective semantic label as specified in *Cluster Label File*. Tagging text units is a prerequisite for exporting text unit vectors in the next iteration as well as for tagging entire documents (i.e., creating semantically annotated output XML documents) after the final clustering iteration. Hence, select *Actions* → *Postprocess Patterns* → *Tag Text Units* and type in the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
Iteration	`1`
Result File Format	`CSV: Comma Separated Values`

Cluster Result File	`${PROJECT_HOME}/1results.csv`
Cluster Label File	`${PROJECT_HOME}/1labels.dcl`
Advanced Options	`Disabled: Ignore First Line of Cluster Result File`

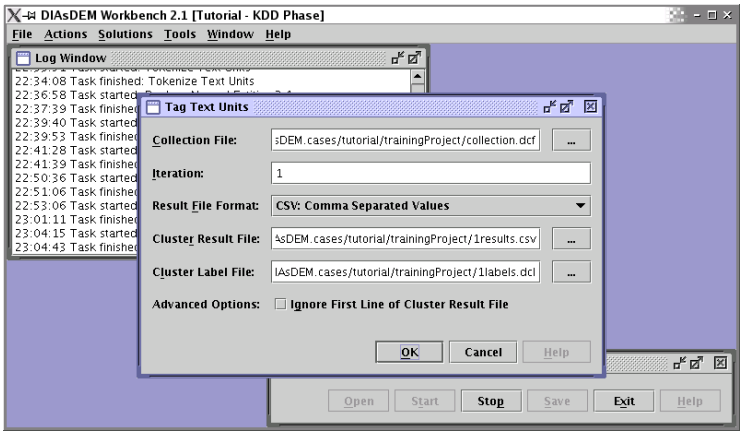

Figure 3.23: *Tag Text Units* dialog

The parameters *Iteration*, *Result File Format* and *Cluster Result File* are discussed in the previous section 3.3.6 in the context of monitoring cluster quality. *Cluster Label File* corresponds to the file that has been created by *Actions* → *Postprocess Patterns* → *Monitor Cluster Quality*. Click on *OK* to tag all text units accordingly. Thereafter, open the file `${PROJECT_HOME}/inputCollection/volume100878.xml`. The elements of section `<ProcessedTextUnits>` mark up the same contents as before. However, they have been extended by the attributes `Iteration`, `ClusterID` and `ClusterLabel` in order to keep track of cluster assignments. Values of `ClusterLabel` equal either "-" for unlabeled clusters or correspond to the semantic label of the respective cluster.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0"> ...
    <TextUnitsLayer TextUnitsLayerID="0" TextUnitsDescription="Algorithm:
    HEURISTIC_SENTENCE_IDENTIFIER"> ...
      <ProcessedTextUnits> ...
        <ProcessedTextUnit TextUnitID="2" Iteration="1" ClusterID="10"
        ClusterLabel="ShareCapital">Stammkapital : <NeRef NeID="0" /> ...
        <ProcessedTextUnit TextUnitID="8" Iteration="1" ClusterID="4"
        ClusterLabel="-">Einzelvertretungsbefugnis können erteilen werden .
        </ProcessedTextUnit><ProcessedTextUnit TextUnitID="9" Iteration="1"
        ClusterID="52" ClusterLabel="AppointmentOfManagingDirector"><NeRef
        NeID="16" />, sein zur Geschäftsführerin bestellen .</ProcessedTextUnit>
```

```
      ... <ProcessedTextUnit TextUnitID="11" Iteration="1" ClusterID="6"
      ClusterLabel="PublicationMediaOfCommercialRegisterEntries">nicht
      eintragen : d Bekanntmachung d Gesellschaft erfolgen im Bundesanzeiger
      .</ProcessedTextUnit>
     </ProcessedTextUnits> ...
   </TextUnitsLayer>
  </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

Consider the first text unit shown in the file excerpt above, which corresponds to the original sentence "Stammkapital: 50.000 DM." Its text unit vector has been assigned to cluster 10, which in turn has been labeled "ShareCapital". Thus, this sentence is subsequently tagged as "`<ShareCapital>` Stammkapital: 50.000 DM. `</ShareCapital>`". In contrast, the second text unit has been assigned to cluster 4, which remains unlabeled after the first clustering iteration. Recall, this text unit vector is input to the second clustering iteration. Finally, the text unit vector corresponding to the sentence "Marion Marcella Adolph geb. Priester, 22.03.1957, Offenbach, ist zur Geschäftsführerin bestellt." has been assigned to cluster 52, which has been semantically labeled "AppointmentOfManagingDirector". Note, all text unit vectors representing annotated text units are not re-clustered in the next iteration. Once a semantic label has been attached to a text unit, it cannot be changed anymore in DIAsDEM Workbench 2.1.

Executing the second clustering iteration is concisely described in section 3.3.9. Thereafter, sections 3.4.1 and 3.4.2 introduce the modules for deriving a preliminary XML document type definition and finally creating semantically tagged XML documents.

Tag Text Units: Summary

Module: *Actions → Postprocess Patterns → Tag Text Units*

Use Case: The user wants to annotate text units in intermediate DIAsDEM documents according to the results of monitoring cluster quality as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. Moreover, clustering results in *Cluster Result File* must conform either to the DIAsDEM-specific CSV or to the DIAsDEM-specific TXT file format. Both file formats are described in section 4.4.3 on page 113.

Result: In the first clustering iteration, the attributes `Iteration`, `ClusterID` and `ClusterLabel` of all processed text units are created or reset. In subsequent iterations, the section `<ProcessedTextUnits>` is updated. In both cases, all text units contained in *Cluster Result File* are annotated with

the iteration number, their current cluster ID and the corresponding label according to *Cluster Label File*. Additionally, the project properties *Default Collection File*, *Default Result File Format*, *Default Cluster Result File*, and *Default Cluster Label File* are set and updated, respectively.

Remarks: Tagging text units is a prerequisite for either starting the next clustering iteration or for finally executing the modules *Actions* → *Postprocess Patterns* → *Derive Preliminary DTD* and thereafter *Actions* → *Postprocess Patterns* → *Tag Documents*.

Tag Text Units: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Iteration: Number of current KDD process iteration; default value: project property *Default Iteration (1, 2, ...)*

Result File Format: Choice of cluster result file format as described in section 4.4.3 on page 113 between comma separated values (CSV file) and fixed width values (TXT file); default value: project property *Default Result File Format*

Cluster Result File: Valid local file name of existing file; file extension depends on choice of *Result File Format*: `.csv` or `.txt`; default value: project property *Default Cluster Result File*

Cluster Label File: Valid local file name of existing file created by DIAsDEM Workbench in *Actions* → *Postprocess Patterns* → *Monitor Cluster Quality* and possibly modified by *Tools* → *Cluster Label Editor*; file extension: `.dcl`; default value: project property *Default Cluster Label File*

Advanced Options: If the first line of *Cluster Result File* contains attribute names, *Ignore First Line of Cluster Result File* must be enabled.

3.3.9 Summary of KDD Process Iteration 2

Text units are re-clustered in iteration 2, if their vectors have not been assigned to a qualitatively "acceptable" cluster in the first iteration. Consequently, text unit vectors corresponding to unlabeled sentences need to be exported and clustered again. After monitoring cluster quality and creating a new *Cluster Label File*, text units have to be tagged according to the results of the second iteration. These steps of the DIAsDEM KDD process have been discussed in detail in sections 3.3.4 through 3.3.8. Hence, this section only summarizes parameter settings and briefly explains particularities.

Firstly, text unit vectors, which constitute the input data set to iteration 2, should be exported. Please select *Actions* → *Prepare Data Set* → *Vectorize Text Units*, type in the following parameters and click on *OK*.

Parameter	Value
Collection File	${PROJECT_HOME}/collection.dcf
KDD Process Iteration	2
Text Unit Vectors Format	ARFF: Weka Data Mining Project
Text Unit Vectors File	${PROJECT_HOME}/2vectors.arff
Thesaurus File	${PARAMETER_HOME}/thesauri/de/Case123Thesaurus.dth
Text Unit Descriptors	Descriptors whose Scope Notes Contain String Case1
Descriptor Frequency	Boolean Descriptor Frequency
Descriptor Weights	Create Descriptor Weights File: IDF Weights ${PROJECT_HOME}/2weights.ddw
Advanced Options	Disabled: Create File for Mining Descriptor Association Rules Enabled: Create Meta-Data File for Text Unit Vectors File

Compared to the first iteration, *Text Unit Vectors File* contains approx. one quarter of text unit vectors. Note additionally, collection-based term weights such as inverse document frequency are always computed on the basis of the remaining text unit vectors. For example, compare the different term weights for iteration 1 and 2, which are listed in the meta-data files ${PROJECT_HOME}/1vectors.arff.meta and 2vectors.arff.meta, respectively. In order to cluster exported text unit vectors, select *Actions* → *Discover Patterns* → *Cluster Text Unit Vectors (Weka)*, enter the following parameters and click on the *OK* button. In contrast to iteration 2, the maximum number of clusters to be discovered by k-means is 50.

Parameter	Value
Clustering Mode	Clustering Phase (Create New Clustering Model)
Collection File	${PROJECT_HOME}/collection.dcf
Text Unit Vectors File	${PROJECT_HOME}/2vectors.arff
Clustering Algorithm	weka.clusterers.SimpleKMeans
Clustering Parameters	1) Number of Clusters = 50 2) Acuity = 3) Cutoff = 4) Max. Iterations = 5) Random Number Seed = 6) Min. Std. Deviation =
Clustering Results File	${PARAMETER_HOME}/2results.csv
Text Unit Clusterer File	${PARAMETER_HOME}/2clusterer.wskm

Analogously to iteration 1, clustering text unit vectors is followed by monitoring cluster quality. Hence, select *Actions* → *Postprocess Patterns* → *Monitor Cluster Quality*, submit the following parameters and click on *OK*. Compared with the first iteration, the cluster cardinality threshold should be decreased from 50 to 25.

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
KDD Process Iteration	`2`
Result File Format	`CSV: Comma Separated Values`
Cluster Result File	`${PROJECT_HOME}/2results.csv`
Cluster Directory	`${PROJECT_HOME}/kddProcessIteration2`
Cluster Label File	`${PROJECT_HOME}/2labels.dcl`
Max. Cluster ID	`49`
Thesaurus File	`${PARAMETER_HOME}/thesauri/de/Case123Thesaurus.dth`
Text Unit Descriptors	`Descriptors whose Scope Notes Contain String` `Case1`

Cluster Quality Criteria	`1) Min. Cardinality = 25`
	`2) Max. Distinct Ratio = 0.75`
	`3) Min. Frequent Ratio = 0.25`

Advanced Options	`Disabled: Ignore First Line of Cluster Result File`
	`Enabled: Ignore Empty Clusters in Cluster Index HTML File`
	`Enabled: Launch Web Browser with Cluster Index HTML File`
	`Enabled: Launch Cluster Label Editor with Cluster Label File`
	`Enabled: Dump DIAsDEM Documents for Visualization`

After monitoring cluster quality, the Web browser pops up and displays the HTML file `${PROJECT_HOME}/kddProcessIteration2/index.html`, which references all non-empty cluster files in the same directory. Additionally, *Cluster Label Editor* is launched within DIAsDEM Workbench. DIAsDEM Workbench has automatically discovered six qualitatively "acceptable" clusters only. Please have a look at these clusters and modify the file `${PROJECT_HOME}/2labels.dcl` in *Cluster Label Editor* according to Table 3.3.

Cluster ID	Semantic Cluster Label
1	SolePowerToRepresentCanBeGranted
3	FullyLiablePartner
4	JointStockCompany
8	PurposeOfCompany
10	PurposeOfCompany
15	CommencementOfPartnership
19	PurposeOfCompany
23	AppointmentOfManagingDirector
36	ResolutionByShareholders_ChangeOfPlaceOfDomicile
41	ChangeOfFirmName
42	ConfermentOfProkura
48	NameOfMerchant

Table 3.3: Summary of semantic cluster labels in the second iteration

After editing `${PROJECT_HOME}/2labels.dcl`, select *Actions* → *Postprocess Patterns* → *Tag Text Units*, type in the following parameters and click the *OK* button to annotate text units accordingly:

Parameter	Value
Collection File	${PROJECT_HOME}/collection.dcf
Iteration	2
Result File Format	CSV: Comma Separated Values
Cluster Result File	${PROJECT_HOME}/2results.csv
Cluster Label File	${PROJECT_HOME}/2labels.dcl
Advanced Options	Disabled: Ignore First Line of Cluster Result File

Open the file ${PROJECT_HOME}/inputCollection/volume100878.xml and consider the original sentence "Einzelvertretungsbefugnis kann erteilt werden.", which has not been tagged in the first iteration. However, the corresponding text unit vector has been assigned to cluster 1 in the second clustering. This cluster has been manually labeled to become a qualitatively "acceptable" one. Consequently, "'Einzelvertretungsbefugnis kann erteilt werden." has been annotated with the corresponding semantic label "SolePowerToRepresentCanBeGranted" of cluster 1. Note again that annotations and cluster IDs assigned in the first iteration remain untouched. For example, the original sentence "Stammkapital: 50.000 DM." is still assigned to cluster 10 of the first iteration and is still labeled "ShareCapital".

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DefaultDIAsDEMvolume SYSTEM "DefaultDIAsDEMvolume.dtd">
<DefaultDIAsDEMvolume NumberOfDocuments="1">
  <DefaultDIAsDEMdocument NumberOfTextUnitsLayers="0"> ...
    <TextUnitsLayer TextUnitsLayerID="0" TextUnitsDescription="Algorithm:
    HEURISTIC_SENTENCE_IDENTIFIER"> ...
      <ProcessedTextUnits>
        <ProcessedTextUnit TextUnitID="0" Iteration="1" ClusterID="26"
        ClusterLabel="PurposeOfCompany">d Handel mit Ware alle Art sowie
        Import und Export .</ProcessedTextUnit><ProcessedTextUnit
        TextUnitID="2" Iteration="1" ClusterID="10" ClusterLabel=
        "ShareCapital">Stammkapital : <NeRef NeID="0" /> ...
        <ProcessedTextUnit TextUnitID="8" Iteration="2" ClusterID="1"
        ClusterLabel="SolePowerToRepresentCanBeGranted">Einzelvertretungs-
        befugnis können erteilen werden .</ProcessedTextUnit>
        <ProcessedTextUnit TextUnitID="9" Iteration="1" ClusterID="52"
        ClusterLabel="AppointmentOfManagingDirector"><NeRef NeID="16" />,
        sein zur Geschäftsführerin bestellen .</ProcessedTextUnit>
        ... <ProcessedTextUnit TextUnitID="11" Iteration="1" ClusterID="6"
        ClusterLabel="PublicationMediaOfCommercialRegisterEntries">nicht
        eintragen : d Bekanntmachung d Gesellschaft erfolgen im Bundesanzeiger
        .</ProcessedTextUnit>
      </ProcessedTextUnits> ...
    </TextUnitsLayer>
  </DefaultDIAsDEMdocument>
</DefaultDIAsDEMvolume>
```

In this case study, only two clustering iterations are performed in order to exemplify the interactive and iterative DIAsDEM KDD process. When applying the DIAsDEM KDD process to real document archives, the iterative clustering should be continued until further qualitatively "acceptable" clusters cannot be discovered. Nevertheless, approx. three quarters or 7,066 (76.4%) of altogether 9,254 text units have been assigned a semantic label in the training phase of this case study.

3.4 XML Tagging of Texts in the KDD Phase

After finishing the second, in our case the final clustering iteration, text documents have to be converted into an archive of semantically tagged XML documents in order to reach the objectives of the DIAsDEM framework. Hence, a collection-specific XML document type definition has to be derived in the postprocessing phase of the DIAsDEM KDD process. Thereafter, semantically tagged XML documents, which conform to this XML DTD, can be constructed from the collection of intermediate DIAsDEM documents.

3.4.1 Deriving a Preliminary XML DTD

A preliminary XML document type definition concisely describes frequently occurring, collection-specific semantic concepts in the form of DTD elements, which can be either XML tags or attributes of XML tags. The latter correspond to named entity types whose instances exceed a relative frequency threshold within all text units annotated with the respective tag. To continue, select the task *Actions* → *Postprocess Patterns* → *Derive Preliminary DTD* and input the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
Preliminary DTD File	`${PROJECT_HOME}/preliminaryDtd.dpd`
DTD Root Element	`CommercialRegisterEntry`
Min. Attribute Support	`0.1`

Note, *Preliminary DTD File* is a DIAsDEM-specific file that contains meta-data about the XML DTD for internal usage only. According to *DTD Root Element*, the term `CommercialRegisterEntry` is the root element of the XML document type definition. Hence, `CommercialRegisterEntry` is the root tag of output XML documents, which are created afterwards. Due to *Min. Attribute Support*, named entity type e such as "date" only qualifies as an attribute of XML tag t, if instances of e (e.g., "2003-03-31" and "2003-04-01") occur in at least 10% of all text units annotated with t. Click the *OK* button to derive the DIAsDEM-specific *Preliminary DTD File*, which is a required input parameter for tasks that create final, semantically annotated output documents such as *Actions* → *Postprocess Patterns* → *Tag Documents*.

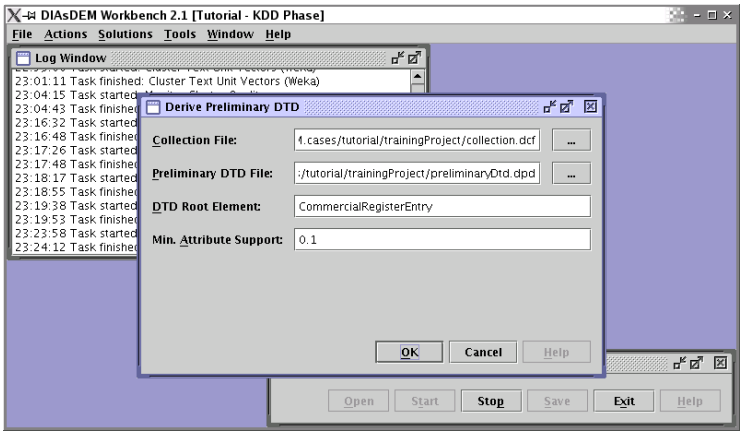

Figure 3.24: *Derive Preliminary DTD* dialog

${PROJECT_HOME} now contains four new files, whereas only `preliminaryDtd.dpd` is referred to as *Preliminary DTD File*. The auxiliary files `preliminaryDtd.dpd.elements`, `preliminaryDtd.dpd.attributes`, and `preliminaryDtd.dpd.xml` are referenced by *Preliminary DTD File* and must thus always reside in the same directory. DIAsDEM Workbench 2.1 derives a preliminary, rather unstructured XML document type definition that simply enumerates occurring DTD elements (i.e., XML tags) and attributes associated with XML tags. Currently, DIAsDEM Workbench supports neither the discovery of nested XML tags nor the identification of frequently occurring sequences of XML tags within annotated documents. Using any text editor, open the DTD file `preliminaryDtd.dpd.xml`, which is located in ${PROJECT_HOME}.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT CommercialRegisterEntry (MetaData*, TaggedDocument)>

<!ELEMENT MetaData (Name, Content)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Content (#PCDATA)>

<!ELEMENT TaggedDocument ( #PCDATA | AppointmentOfManagingDirector
 | ChangeOfFirmName | CommencementOfPartnership | ... | ConfermentOfProkura
 | FullyLiablePartner | ... | JointStockCompany | LimitedLiabilityCompany
 | LimitedPartnership | NameOfMerchant | NumberOfLimitedPartners | ShareCapital
 | ... | SolePowerToRepresent_PowerToContractWithOneself
)* >

<!ELEMENT AppointmentOfManagingDirector (#PCDATA)>
<!ELEMENT ChangeOfFirmName (#PCDATA)>
```

```
<!ELEMENT CommencementOfPartnership (#PCDATA)> ...
<!ELEMENT SolePowerToRepresent_PowerToContractWithOneself (#PCDATA)>

<!ATTLIST AppointmentOfManagingDirector Date CDATA #IMPLIED>
<!ATTLIST AppointmentOfManagingDirector Person CDATA #IMPLIED>
<!ATTLIST AppointmentOfManagingDirector Place CDATA #IMPLIED>
<!ATTLIST ChangeOfFirmName Company CDATA #IMPLIED> ...
<!ATTLIST ShareCapital AmountOfMoney CDATA #IMPLIED>
```

Valid output documents (i.e., annotated Commercial Register entries) might consist of two main sections: Meta-data stored in DIAsDEM documents is copied into the optional section `<MetaData>` to facilitate further data processing. The mandatory section `<TaggedDocument>` includes semantically annotated text. Elements of the latter section are defined as a listing of unordered DTD elements, whereas semantic tags can occur anywhere in the text. Furthermore, attributes of XML tags are defined as well. For example, the XML tag `AppointmentOfManagingDirector` has three optional attributes `Date`, `Person`, and `Place`. Due to the setting of *Min. Attribute Support*, instances of named entity types "date", "person", and "place" occur in at least 10% of all text units annotated with `AppointmentOfManagingDirector`. Note, semantically tagged XML documents created by *Actions* → *Postprocess Patterns* → *Tag Documents* are valid XML documents with respect to this DTD. However, attributes of XML tags cannot be semantically named in this release of DIAsDEM Workbench.

Derive Preliminary DTD: Summary

Module: *Actions* → *Postprocess Patterns* → *Derive Preliminary DTD*

Use Case: The user wants to derive a preliminary XML DTD from semantically annotated DIAsDEM documents as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. All elements `<ProcessedTextUnit>` must have the attributes `Iteration`, `ClusterID`, and `ClusterLabel`.

Results: A collection-specific XML document type definition is derived, which enumerates valid XML tags and their attributes. Additionally, the project properties *Default Collection File*, *Default Preliminary DTD File*, *Default DTD Root Element*, and *Default Min. Attribute Support* are set and updated, respectively.

Remarks: After deriving the collection-specific preliminary DTD, semantically annotated XML documents can be output by *Actions* → *Postprocess Patterns* → *Tag Documents*.

Derive Preliminary DTD: Summary

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Preliminary DTD File: Valid local file name of file to be created or replaced by DIAsDEM Workbench; file extension: `.dpd`; default value: project property *Default Preliminary DTD File*

DTD Root Element: ISO-8859-1 encoded string without blank spaces; default value: project property *Default DTD Root Element*

Min. Attribute Support: Floating point threshold in the interval $[0; 1]$: named entity type e only qualifies as attribute of XML tag t, if instances of e occur in at least the specified portion of all text units annotated with t; default value: project property *Default Min. Attribute Support*

3.4.2 Tagging Documents

After deriving an archive-specific document type definition, output XML documents are created by assembling both tagged (i.e., text units whose vectors have been assigned to a semantically labeled cluster) and untagged text units in the order of their occurrence in the original text. Besides the derived XML document type definition and the generated text unit clusterer for subsequent batch processing, semantically tagged XML documents constitute the main output of the DIAsDEM KDD process. To proceed, select *Actions* → *Postprocess Patterns* → *Tag Documents* and enter the following parameters:

Parameter	Value
Collection File	`${PROJECT_HOME}/collection.dcf`
XML Document Directory	`${PROJECT_HOME}/outputXmlDocuments`
Preliminary DTD File	`${PROJECT_HOME}/preliminaryDtd.dpd`
Random Sample File	`${PROJECT_HOME}/outputSampleFiles/textUnitSample5Pct.dts`
Random Sample Size	`0.05`

Advanced Options	`Enabled: Create Tag-by-Document-Matrix as CSV-File`
	`Disabled: Create Log Files for Tag Analysis with WUM`
	`Disabled: Export XML Documents as GATE Files. Directory:`

Click the *OK* button to start the semantic tagging of documents. Besides outputting semantically tagged XML documents in subdirectories of *XML Document Directory*, the module *Actions* → *Postprocess Patterns* → *Tag Documents* draws a 5% random text unit sample. This sample is saved in the DIAsDEM-specific *Random Sample File* for subsequent evaluation of tagging quality. For each intermediate DIAsDEM document in the collection, a new XML file is output, which contains semantically annotated contents of the corresponding text. For example, the result XML document `${PROJECT_HOME}/outputXmlDocuments/part1/volume100878.xml` is depicted below.

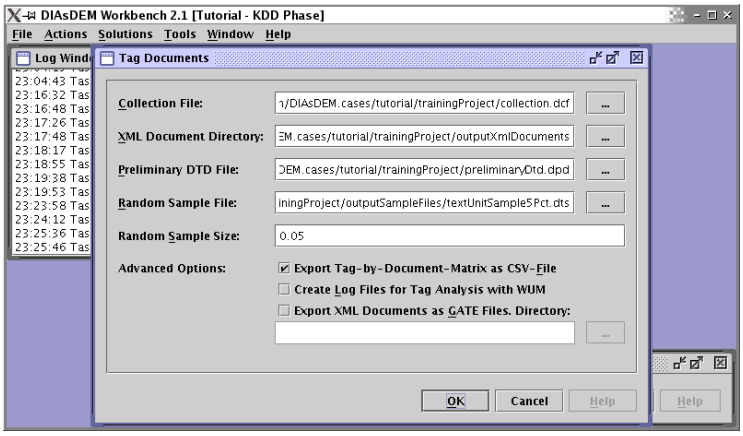

Figure 3.25: *Tag Documents* dialog

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CommercialRegisterEntry SYSTEM "CommercialRegisterEntry.dtd">
<CommercialRegisterEntry>
  <MetaData>
    <Name>DiasdemDocumentID</Name>
    <Content>/home/.../trainingProject/inputCollection/volume100878.xml:0</Content>
  </MetaData>
  <MetaData>
    <Name>SourceFile</Name>
    <Content>/home/.../data/samples/de/case1/file10780.training.txt</Content>
  </MetaData>
  <TaggedDocument>
    <PurposeOfCompany>Der Handel mit Waren aller Art sowie Import und Export.
    </PurposeOfCompany> ... <ShareCapital AmountOfMoney="50000 DEM">Stammkapital:
    50.000 DM.</ShareCapital><LimitedLiabilityCompany>Gesellschaft mit beschränkter
    Haftung.</LimitedLiabilityCompany> ... <SolePowerToRepresentCanBeGranted>
    Einzelvertretungsbefugnis kann erteilt werden.</SolePowerToRepresentCanBeGranted>
    <AppointmentOfManagingDirector Person="16; Marion Marcella Adolph; null; null;
    null; 22.03.1957; Priester; Offenbach; null; null">Marion Marcella Adolph geb.
    Priester, 22.03.1957, Offenbach, ist zur Geschäftsführerin bestellt.
    </AppointmentOfManagingDirector> ... <PublicationMediaOfCommercialRegisterEntries>
    Nicht eingetragen: Die Bekanntmachungen der Gesellschaft erfolgen im
    Bundesanzeiger.</PublicationMediaOfCommercialRegisterEntries>
  </TaggedDocument>
</CommercialRegisterEntry>
```

The maximum number of result files per subdirectory of *XML Document Directory* is determined by the project property *Maximum Files per Directory*. The supplementary

Random Sample File contains a random sample of text units for subsequent evaluation of tagging quality using *Tools → Assessment of Tagging Quality*. In addition, a random sample of completely tagged documents can be created by the task *Actions → Postprocess Patterns → Draw Document Sample*. Open `${PROJECT_HOME}/outputXmlDocuments/TagByDocumentMatrix.csv`, which is partly shown below. Each line contains a relational representation of semantic XML tags that occur in a certain text document. For example, the XML tag `AppointmentOfManagingDirector` occurs in the XML file created from source document `/home/.../file10780.training.txt`, whereas the tag `ChangeOfFirmName` does not occur in this file. This CSV file can easily be imported in any relational database for further analysis.

```
DiasdemDocumentID,SourceFile,AppointmentOfManagingDirector,ChangeOfFirmName,...
"/home/.../volume100000.xml:0","/home/.../file10511.training.txt",1,0,0,1,0,...
"/home/.../volume100001.xml:0","/home/.../file10338.training.txt",1,0,0,1,0,...
...
"/home/.../volume100878.xml:0","/home/.../file10780.training.txt",1,0,0,1,0,...
...
```

Tag Documents: Summary

Module: *Actions → Postprocess Patterns → Tag Documents*

Use Case: The user wants to create result XML documents from semantically annotated DIAsDEM documents as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. All elements `<ProcessedTextUnit>` must have the attributes `Iteration`, `ClusterID` and `ClusterLabel`. A collection-specific preliminary XML DTD must have been derived.

Result: For each intermediate DIAsDEM document, a semantically annotated XML file is output. Additionally, the project properties *Default Collection File*, *Default Preliminary DTD File*, *Default Random Sample File*, and *Default Random Sample Size* are set and updated, respectively.

Remarks: After tagging result documents, only one task remains to be done: The quality of semantic tags should be evaluated using the module *Tools → Assessment of Tagging Quality*.

Tag Text Units: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Preliminary DTD File: Valid local file name of exiting file; file extension: `.dpd`; default value: project property *Default Preliminary DTD File*

Random Sample File: Valid local file name of file to be created or replaced by DIAsDEM Workbench; file extension: `.dts`; default value: project property *Default Random Sample File*

Random Sample Size: Floating point number in the interval $[0; 1]$; proportion of text units to be randomly drawn for quality evaluation; default value: project property *Default Random Sample Size*

Advanced Options: If *Create Tag-by-Document-Matrix as CSV-File* is enabled, a file `TagByDocumentMatrix.csv` is created in *XML Document Directory* that contains a relational mapping of source file names onto discovered XML tags. If *Create Log Files for Tag Analysis with WUM* is enabled, all sequences of XML tags in result files are exported into a log file for subsequent sequence mining and association rules discovery using WUM: The Web Utilization Miner. If *Export XML Documents as GATE Files* is enabled, semantically annotated GATE files are exported into the specified directory.

3.4.3 Evaluating the Tagging Quality

Finally, the quality of semi-automatically created semantic XML mark up should be evaluated. Due to the absence of pre-tagged documents, a random sample of both tagged and untagged text units has been drawn by *Actions* → *Postprocess Patterns* → *Tag Documents* to allow a quality assessment. Now, a domain specialist should verify the annotations of randomly chosen text units with respect to the following error types:

- *Error type I (false positive):* A text unit is annotated with a wrong XML tag, because the tag does not properly reflect the contents of the text unit.
- *Error type II (false negative):* A text unit is not annotated at all, although there exists an XML tag in the derived DTD that reflects the contents of the text unit.

In contrast to the quality assessment of semantic annotations, the accuracy of named entity extraction cannot be evaluated in DIAsDEM Workbench 2.1. Nevertheless, start the quality evaluation by selecting *Tools* → *Assessment of Tagging Quality*, clicking the *Start* button and choosing the following five parameter files one after the other:

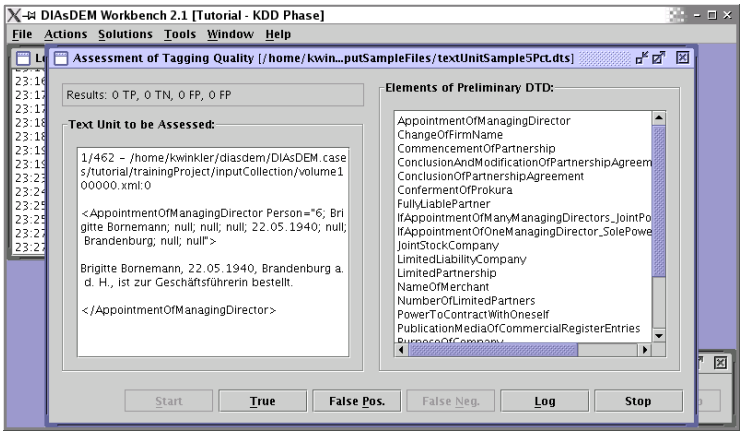

Figure 3.26: *Assessment of Tagging Quality* tool (1st session)

Parameter	Value
Existing Text Unit Sample File	`${PROJECT_HOME}/outputSamplesFile/textUnitSample5Pct.dts`
New or Existing File of Evaluated Text Units	`${PROJECT_HOME}/outputSamplesFile/evaluatedTextUnits.det`
Text Unit Sample File to be Created for Next Evaluation	`${PROJECT_HOME}/outputSamplesFile/textUnitSample5PctB.dts`
Existing Preliminary DTD File	`${PROJECT_HOME}/preliminaryDtd.dpd`
New or Existing Log for Personal Notes	`${PROJECT_HOME}/outputSamplesFile/evaluationLog.txt`

The parameter files *Existing Text Unit Sample File* and *Existing Preliminary DTD File* have been created before. They are concisely described in sections 3.4.1 and 4.5.2, respectively. Evaluating tagging quality can be a lengthy task, even for rather small text unit samples. Hence, DIAsDEM Workbench supports the assessment of tagging quality in multiple sessions. In the first assessment session, a new file *New or Existing File of Evaluated Text Units* is created. It contains the domain specialist's decision for each text unit as well as the text unit itself. After clicking the *Stop* button, text units that remain to be evaluated in subsequent sessions are copied into *Text Unit Sample File to be Created for Next Evaluation*. In the next session, this file *Text Unit Sample File to be Created for Next Evaluation* must be chosen as *Existing Text Unit Sample File*.

Figure 3.26 depicts the *Assessment of Tagging Quality* tool after opening the *Existing Text Unit Sample File* `textUnitSample5Pct.dts` in the first assessment session. In the left pane, the current text unit to be assessed is displayed along with its semantic tag if present. The entire set of XML tags as contained in the derived, collection-specific XML DTD is shown in the right pane. Note that you probably have to evaluate different sentences, since text units are randomly chosen. For obvious reasons, tagged sentences can either be true positives (i.e., having a correct XML tag) or false positives (i.e.,

having a false XML tag). On the other hand, text units that have not been semantically annotated by DIAsDEM Workbench can either be true negatives (i.e., appropriate XML tags are not contained in DTD) or false negatives (i.e., appropriate XML tags are actually part of DTD). Please assess 10 text units by clicking the appropriate buttons *True*, *False Pos.*, and *False Neg.*, respectively. Thereafter, click on *Stop* and open the file `${PROJECT_HOME}/outputSamplesFile/evaluatedTextUnits.det`:

```
# TP,TN,FP,FN,Type,TextUnit
1,0,0,0,"TP","/home/.../volume100000.xml:0 <AppointmentOfManagingDirector
  Person="6; Brigitte Bornemann; null; null; null; 22.05.1940; null; Brandenburg;
  null; null">Brigitte Bornemann, 22.05.1940, Brandenburg a. d. H., ist zur
  Geschäftsführerin bestellt.</AppointmentOfManagingDirector>"
1,0,0,0,"TP","/home/.../volume100003.xml:0 <PurposeOfCompany>(Gegenstand:
  Durchführung von vermessungstechnischen Arbeiten).</PurposeOfCompany>"
1,0,0,0,"TP","/home/.../volume100005.xml:0 <NameOfMerchant Person="3; Stefan
  Thümmler; null; null; null; null; null; null; null; null">Inhaber: Stefan
  Thümmler, Kaufmann, Wustermark.</NameOfMerchant>"
```

The results of each assessment session are appended to `evaluatedTextUnits.det`. After completing the quality evaluation, this file can be renamed `evaluatedTextUnits.csv` and imported into any spreadsheet application for detailed analysis. Note, the number of text units contained in the entire training collection is listed in *Existing Preliminary DTD File* as the property `NUMBER_OF_TEXT_UNITS`. Close the *Tagging Quality Evaluation* window to simulate the end of the current session. Thereafter, the second assessment session can be started by again selecting *Tools* → *Assessment of Tagging Quality*, clicking the *Start* button and choosing the following five parameter files one after the other:

Parameter	Value
Existing Text Unit Sample File	`${PROJECT_HOME}/outputSamplesFile/textUnitSample5PctB.dts`
New or Existing File of Evaluated Text Units	`${PROJECT_HOME}/outputSamplesFile/evaluatedTextUnits.det`
Text Unit Sample File to be Created for Next Evaluation	`${PROJECT_HOME}/outputSamplesFile/textUnitSample5PctC.dts`
Existing Preliminary DTD File	`${PROJECT_HOME}/preliminaryDtd.dpd`
New or Existing Log for Personal Notes	`${PROJECT_HOME}/outputSamplesFile/evaluationLog.txt`

Figure 3.27 illustrates the *Tagging Quality Evaluation* window at the beginning of the second assessment session. In the left pane, the results of previous sessions as contained in `${PROJECT_HOME}/outputSamplesFile/evaluatedTextUnits.det` are displayed as well. If you have time, you might assess the remaining 452 text units by clicking the appropriate buttons *True*, *False Pos.*, and *False Neg.*, respectively.

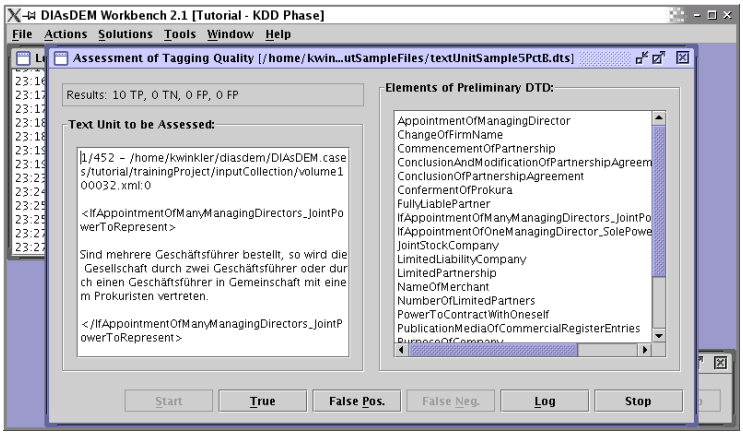

Figure 3.27: *Assessment of Tagging Quality* tool (2nd session)

3.4.4 Stopping the Batch Script Recorder

Before stopping the batch script recording, close the project by selecting *File* → *Close Project*. Thereafter, stop recording the tasks performed during the KDD phase by clicking the button *Stop* of *Batch Script Recorder*. Save the recorded batch script by clicking *Save* and choosing the file name `${PROJECT_HOME}/batchScripts/training.dsc`. Figure 3.28 depicts DIAsDEM Workbench after stopping the recording and saving the script. The KDD phase of the DIAsDEM framework is now finished. As explained in section 3.5, the new batch script has to be edited before it can be executed to tag text archives in the application phase.

3.5 Summary of the Application Phase

Before this case is finished, 161 Commercial Register entries remain to be semantically tagged in the application phase of the DIAsDEM framework. The corresponding files (extension: `.application.txt`) are located in the directory `${SAMPLES_HOME}/de/case1`. The term "application phase" refers to the activity of iteratively applying previously created text unit clusterers in order to convert new text documents into semantically tagged XML documents. Output XML documents conform to the preliminary XML document type definition, which was derived in the KDD phase. Note, texts to be tagged in the application phase must have similar contents as the training documents for obvious reasons. However, you might for example process a stratified sample of text documents in the KDD phase.

In contrast to the interactive training phase of the DIAsDEM framework, text documents are semantically annotated in an automated batch process, which does not require

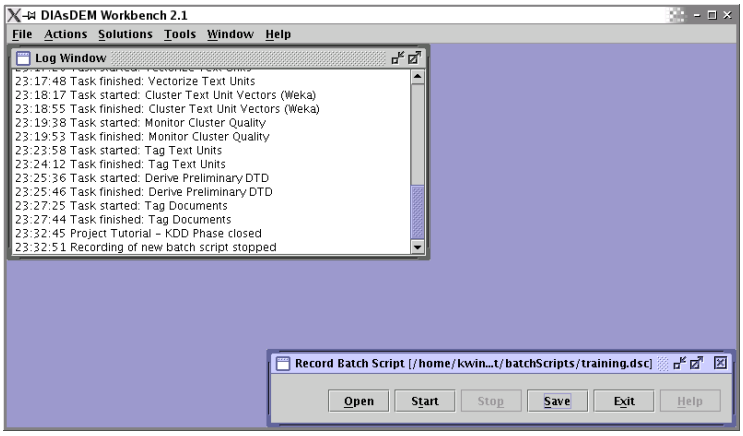

Figure 3.28: DIAsDEM Workbench 2.1 after saving the batch script

any human intervention at all. The application phase of the DIAsDEM framework differs from training text unit clusterers in the following steps:

- When vectorizing text units, an existing iteration-specific text unit descriptor weights file is applied to weight terms frequencies instead of creating a new one.
- When clustering text unit vectors, an existing iteration-specific text unit clusterer, which is often referred to as "score code", is applied to assign vectors to clusters.
- Monitoring the quality of text unit vector clusters is not necessary, because there is not need to create and edit a new cluster label file in the application phase.
- When tagging text units, an existing iteration-specific cluster label file is applied to tag text units according to the cluster labels assigned in the KDD phase.
- Deriving a preliminary XML DTD is not necessary, because output documents must conform to the preliminary XML DTD derived in the KDD phase.

The term "application parameters" encompasses certain parameter files, which are created in the KDD phase for subsequent usage in the application phase of the DIAsDEM framework. For each KDD process iteration, three files constitute specific application parameters: *Descriptor Weights File* listing iteration-specific weights, *Text Unit Clusterer File* including an iteration-specific clustering model, and *Cluster Label File* containing mappings from cluster IDs onto iteration-specific cluster labels. Furthermore, *Preliminary DTD File* along with its three auxiliary files are application parameters as well. When applying text unit clusterers to a new collection, the remaining parameter settings and the sequence of tasks must exactly correspond to the training procedure.

3.5.1 Preparing the Application Phase

The application phase of this case study constitutes a new project, whose files should reside in a dedicated project directory. To avoid confusing training and application phase, the abbreviation ${APP_PROJECT_HOME} corresponds to the directory /home/kwinkler/diasdem/DIAsDEM.cases/tutorial/applicationProject in the remainder of this case study. Create a new local directory ${APP_PROJECT_HOME} on your machine and copy the entire directory template for application projects into this directory:

```
.cases/tutorial> pwd
/home/kwinkler/diasdem/DIAsDEM.cases/tutorial
.cases/tutorial> cp -R ../../DIAsDEM.workbench21/data/templates/applicationProject .
.cases/tutorial> ls
applicationProject trainingProject
```

Thereafter, copy all application parameter files from the training project directory into the new directory: These files must include a *Descriptor Weights File* for each iteration, a *Text Unit Clusterer File* for each iteration, a *Cluster Label File* for each iteration, and the *Preliminary DTD File* along with its three auxiliary files. Furthermore, copy the previously recorded DIAsDEM batch script into the respective application subdirectory.

```
.cases/tutorial> pwd
/home/kwinkler/diasdem/DIAsDEM.cases/tutorial
.cases/tutorial> cp trainingProject/*.ddw applicationProject
.cases/tutorial> cp trainingProject/*.wskm applicationProject
.cases/tutorial> cp trainingProject/*.dcl applicationProject
.cases/tutorial> cp trainingProject/preliminary* applicationProject
.cases/tutorial> ls applicationProject
1clusterer.wskm  applicationParameters outputSqlScripts
1labels.dcl      batchScripts          outputXmlDocuments
1weights.ddw     inputCollection       preliminaryDtd.dpd
2clusterer.wskm  outputGateDocuments   preliminaryDtd.dpd.attributes
2labels.dcl      outputNeex21Files     preliminaryDtd.dpd.elements
2weights.ddw     outputSampleFiles     preliminaryDtd.dpd.xml
.cases/tutorial> cp trainingProject/preliminary* applicationProject/
.cases/tutorial> cd applicationProject/batchScripts
applicationProject/batchScripts> mv training.dsc application.dsc
applicationProject/batchScripts> ls
application.dsc
```

3.5.2 Editing the Batch Script

DIAsDEM batch scripts are XML documents that conform to the XML document type definition listed in section 4.2 on page 92. They can be modified using any text editor

or preferrably using the dedicated editor (*Solutions → Batch Script Processing → Edit Batch Script*). Open the file `${APP_PROJECT_HOME}/batchScripts/application.dsc` in your preferred text editor to quickly correct the project directory in this script. Replace all occurrences of the training directory `${PROJECT_HOME}` such as `/home/kwinkler/diasdem/DIAsDEM.cases/tutorial/trainingProject`) with the directory that now corresponds to `${APP_PROJECT_HOME}` such as `/home/kwinkler/diasdem/DIAsDEM.cases/tutorial/applicationProject`. Do not include the trailing slash in directory names to ensure the proper replacement of all 36 `${PROJECT_HOME}` occurrences in the script.

Thereafter, open the DIAsDEM Batch Script Editor by selecting *Solutions → Batch Script Processing → Edit Batch Script*, clicking the *Open* button and choosing the application script `application.dsc` located in the directory `${APP_PROJECT_HOME}/batchScripts`. Figure 3.29 illustrates the initially shown *1. Settings* tab of the editor. After clicking on the *3. Tasks* tab, the sequence of 19 recorded tasks appears as depicted in Figure 3.30. Select task 11 ("Monitor Cluster Quality") by clicking once on the respective table row. Delete the respective task from the batch script by clicking the *Delete* button, because monitoring cluster quality is not necessary in the application phase. Afterwards, task 14 ("Monitor Cluster Quality") must also be deleted. For the same reason, delete the task "Derive Preliminary DTD" as well.

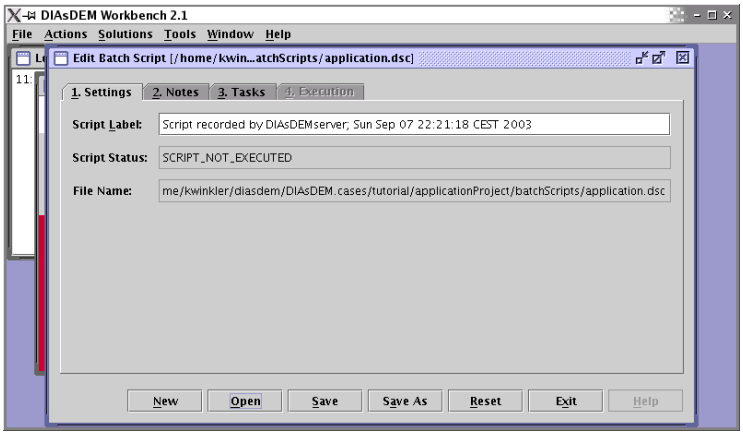

Figure 3.29: *1. Settings* tab of *Edit Batch Script* window

The application script should now comprise the following sequence of 16 tasks: "New Project", "Create Document Collection", "Import Plain Text Files", "Create Text Units", "Tokenize Text Units", "Replace Named Entities 2.1", "Lemmatize Text Units", "Compute Term Frequency Statistics", "Vectorize Text Units", "Cluster Text Unit Vectors (Weka)", "Tag Text Units", "Lemmatize Text Units", "Compute Term Frequency Statistics", "Vectorize Text Units", "Cluster Text Unit Vectors (Weka)", "Tag Text Units",

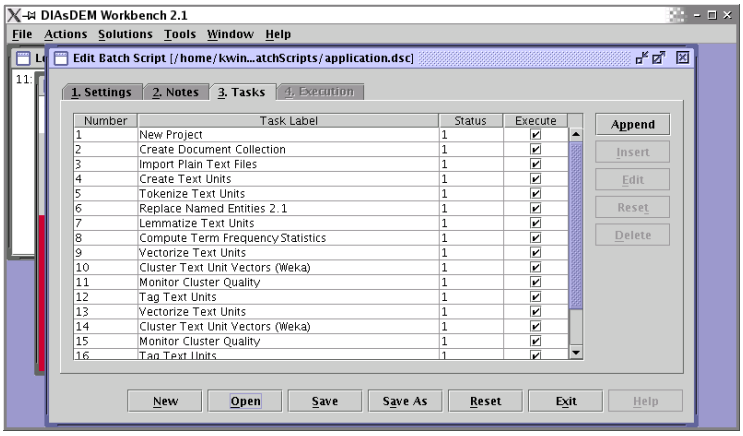

Figure 3.30: *3. Tasks* tab of *Edit Batch Script* window

"Tag Documents", and finally "Close Project". A few tasks have to be edited by hand to adjust their parameters settings. To proceed, select the row corresponding to task 1 ("New Project") and click on *Edit*. Figures 3.31 and 3.32 illustrate the appearing task dialog, which allows you to modify the parameter settings. For example, input the *Project Name* "Tutorial - Application Phase" and click on *OK* to commit the change.

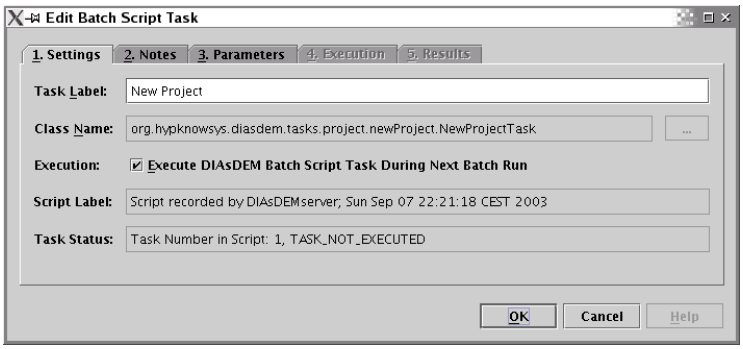

Figure 3.31: *1. Settings* tab of *Edit Batch Script Task* dialog

In addition, the following five tasks have to edited as well: In the "Import Plain Text Files" task, modify the *File Name Extension* of files to be imported from ".training.txt" to ".application.txt". Ignore the appearing warning message stating that the specified *Collection File* does not exist. During script execution, this file is created in the preceding task "Create Document Collection". In the "Vectorize Text Units" task of both iterations, change *Descriptor Weights* from "Create Descriptor Weights

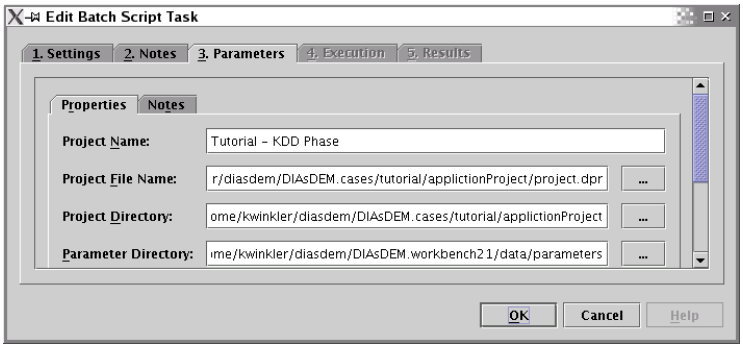

Figure 3.32: *3. Parameters* tab of *Edit Batch Script Task* dialog

File: IDF Weights" to "Apply Existing Descriptor Weights File". In the "Cluster Text Unit Vectors (Weka)" task of both iterations, change *Clustering Mode* from "Clustering Phase (Create New Clustering Model" to "Application Phase (Apply Existing Clustering Model)". After saving the script by clicking on *Save*, the modified batch script is now ready for execution, which is explained in the next section. Finally, click the *Exit* button to close the DIAsDEM Batch Script Editor.

3.5.3 Executing the Batch Script

There are two options for executing batch scripts, because DIAsDEM Workbench comprises both a GUI-based and a command line batch script processor. Firstly, open the GUI-based script processor by selecting *Solutions* → *Batch Script Processing* → *Execute Batch Script*. Open the batch script `application.dsc` located in the directory `${APP_PROJECT_HOME}/batchScripts` and click on *OK* to start the script execution. Figure 3.33 illustrates DIAsDEM Workbench while executing this script. Secondly, the batch script `application.dsc` can be executed by the command line script processor as indicated in the installation notes:

```
batchScripts> pwd
/home/kwinkler/diasdem/DIAsDEM.cases/tutorial/applicationProject/batchScripts
batchScripts> ../../../DIAsDEM.workbench21/bin/diasdembatch application.dsc verbose
*
* DIAsDEM Workbench 2.1 2.1.0.0 for Java 1.4.2 Released 15 Aug 2003
* Executing Batch Script: application.dsc ...
*
15:39:04 Task started: Execute Batch Script
15:39:04 Starting execution of task 1/16 (New Project)
15:39:04 Execution of task 1/16 (New Project) has terminated successfully
15:39:04 Starting execution of task 2/16 (Create Document Collection)
```

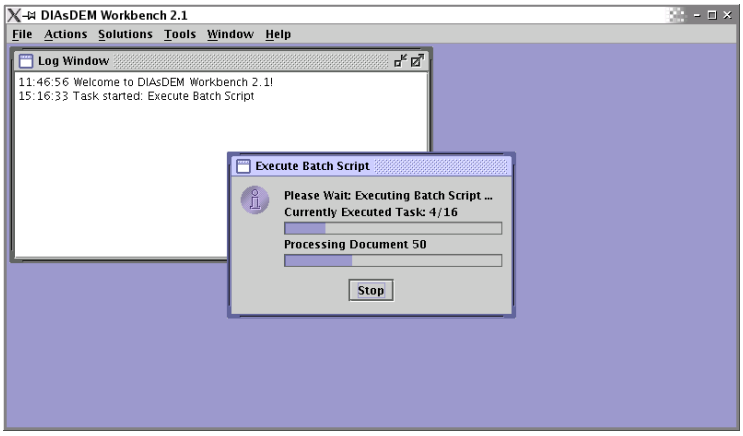

Figure 3.33: DIAsDEM Workbench 2.1 during GUI-based batch script execution

```
15:39:04 Execution of task 2/16 (Create Document Collection) has terminated successfully
...
15:41:25 Execution of task 15/16 (Tag Documents) has terminated successfully
15:41:25 Starting execution of task 16/16 (Close Project)
15:41:25 Execution of task 16/16 (Close Project) has terminated successfully
15:41:25 Task successfully finished: Execute Batch Script
```

Semantically tagging 161 Commercial Register entries takes approx. two minutes in both cases. After script execution, altogether 161 semantically tagged XML documents are located in the directory `${APP_PROJECT_HOME}`. Using for example *Tools* → *Miscellaneous* → *XML Document Viewer*, open the output document `${APP_PROJECT_HOME}/outputXmlDocuments/part1/document1.xml`, which is partly depicted below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CommercialRegisterEntry SYSTEM "CommercialRegisterEntry.dtd">
<CommercialRegisterEntry>
  <MetaData>
    <Name>DiasdemDocumentID</Name>
    <Content>/home/.../applicationProject/inputCollection/volume100000.xml:0</Content>
  </MetaData>
  <MetaData>
    <Name>SourceFile</Name>
    <Content>/home/.../data/samples/de/case1/file11075.application.txt</Content>
  </MetaData>
  <TaggedDocument>
    Dachdeckerarbeiten. <ShareCapital AmountOfMoney="25000 EUR">Stammkapital:
    25.000 EUR.</ShareCapital><LimitedLiabilityCompany>Gesellschaft mit
    beschränkter Haftung.</LimitedLiabilityCompany> ...
```

```
    <SolePowerToRepresentCanBeGranted>Einzelvertretungsbefugnis kann erteilt
    werden.</SolePowerToRepresentCanBeGranted><AppointmentOfManagingDirector
    Person="10; Mario Schmeling; null; null; null; 05.07.1967; null; Bamme; null;
    null [AND] 11; Thomas Weber; null; null; null; 16.06.1967; null; Rathenow;
    null; null">Mario Schmeling, 05.07.1967, Bamme; und Thomas Weber, 16.06.1967,
    Rathenow, sind zu Geschäftsführern bestellt.</AppointmentOfManagingDirector> ...
    <PublicationMediaOfCommercialRegisterEntries>Nicht eingetragen: Die
    Bekanntmachungen der Gesellschaft erfolgen im Bundesanzeiger.
    </PublicationMediaOfCommercialRegisterEntries>
  </TaggedDocument>
</CommercialRegisterEntry>
```

Furthermore, open the script `${APP_PROJECT_HOME}/batchScripts/application.dsc` in *Solutions → Batch Script Processing → Edit Batch Script*. As Figures 3.34 and 3.35 illustrate, the entire script as and each performed task now contains additional information (e.g., log messages and time stamps) about the execution.

You have now reached the end of this introductory case study! We are looking forward to getting your feedback on the DIAsDEM framework and DIAsDEM Workbench.

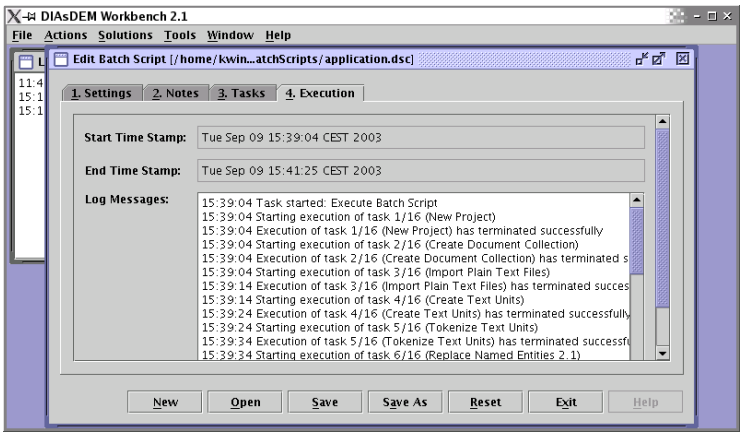

Figure 3.34: *4. Execution* tab of *Edit Batch Script* window

3.6 Auxiliary Tasks

3.6.1 Replacing Named Entities with NEEX 2.0

NEEX 2.0 is the predecessor of the current release NEEX 2.1, which should be preferably used for named entity extraction. Nevertheless, NEEX 2.0 is briefly described in this section for the sake of completeness. Based on lists and regular expressions, it is capable

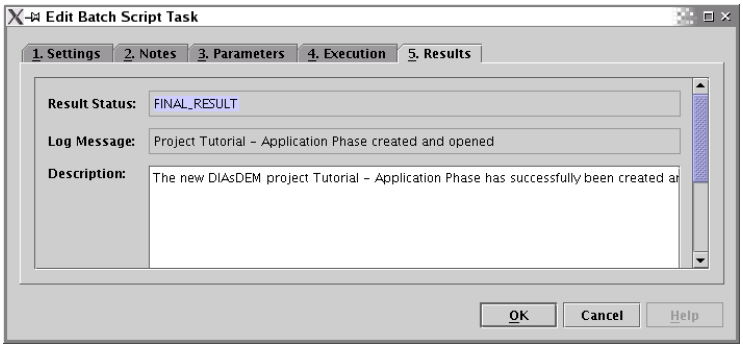

Figure 3.35: *3. Results* tab of *Edit Batch Script Task* dialog

of identifying named entities of the following types: "person", "company", "number", "date", "time", "amount_of_money", "paragraph", "email", "url", "organization_id", "document_id", "court", "postal_code", "isin", and "wkn". Similar to its successor, NEEX 2.0 can be fully parameterized by editing the corresponding parameter files. However, it employs a slightly different approach to named entity extraction:

1. Firstly, regular expressions listed in *Regex NE File* are matched against each text unit to identify instances of the basic named entities "number", "date", "time", "amount_of_money", "paragraph", "email", and "url". For example, "50000 DM" is an instance of named entity type "amount_of_money", which occurs in the tokenized and normalized sentence "Stammkapital : 50000 DM ."

2. Secondly, all tokens of each text unit are looked up in a dictionary to identify instances of the basic named entities "place", "organization_abbreviation", "forename", "surname", "title", and "middle_initial". Basically, this dictionary comprises the contents of *Places File*, *Organizations End File*, *Forenames File*, *Surnames File*, *Titles File*, and *Middle Initials File*. Additionally, *Surname Suffixes File* is used to identify rare surnames on the basis of frequently occurring surname suffixes such as "isz". Organizations are extracted by utilizing abbreviations of organizations listed in *Organizations End File*. Additionally, *Organizations Start File* contains terms and phrases that frequently precede names of organizations. All basic named entities are replaced by their respective placeholders in this phase.

3. Thirdly, rules contained in *Composite NE File* are applied to text units in order to identify instances of the composite named entities "person" and "company". Each composite named entity consists of certain basic named entities that occur in a context described by rules in *Composite NE File*. For instance, a "person" can be instantiated from basic named entities such as "forename" and "surname". If a

composite named entity is identified, both the textual contents and the named entity placeholders that are matched by the rule are substituted by the corresponding composite named entity placeholder.

4. Finally, NEEX applies heuristics to map various occurrences of identical persons and companies within one text document onto a canonical form. Thereby, all composite named entity placeholders, which probably reference the same real world entity, are replaced by a new placeholder referencing the canonical form of the corresponding named entities.

Replace Named Entities 2.0: Summary

Module: *Actions → Prepare Data Set → Replace Named Entities 2.0*

Use Case: The user must pre-process all imported texts as part of the DIAsDEM KDD process for semantic tagging of domain-specific texts archives. Identifying and replacing named entities is pre-processing phase 3 of 4.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`, whose elements `<ProcessedTextUnit>` must not contain previously inserted named entity references `<NeRef>`. Text units should have been created and tokenized in the collection.

Result: Elements of the section `<ProcessedTextUnits>` mark up text units containing placeholders for extracted named entities, which are stored in the elements `<NamedEntity>` of section `<NamedEntities>`. Additionally, the project properties *Default Collection File*, *Default Forenames File*, *Default Surnames File*, *Default Surname Suffixes File*, *Default Middle Initials File*, *Default Title File*, *Default Places File*, *Default Organizations Start File*, *Default Organizations End File*, *Default Composite NE File*, and *Default Regex NE File* are set and updated, respectively.

Replace Named Entities 2.0: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Forenames File: Valid local file name of existing file that contains a list of forenames in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Forenames File*

Surnames File: Valid local file name of existing file that contains a list of surnames in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Surnames File*

Surname Suffixes File: Valid local file name of existing file that contains a list of frequent surname suffixes in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Surname Suffixes File*

Middle Initials File: Valid local file name of existing file that contains a list of middle initials in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Middle Initials File*

Title File: Valid local file name of existing file that contains a list of academic and professional titles in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Titles File*

Places File: Valid local file name of existing file that contains a list of places (i.e., cities) in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Places File*

Organizations Start File: Valid local file name of existing file that contains terms that are followed by an organization ending with a known abbreviation in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Organizations Start File*

Organizations End File: Valid local file name of existing file that contains a list of organizational abbreviations in the format described in section 4.3.4 on page 102; file extension: `.txt`; default value: project property *Default Organizations End File*

Composite NE File: Valid local file name of existing file that contains rules for instantiating composite named entities (i.e., persons and companies) in the format described in section 4.3.4 on page 102; file extension: `.csv`; default value: project property *Default Composite NE File*;

Regex NE File: Valid local file name of existing file that contains regular expressions for the identification of basic named entities in the format described in section 4.3.4 on page 102; file extension: `.csv`; default value: project property *Default Regex NE File*

3.6.2 Removing Stopwords

DIAsDEM Workbench is capable of removing meaningless stopwords from processed text units. As explained in section 1, the DIAsDEM framework proposes utilizing a controlled vocabulary (i.e., a domain-specific thesaurus) for dimension reduction. Thus, stopword removal can be skipped in this case study due to the existence of a domain-specific thesaurus. However, stopswords should be removed in case of establishing a new controlled

vocabulary for a different domain. The text file `${PARAMETER_HOME}/removeStopwords/de/StopwordsDE.txt` contains a default German stopword list, which should be modified according to domain-specific needs.

Remove Stopwords: Summary

Module: *Actions → Prepare Data Set → Remove Stopwords*

Use Case: The user wants to remove meaningless stopwords from text units that are contained in the section `<ProcessedTextUnits>`.

Prerequisites: The default text units layer of each DIAsDEM document must contain the section `<ProcessedTextUnits>`. Text units should have been created and tokenized in the DIAsDEM collection.

Result: Elements `<ProcessedTextUnit>` do not contain terms listed in *Stopwords File*. Additionally, the project properties *Default Collection File* and *Default Stopwords File* are set and updated, respectively.

Remove Stopwords: Parameters

Collection File: Valid local file name of existing collection file; file extension: `.dcf`; default value: project property *Default Collection File*

Stopwords File: Valid local file name of existing file that contains stopwords in the format described in section 4.3.5 on page 107; file extension: `.txt`; default value: project property *Default Stopwords File*

3.6.3 Establishing an Initial Thesaurus

Before applying DIAsDEM Workbench to a new text collection, an application-specific thesaurus has to be created. This module establishes an initial thesaurus on the basis of term frequency statistics output by *Actions → Understand Domain → Compute Term Frequency Statistics*. Terms are inserted into the new thesaurus as descriptor terms, if their collection-specific absolute frequency is greater than or equal to a minimum and less than or equal to a maximum threshold. However, the resulting initial thesaurus should be manually refined by removing for example semantically unimportant terms. Moreover, the semantics of terms and concepts should be taken into account by defining relations between less important non-descriptors and associated descriptors of importance.

Note, it is strongly recommended to remove stopwords before establishing an initial thesaurus for a new application domain. If semantically less important stopwords are not removed, the resulting thesaurus contains them as well. After establishing an initial thesaurus, it should be refined by hand using *Tools → Thesaurus Editor*. In order to

quickly remove less important terms, thesaurus files might also be edited using any common text editor: Simply delete all lines that correspond to terms to be removed.

Establish Initial Thesaurus: Summary

Module: *Actions* → *Understand Domain* → *Establish Initial Thesaurus*

Use Case: The user wants to establish an application-specific initial thesaurus, which is based on collection-specific term frequencies. Subsequently, this thesaurus should be refined using *Tools* → *Thesaurus Editor*.

Prerequisites: Using *Actions* → *Understand Domain* → *Compute Term Frequency Statistics*, a collection-specific term frequency file must have been created. Text units should have been created, tokenized, and lemmatized in the DIAsDEM collection and named entities should have been replaced with placeholders in all text units.

Result: Descriptors in *Initial Thesaurus File* correspond to terms in *TF Statistics File*, if their term frequency is greater than or equal to *Min. Term Frequency* and less than or equal to *Max. Term Frequency*. The remaining terms are not inserted into *Initial Thesaurus File*.

Establish Initial Thesaurus: Parameters

TF Statistics File: Valid local file name of existing file; file extension: `.dws`; default value: project property *Default Word Statistics File*

Initial Thesaurus File: Valid local file name of file to be created or replaced by DIAsDEM Workbench; file extension: `.dth`

Min. Term Frequency: Non-negative integer; minimum absolute term frequency of descriptors in *Initial Thesaurus File*

Max. Term Frequency: Non-negative integer; maximum absolute term frequency of descriptors in *Initial Thesaurus File*

4 Technical Specification

4.1 DIAsDEM Documents

The default implementation of DIAsDEM Workbench stores DIAsDEM documents as part of so-called DIAsDEM volumes. The latter are XML files that conform to the following XML document type definition `DefaultDIAsDEMvolume.dtd`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT DefaultDIAsDEMvolume (DefaultDIAsDEMdocument*)>
<!ATTLIST DefaultDIAsDEMvolume NumberOfDocuments CDATA #IMPLIED>

<!ELEMENT DefaultDIAsDEMdocument (MetaData*, OriginalText, TextUnitsLayer*)>
<!ATTLIST DefaultDIAsDEMdocument NumberOfTextUnitsLayers CDATA #IMPLIED>

<!ELEMENT MetaData (Name, Content)>
<!ELEMENT Name (#PCDATA)> <!ELEMENT Content (#PCDATA)>
<!ELEMENT OriginalText (#PCDATA)>

<!ELEMENT TextUnitsLayer (MetaData*, OriginalTextUnits, ProcessedTextUnits?,
  RollbackTextUnits*, NamedEntities? )>
<!ATTLIST TextUnitsLayer TextUnitsLayerID CDATA #IMPLIED
  TextUnitsDescription CDATA #IMPLIED>

<!ELEMENT OriginalTextUnits (OriginalTextUnit+)>
<!ELEMENT OriginalTextUnit (#PCDATA)>
<!ATTLIST OriginalTextUnit TextUnitID CDATA #IMPLIED BeginIndex CDATA #IMPLIED
  EndIndex CDATA #IMPLIED>
<!ELEMENT ProcessedTextUnits (ProcessedTextUnit+)>
<!ELEMENT ProcessedTextUnit (#PCDATA | NeRef)*>
<!ATTLIST ProcessedTextUnit TextUnitID CDATA #IMPLIED Iteration CDATA #IMPLIED
  ClusterID CDATA #IMPLIED ClusterLabel CDATA #IMPLIED>
<!ELEMENT RollbackTextUnits (ProcessedTextUnit+)>
<!ELEMENT NamedEntities (NamedEntity+)>
<!ELEMENT NamedEntity (#PCDATA)>
<!ATTLIST NamedEntity NeID CDATA #IMPLIED NeType CDATA #IMPLIED>
<!ATTLIST RollbackTextUnits RollbackID CDATA #IMPLIED>
<!ELEMENT RollbackTextUnit (#PCDATA | NeRef)*>

<!ELEMENT NeRef EMPTY>
<!ATTLIST NeRef NeID CDATA #IMPLIED>
```

4.2 DIAsDEM Batch Scripts

DIAsDEM Workbench is capable of executing batch scripts (i.e., XML documents) that conform to the following XML document type definition `DiasdemBatchScript.dtd`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT DIAsDEMscript (Label, DIAsDEMscriptTask*, Notes?, Log?, Status?,
  StartTimeStamp?, EndTimeStamp?)>

<!ELEMENT DIAsDEMscriptTask (Label, Parameter, Result?, Notes?, Log?, Status?,
  StartTimeStamp?, EndTimeStamp?)>
<!ATTLIST DIAsDEMscriptTask ClassName CDATA #IMPLIED Execute CDATA #IMPLIED>

<!ELEMENT Label (#PCDATA)>
<!ELEMENT ClassName (#PCDATA)>
<!ELEMENT Parameter (ParameterAttributes)>
<!ATTLIST Parameter ClassName CDATA #IMPLIED>
<!ELEMENT Result (ResultAttributes)>
<!ATTLIST Result ClassName CDATA #IMPLIED>
<!ELEMENT Notes (#PCDATA)>
<!ELEMENT Log (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
<!ELEMENT StartTimeStamp (#PCDATA)>
<!ELEMENT EndTimeStamp (#PCDATA)>

<!ELEMENT ParameterAttributes (ParameterAttribute*)>
<!ELEMENT ParameterAttribute (AttributeName, AttributeValue)>
<!ELEMENT ResultAttributes (ResultAttribute*)>
<!ELEMENT ResultAttribute (AttributeName, AttributeValue)>

<!ELEMENT AttributeName (#PCDATA)>
<!ELEMENT AttributeValue (#PCDATA)>
```

4.3 Text Pre-Processing

4.3.1 Module: Create Text Units

Abbreviations File: Valid local file name of existing text file, which contains known abbreviations in the following format: Each line of *Abbreviations File* contains exactly one abbreviation whose capitalization is relevant. However, this module only matches abbreviations, if they either occur at the beginning of the text or if they follow one of certain special characters (i.e., the blank space and `()`,`;`:`/`-`'`"). Comment lines starting with "#" are ignored and can hence be used to structure the file. Example:

```
# Format: One case-sensitive, single- or multi-token abbreviation per line
Ph.D.
SAT.1
E.ON
a.d.
a. d.
a.D.
```

Full Stop Regex File: Valid local file name of existing text file, which contains regular expressions in the following format: Each line of *Full Stop Regex File* contains a regular expression matching full stops, exactly one tab stop and thereafter a corresponding replacement string that substitutes matched full stops with asterisks. Therefore, the replacement string usually includes references such as `$1` to captured subsequences like `(ges|Ges)` of the corresponding regular expression. Both the regular expression and the replacement string must be Java-compliant constructs as specified in the API documentation of the Java package `java.util.regex`. Before these regular expressions are matched against the text, full stops in abbreviations listed in *Abbreviations File* have been replaced by asterisks. Comment lines starting with "#" are ignored. Example:

```
# full stops in dates
([0-9]{1,2})\.([\ ]*[0-9]{1,2})\.([\ ]*[0-9]{2,4}) $1\*$2\*$3
([0-9]{1,2})\.([\ ]*[0-9]{1,2})\.([\ ]*)   $1\*$2\*$3
# full stops in abbreviations not preceded by a letter
(str|Str|pl|Pl)\.([\ ]*\d*)  $1\*$2
(ges|Ges)\.([\ ]*mbH) $1\*$2
# full stops in generic abbreviations such as titles (e.g., 'Dipl.-Kfm.')
(Dipl)\*([\ ]*|[\ ]*\-[\ ]*)([a-zöäüßA-ZÖÄÜ]+)\.  $1\*$2$3\*
(Art)\.\ (\d) $1\*\ $2
# full stops in numbers (longest numbers have to matched first)
(\s*[0-9]+)\.(\s*[0-9]+)     $1\*$2
(\s*[0-9]{1,3})\.(\s) $1\*$2
```

Default files of both *Abbreviations File* and *Full Stop Regex File* are provided in the language-specific subdirectories of `${PARAMETER_HOME}/createTextUnits`.

4.3.2 Module: Tokenize Text Units

Tokenize Regex File: Valid local file name of existing text file, which contains regular expressions in the following format: Each line of *Tokenize Regex File* contains a regular expression matching characters to be separated from letters, exactly one tab stop and thereafter a corresponding replacement string that separates tokens. Therefore, the replacement string usually includes references such as `$1` to captured subsequences of the corresponding regular expression. Both the regular expression and the replacement string

must be Java-compliant constructs as specified in the API documentation of the Java package `java.util.regex`. Comment lines starting with "#" are ignored. Example:

```
# Format: searchRegex<TAB>replaceString
(\S)(\.|\!|\?|\(|\)|\{|\}|\[|\]|\-|"|'|‘|’|:|;|,|\+|\/|\\) $1\ $2
(\.|\!|\?|\(|\)|\{|\}|\[|\]|\-|"|'|‘|’|:|;|,|\+|\/|\\)(\S) $1\ $2
```

Normalize Regex File: Valid local file name of existing text file, which contains regular expressions in the same format as *Tokenize Regex File* described above. Example:

```
# Format: searchRegex<TAB>replaceString
# dates
([0-9]{1,2})\.[\ ]*(Januar|Jan[\.]?)[\ ]*([\d]{2,4}) $1.01.$3
([\ ][0-9]{1,2})\.[\ ]*([0-9]{1})\.[\ ]*([\d]{2,4}) $1.0$2.$3
# numbers
(\s[0-9]{1,3})\.([0-9]{3,3})\.([0-9]{3,3}\s) $1$2$3
(\s[0-9]{1,3})\.([0-9]{3,3}\s) $1$2
# amounts of money
([\-]?\s*{0,1}\d{1,}[,.\d]{1,})\s(DM|DEM|Deutsche\s+Mark|D\s+\-\s+Mark) $1\ DEM
([\-]?\s*{0,1}\d{1,}[,.\d]{1,})\s(Euro|EUR[O]?|Euros) $1\ EUR
```

Multi Token Words File: Valid local file name of existing text file, which contains known multi-token terms in the following format: Each line of *Multi Token Words File* contains exactly one known multi-token word whose capitalization is relevant. Multi-token terms consist of multiple single tokens and blank spaces. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive multi-token term per line
Gesellschaft mit beschränkter Haftung
mit beschränkter Haftung
Offene Handelsgesellschaft
offene Handelsgesellschaft
```

The language-specific subdirectories of `${PARAMETER_HOME}/tokenizeTextUnits` contain defaults for *Tokenize Regex File*, *Normalize Regex File*, and *Multi Token Words File*.

4.3.3 Module: Replace Named Entities 2.1

Regex NE File: Valid local file name of existing file, which contains regular expressions for instantiating basic named entities (i.e., dates, amounts of money, URLs, and e-mail addresses). Each line contains a java.util.regex.Pattern regular expression that matches sequences of tokens as well as the corresponding name of the basic named entity separated

by exactly one tab stop. The following basic named entities could be instantiated using regular expressions: 'number', 'date', 'time', 'amount_of_money', 'paragraph', 'email', 'url', 'organization_id', 'document_id', 'court', 'postal_code', 'isin', and 'wkn'. Comment lines starting with "#" are ignored. Example:

```
# Format: searchRegex<TAB>namedEntityType
# normalized amounts of money
\d{1,}[,\.\d]{1,}\s(DEM|EUR|ATS)    amount_of_money
# normalized dates
\d{1,2}\.\d{1,2}\.\d{2,4}    date
```

Organization Indicators File: Valid local file name of existing file, which contains terms and term groups which frequently precede names of organizations. Each line contains one indicator term (group), which are processed case-sensitively. Note, groups of indicator terms such as 'Gesellschafterin:' must be entered tokenized and in reverse order (e.g., ': Gesellschafterin'), because NEEX 2.1 employs a backward search algorithm. All organization indicators containing full stops must be listed in *Abbreviations File* to ensure correct sentence splits. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, indicator term or reversed term group per line
: Gesellschafterin
Gesellschafterin
: Gesellschafter
Gesellschafter
Mitgesellschafter
```

Organization Suffixes File: Valid local file name of existing file, which contains a list of frequent organization suffixes in the following format: Each line contains exactly one suffix whose capitalization is relevant. NEEX 2.1 can process both single- (e.g., 'KG') and multi-token suffixes containing for example blank spaces (e.g., 'GmbH & Co. KG'). Note, the term 'mit_beschränkter_Haftung' is a multi-token term whose real counterpart 'mit beschränkter Haftung' is listed in *Multi Token Words File* All suffixes containing full stops such as 'e.Kfr.' must be listed in *Abbreviations File* to ensure correct sentence splits. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token
# organization suffix per line
GmbH & Co. KG
KG
Gesellschaft_mit_beschränkter_Haftung
mit_beschränkter_Haftung
AG
```

Organization Affixes File: Valid local file name of existing file, which contains a list of terms that frequently follow organization suffixes such as 'GmbH' or 'AG' in the following format: Each line contains exactly one affix whose capitalization is irrelevant. NEEX 2.1 can process both single- (e.g., 'Import') and multi-token organization affixes containing for example blank spaces (e.g., 'Import und Export'). Organization affixes containing special characters such as '(Deutschland)' or 'Wach- und Werkschutz' must be entered in their tokenized form '(Deutschland)' or 'Wach - und Werkschutz'. To that end, *Actions* → *Miscellaneous* → *Tokenize Parameter Text File* might be employed. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token affix per line
Import
Export
Import und Export
( Deutschland )
Wach - und Werkschutz
```

Organizations File: Valid local file name of existing file, which contains complete names of large organizations NEEX 2.1 can process both single- (e.g., 'Adidas') and multi-token organizations containing for example blank spaces (e.g., 'Adidas - Salomon'). They are processed case-sensitively. All organizations containing full stops must be listed in *Abbreviations File* to ensure correct sentence splits. Organizations containing special characters such as 'Adidas-Salomon' or 'E.ON' must be entered in their tokenized form 'Adidas - Salomon' or 'E.ON'. Note, 'E.ON' is a abbreviation listed in *Abbreviations File*. Include organizations in this file, if their occurrences without known organization suffixes have to be extracted or if they include terms listed in the file containing organization indicators.

```
# Format: One case-sensitive, single- or tokenized multi-token organization per line
Adidas
Adidas - Salomon
Allianz
Altana
BASF
```

Place Indicators File: Valid local file name of existing file, which contains terms and term groups which frequently precede places to be extracted as named entities. Each line contains one indicator term (group), which are not processed case-sensitively. Note, groups of indicator terms such as 'mit Niederlassung in' must be entered tokenized and in reverse order (e.g., 'in Niederlassung in'), because NEEX 2.1 employs a backward search algorithm. All place indicators containing full stops must be listed in *Abbreviations File* to ensure correct sentence splits. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, indicator term or reversed term group per line
,
:
in
und
Sitz
```

Places File: Valid local file name of existing file, which contains a list of frequently occurring places, which are not processed case-sensitively. NEEX 2.1 can process both single- (e.g., 'Neudörfer') and multi- token places containing for example blank spaces (e.g., 'Baden Baden'). Places containing special characters such as 'Frankfurt am Main', 'Frankfurt (Main)', 'Frankfurt/Oder' or 'Halle/Westf.' must be entered in their tokenized form 'Frankfurt am Main', 'Frankfurt (Main)', 'Frankfurt / Oder' or 'Halle / Westf.' as 'Westf.' is contained in the list of known abbreviations. To that end, *Actions → Miscellaneous → Tokenize Parameter Text File*. Place affixes such as names of rivers, districts or countries should be entered separately in *Place Affixes File*. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token organization per line
Berlin
Hamburg
Frankfurt am Main
Frankfurt / Oder
Halle / Westf.
```

Place Affixes File: Valid local file name of existing file, which contains frequently occurring place affixes such as names of rivers and districts. NEEX 2.1 can process both single- (e.g., 'Main') and multi-token place affixes containing for example blank spaces (e.g., '/ Main'). Place affixes are processed case-sensitively. Place affixes containing special characters such as '/Main' or '(Main)' must be entered in their tokenized form '/ Main' or '(Main)'. To that end, *Actions → Miscellaneous → Tokenize Parameter Text File*. Each place affix can either be a weak or a strong place affix, whereas places can not end with a weak affix. Each line contains the affix type (either 'weak_place_affix' or 'strong_place_affix') and the affix itself separated by exactly one tab stop. Comment lines starting with "#" are ignored. Example:

```
# Format: placeAffixType<TAB>single- or tokenized multi-token place affix
weak_place_affix      an der
weak_place_affix      a. d.
weak_place_affix      im
strong_place_affix    Main
strong_place_affix    ( Main )
```

```
strong_place_affix    / Main
strong_place_affix    / M.
```

Person Name Indicators File: Valid local file name of existing file, which contains terms and term groups which frequently precede person names. Person name indicators are processed case- sensitively. Groups of indicator terms such as 'Gesellschafter:' must be entered tokenized and in reverse order (e.g., ': Gesellschafter'), because NEEX 2.1 employs a backward search algorithm. All person name indicators containing full stops must be listed in *Abbreviations File* to ensure correct sentence splits. Each line contains the type literal as well as the indicator term (group) separated by exactly one tab stop. Valid person name indicator types are 'weak_pos_person_indicator', 'strong_pos_person_indicator' and 'strong_neg_person_indicator'. Note, person name indicators of the type 'weak_pos_person_indicator' are not used in NEEX 2.1. The occurrence of negative indicators is checked before and after any person name candidate. Hence they should comprise of one token only. Comment lines starting with "#" are ignored. Example:

```
# Format: placeAffixType<TAB>single- or tokenized, reversed multi-token indicator term
strong_pos_person_indicator  Herr
strong_pos_person_indicator  Mr.
strong_pos_person_indicator  geb.
strong_neg_person_indicator  Firma
strong_neg_person_indicator  Straße
strong_neg_person_indicator  Flughafen
```

Titles File: Valid local file name of existing file, which contains frequent academic and professional titles. NEEX 2.1 can process single- (e.g., 'Prof.') and multi-token titles containing for example blank spaces (e.g., 'Prof. Dr.'). Titles are processed case-sensitively. Title containing special characters such as 'Prof. Dr.' or 'Dipl.-Ingenieurin' or 'Dipl.-Kfm. (FH)' must be entered in their tokenized form 'Prof. Dr.', 'Dipl.-Ingenieurin' or 'Dipl.-Kfm. (FH)'. To that end, *Actions → Miscellaneous → Tokenize Parameter Text File* might be employed. All titles containing full stops such as must either be listed in *Abbreviations File* or be matched by a regular expression in *Full Stop Regex File* (e.g., 'Dipl.-Ingenieurin'). Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token title per line
Prof. Dr.
Dipl.-Ingenieurin
Dipl.-Kfm. ( FH )
Prof.
Dr.
```

Forenames File: Valid local file name of existing file, which contains a list of forenames in the following format: Each line contains exactly one forename whose capitalization is relevant. NEEX 2.1 can process both single- (e.g., 'Stanka') and multi-token forenames containing for example blank spaces (e.g., 'Stanka Cevdet'). Do not include multi-token forenames consisting of multiple forenames such as 'Hans-Joachim' or 'Hans Joachim', because they are extracted automatically. However, forenames containing special characters such as 'Hans-Joachim' must be entered in their tokenized form 'Hans - Joachim'. To that end, *Actions* → *Miscellaneous* → *Tokenize Parameter Text File* might be employed. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token forename per line
Stanka
Cevdet
Wolfgang
Vid
Joaquin
```

Middle Initials File: Valid local file name of existing file, which contains a list of middle initials in the following format: Each line contains exactly one middle initial whose capitalization is relevant. NEEX 2.1 can process both single- (e.g., 'A.' or 'von') and multi-token middle initials containing for example blank spaces (e.g., 'de la'). Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token middle initial per line
de la
A.
von
De
de
```

Surnames File: Valid local file name of existing file, which contains frequent surnames. NEEX 2.1 can process both single- (e.g., 'Schöppe') and multi-token surnames containing for example blank spaces (e.g., 'Schöppe Rocher'). Surnames are processed case-sensitively. Do not include multi-token surnames consisting of multiple surnames such as 'Schöppe-Rocher' or 'Schöppe Rocher', because they are extracted automatically. However, surnames containing special characters such as 'Schöppe-Rocher' must be entered in their tokenized form 'Schöppe - Rocher'. To that end, *Actions* → *Miscellaneous* → *Tokenize Parameter Text File* might be employed. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token surname per line
Meier
Müller
Schulze
Schmidt
Schmitt
```

Surname Suffixes File: Valid local file name of existing file, which contains frequently occurring suffixes of surnames. Capitalized tokens following a forename or an academic title are assumed to be a surname, if they end with a suffix listed in this file. Each line contains exactly one surname suffix whose capitalization is relevant. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token surname suffix per line
aci
ack
ad
wsky
yer
```

Name Affixes File: Valid local file name of existing file, which contains frequently occurring name affixes. NEEX 2.1 can process both single- (e.g., 'jun.') and multi-token name affixes containing for example blank spaces (e.g., ', jun.'). Each line contains exactly one name affix whose capitalization is relevant. Name affixes containing special characters such as ',jun.' or 'Ph.D.' must be entered in their tokenized form ', jun.' or 'Ph.D.' as both 'jun.' and 'Ph.D.' are contained in the list of known German abbreviations. To that end, *Actions* → *Miscellaneous* → *Tokenize Parameter Text File* might be employed. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token name affix per line
Ph.D.
, Ph.D.
sen.
, sen.
jun.
```

Professions File: Valid local file name of existing file, which contains frequently occurring professions. NEEX 2.1 can process both single- (e.g., 'Angestellter') and multi-token terms containing for example blank spaces (e.g., 'Kaufmännischer Angestellter'). Professions processed case-sensitively. Professions containing special characters such as 'Dipl.-Kaufmann' or 'Kfz.-Schlosser' must be entered in their tokenized form 'Dipl. -

Kaufmann' or 'Kfz. - Schlosser'. To that end, *Actions → Miscellaneous → Tokenize Parameter Text File*. All professions containing full stops such as must either be listed in *Abbreviations File* or be matched by a regular expression in *Full Stop Regex File*. For obvious reasons, do not include text unit descriptors such as the term 'Geschäftsführer' in this file. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single- or tokenized multi-token profession per line
Angestellter
Angestellte
Dipl. - Kaufmann
Dipl. - Kauffrau
```

Composite NE File: Valid local file name of existing file, which contains rules for instantiating composite named entities (i.e., persons, companies, and company relocations) from previously identified basic named entities such as person names, places, or dates. Each line contains one DIAsDEM-specific rule, which matches sequences of tokens and basic named entities, as well as the corresponding composite named entity constructor separated by exactly one tab stop.

The DIAsDEM-specific rule is a simple regular expression, which must be matched by a tokenized text unit in order to instantiate a composite named entity such as "person" or "company". This expression can include case-sensitive tokens (e.g., "mit", "Sitz") and generic placeholders for basic named entities (e.g., "<<organization>>", "<<person_name>>") as defined in `org.hypknowsys.diasdem.core.neex.NamedEntity`. The corresponding named entity constructor instantiates a composite NE of type "person", "company", or "company_relocation". Each constructor references terms and generic placeholders of the corresponding regular expression, which are attribute values of the new composite named entity. Note, tokens in both expressions must be separated from each others by blank spaces, because named entities are identified in tokenized text units. Comment lines starting with "#" are ignored.

```
# Format: DIAsDEM-specific rule<TAB>DIAsDEM-specific composite NE constructor
# Note, there are three DIAsDEM-specific composite NE constructors available:
# company( Name , Place , DistrictCourt , CommercialRegisterID )
# person( Name , Surname , Forename , Title , MiddleInitial , DoB ,
#   MothersName , Place , Occupation )
# company_relocation( Name , OriginPlace , OriginDistrictCourt ,
#   OriginCommercialRegisterID , DestinationPlace , DestinationDistrictCourt ,
#   DestinationCommercialRegisterID )
<<organization>>      company( 0 , null , null , null )
<<organization>> <<place>>   company( 0 " " 1 , 1 , null , null )
<<organization>> in <<place>> ( <<organization_id>> ) company( 0 , 2 , null , 4 )
<<person_name>> person( 0 , null , null , null , null , null , null , null , null )
```

The language-specific subdirectories of `${PARAMETER_HOME}/replaceNamedEntities` contain defaults for all parameter files described above. However, NEEX 2.1 parameter files are provided in the subdirectory `neex21` only.

4.3.4 Module: Replace Named Entities 2.0

Forenames File: Valid local file name of existing file, which contains a list of forenames in the following format: Each line contains exactly one forename whose capitalization is relevant. In NEEX 2.0, forenames must not be multi-token terms that include blank spaces. However, forenames consisting of multiple tokens separated by either a blank space (e.g., "Hans Joachim") or a hyphen (e.g., "Hans-Joachim") can be identified by user-specific rules in *Composite NE File*. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token forename per line
Stanka
Cevdet
Wolfgang
Vid
Joaquin
```

Surnames File: Valid local file name of existing file, which contains frequent surnames. In NEEX 2.0, surnames must not be multi-token terms that include blank spaces. However, surnames consisting of multiple tokens separated by a hyphen (e.g., "Müller-Schmidt") can be identified by user-specific rules in *Composite NE File*. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token surname per line
Meier
Müller
Schulze
Schmidt
Schmitt
```

Surname Suffixes File: Valid local file name of existing file, which contains frequently occurring suffixes of surnames. Only tokens that are preceded by an instance of named entity types "forename" or "middle_initial" are checked against the list of surname suffixes. Each line contains exactly one surname suffix whose capitalization is relevant. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token surname suffix per line
aci
ack
ad
wsky
yer
```

Middle Initials File: Valid local file name of existing file, which contains a list of middle initials in the following format: Each line contains exactly one middle initial whose capitalization is relevant. Only tokens that are preceded by an instance of named entity type "forename" are checked against the list of middle initials. NEEX 2.0 can only process single-token middle initials (e.g., 'A.' or 'von'). Hence, middle initials must not include blank spaces. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token middle initial per line
A.
B.
von
De
de
```

Titles File: Valid local file name of existing file, which contains a list of academic and professional titles in the following format: Each line contains exactly one title whose capitalization is relevant. NEEX 2.0 can only process single-token titles which do not contain blank spaces. Note, the first title 'Prof._Dr.' is a multi-token term whose real counterpart 'Prof. Dr.' is listed in *Multi Token Words File*. All titles containing full stops such as must either be listed in *Abbreviations File* (e.g., 'Dr.') or be matched by a regular expression in *Full Stop Regex File* (e.g., 'Dipl.-Ingenieurin'). Terms including full stops, dashes and other special characters must be normalized (i.e., de-tokenized) by an appropriate entry in *Normalize Regex File*. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token title per line
Prof._Dr.
Dipl.-Ingenieurin
Prof.
Dr.
Steuerberaterin
```

Places File: Valid local file name of existing file, which contains a list of places (i.e., cities) in the following format: Each line contains exactly one place whose capitalization

is relevant. NEEX 2.0 can only process single-token places which do not contain blank spaces. The terms 'Frankfurt_am_Main' and 'Frankfurt_(_Main_)' are examples of multi-token terms whose real, but tokenized counterparts 'Frankfurt am Main' and 'Frankfurt (Main)' are listed in *Multi Token Words File*. Alternatively, multiple subsequent places can be merged by user-specific rules in *Composite NE File*. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token place per line
Frankfurt_am_Main
Frankfurt_(_Main_)
Neudörfel
Neualbenreuth
Oebles-Schlechtewitz
```

Organizations Start File: Valid local file name of existing file, which contains terms and term groups which frequently precede names of organizations. Organization indicators are processed case-sensitively. Note, groups of indicator terms such as 'Gesellschafterin:' must be entered tokenized and in reverse order (e.g., ': Gesellschafterin'), because NEEX 2.0 employs a backward search algorithm. All organization indicators containing full stops must be listed in *Abbreviations File* to ensure correct sentence splits. The syntax of this file must conform to `org.hypknowsys.misc.io.CsvFile`. Hence, do not delete the first two meta-data lines and quote all indicator terms. Comment lines starting with "#" are ignored. Example:

```
# Format: "case-sensitive, indicator term or reversed term group"
attribute1
String
# Example: 'Gesellschafterin : <organization>ABC GmbH</organization>'
": Gesellschafterin"
"der"
"die"
"gründenden zu"
":"
```

Organizations End File: Valid local file name of existing file, which contains frequent organization suffixes. NEEX 2.0 can only process single-token organization suffixes which do not contain blank spaces. However, NEEX aims at identifying complex abbreviations that consist of multiple elementary abbreviations listed in *Organizations End File*. Organization suffixes are processed case-sensitively. Note, the first term 'GmbH_&_Co._KG' is a multi-token term whose real counterpart 'GmbH & Co. KG' is listed in *Multi Token Words File*. All suffixes of organizations containing full stops such as 'e.Kfr.' must be listed in *Abbreviations File* to ensure correct sentence splits. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token title per line
GmbH_&_Co._KG
KG
e.K.
e.Kfm
KGaA
```

Composite NE File: Valid local file name of existing file, which contains rules for instantiating composite named entities as comma-separated values in a format defined by the Java class `org.hypknowsys.misc.io.CsvFile`. Except for the first two meta-data lines, each line contains exactly two string attribute values.

The first attribute value is a simple, DIAsDEM-specific regular expression, which must be matched by a tokenized text unit in order to instantiate a composite named entity, i.e., "person" or "company". This expression can include case-sensitive tokens (e.g., "mit", "Sitz") and generic placeholders for basic named entities (e.g., "<<organization>>", "<<surname>>") as defined in the Java class `org.hypknowsys.diasdem.core.neex.NamedEntity`. The second attribute value contains the corresponding named entity constructor that either creates a "person" or a "company". Each constructor references terms and generic placeholders of its corresponding regular expression that are attribute values of the new composite named entity (e.g., "person"). Note, tokens in both expressions must be separated from each others by blank spaces, because named entities are identified in tokenized text units. *Composite NE File* must not contain empty lines. Comment lines starting with "#" are ignored.

Note: Users only edit or create CSV files containing so-called initial composite named entity rules. Thereafter, *Actions → Miscellaneous → Extend Composite NE Rules 2.0* should be employed to extend these initial rules by applying a heuristic permutation algorithm. This module automatically extends user-supplied initial rules and thus creates the final *Composite NE File*. Extended composite named entity rules cater for the fact that many tokens can be instances of multiple basic named entities. Thus, additional rules are created by extending plain instances of the basic named entities "place", "forename", and "surname" with possible combinations of them. For example, the initial rule "<<organization>> mit Sitz in <<place>>" contains the generic basic named entity placeholder "<<place>>". Therefore, three additional extended rules are generated and added to *Composite NE File* that substitute "<<place>>" with placeholders for the following combinations of named entity types: "<<place forename>>", "<<place surname>>" and "<<place forename surname>>". This following CSV file contains user-defined, initial composite named entity rules:

```
# Format: "DIAsDEM-specific rule";"DIAsDEM-specific composite NE constructor"
# Note, there are two DIAsDEM-specific composite NE constructors available:
# company( Name , Place )
```

```
# person( Surname , Forename , Title , MiddleInitial , DoB , MothersName , Place )
Attribute0;Attribute1
String;String
"<<organization>>";"company( 0 , null )"
"<<organization>> mit Sitz in <<place>>";"company( 0 , 4 )"
"<<organization>> Sitz , <<place>> <<place>> ";"company( 0 , 3 4 )"
"<<organization>> Sitz , <<place>> / <<place>> ";"company( 0 , 3 \"/\" 5 )"
"<<forename>> <<surname>>";"person( 1 , 0 , null , null , null , null , null )"
"<<title>> <<surname>> - <<surname>>";"person( 1 \"-\" 3 , null , 0 , null , ... )"
```

Below, an excerpt of *Composite NE File* contains automatically extended composite named entity rules that correspond to the first two initial rules shown above:

```
String;String
"<<organization>>";"company( 0 , null )"
"<<organization>> mit Sitz in <<place>>";"company( 0 , 4 )"
"<<organization>> mit Sitz in <<place forename>>";"company( 0 , 4 )"
"<<organization>> mit Sitz in <<place surname>>";"company( 0 , 4 )"
"<<organization>> mit Sitz in <<place forename surname>>";"company( 0 , 4 )"
```

The DIAsDEM Workbench module *Actions* → *Miscellaneous* → *Extend Composite NE Rules 2.0* can be employed to automatically create *Composite NE File* after editing a CSV file containing domain-specific initial composite named entity rules. Note that the file corresponding to the parameter *Basic NE File* must not be edited. After modifying *Initial Composite NE File*, type in the following parameters to create *Extended Composite NE File* which is commonly referred to as *Composite NE File*:

Parameter	Value
Basic NE File	`${PARAMETER_HOME}/replaceNamedEntities/de/neex20/BasicNE_DoNotEdit.csv`
Initial Composite NE File	`${PARAMETER_HOME}/replaceNamedEntities/de/neex20/Case1234InitialCompositeNE.csv`
Extended Composite NE File	`${PARAMETER_HOME}/replaceNamedEntities/de/neex20/Case1234ExtentedCompositeNE_Modified.csv`

Regex NE File: Valid local file name of existing file, which contains rules for instantiating basic named entities such as dates, amounts of money, URLs, and e-mail addresses. Each line contains a `java.util.regex.Pattern` regular expression that matches sequences of tokens as well as the corresponding name of the basic named entity. The syntax of this file must conform to `org.hypknowsys.misc.io.CsvFile`. Do not delete the first two meta-data lines and quote all values. The following basic named entities could be instantiated using regular expressions: 'number', 'date', 'time', 'amount_of_money', 'paragraph', 'email', 'url', 'organization_id', 'document_id', 'court', 'postal_code', 'isin', and 'wkn'. Regular expressions must be Java-compliant constructs as specified in the

API documentation of the Java package `gnu.regexp`. Comment lines starting with "#" are ignored. Example:

```
# Format: "searchRegex";"namedEntityType"
Attribute0;Attribute1
String;String
# normalized amounts of money
"\d{1,}[,\.\d]{1,}\s(DEM|EUR|ATS)";"amount_of_money"
# normalized dates
"\d{1,2}\.\d{1,2}\.\d{2,4}";"date"
```

The language-specific subdirectories of `${PARAMETER_HOME}/replaceNamedEntities` contain defaults for all parameter files described above. However, NEEX 2.0 parameter files are provided in the subdirectory `neex20` only.

4.3.5 Module: Remove Stopwords

Stopword File: Valid local file name of existing text file, which contains meaningless stopwords. DIAsDEM Workbench 2.1 can only process single-token terms (e.g., 'und') that do not contain blank spaces. Stopwords are not processed case-sensitively. Hence, the stopword 'aber' also matches the term 'ABER'. Each line contains exactly one stopword. Comment lines starting with "#" are ignored. Example:

```
# Format: One case-sensitive, single-token stopword per line
ab
abend
aber
acht
alle
```

Defaults for *Stopword File* are provided in the language-specific subdirectories of the directory `${PARAMETER_HOME}/removeStopwords`.

4.3.6 Module: Lemmatize Text Units

TreeTagger Input File: The name of this temporary file must be set, if *Use TreeTagger to Determine Lemma Form* is enabled. It must be a valid local file name of either a new or an existing file that are replaced by the module. This file is created by DIAsDEM Workbench and includes text to be POS-tagged by TreeTagger. Example:

```
<Document_/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0>
<ProcessedTextUnit_0>
Die Heizungs - und Sanitärinstallation , Gastechnik und Gassicherheitstechnik ...
</ProcessedTextUnit_0>
<ProcessedTextUnit_1>
Weiter ist Gegenstand die Konzeption , Montage , Instandsetzung und Instandhaltung ...
</ProcessedTextUnit_1>
<ProcessedTextUnit_2>
Stammkapital : <<0>> .
</ProcessedTextUnit_2>
<ProcessedTextUnit_3>
Gesellschaft_mit_beschränkter_Haftung .
</ProcessedTextUnit_3>
<ProcessedTextUnit_4>
Der Gesellschaftsvertrag ist am <<1>> abgeschlossen und am <<2>> abgeändert .
</ProcessedTextUnit_4> ...
<ProcessedTextUnit_10>
Nicht eingetragen : Die Bekanntmachungen der Gesellschaft erfolgen im Bundesanzeiger .
</ProcessedTextUnit_10>
</Document_/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0> ...
```

TreeTagger Output File: The name of this temporary file must be set, if *Use TreeTagger to Determine Lemma Form* is enabled. It must be a valid local file name of either a new or an existing file that is extended by the module. This file is created by TreeTagger and includes the results of POS-tagging for subsequent parsing by DIAsDEM Workbench. Example:

```
<Document_/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0>
<ProcessedTextUnit_0>
Die     ART     d
Heizungs        NN      <unknown>
-       $(      -
und     KON     und
Sanitärinstallation   NN      Sanitärinstallation
...
</ProcessedTextUnit_0>
<ProcessedTextUnit_1>
Weiter  ADV     weiter
ist     VAFIN   sein
Gegenstand      NN      Gegenstand
die     ART     d
...
```

Known Lemma Forms: The name of this parameter file must be set, if *Look Up Lemma Form in List* is enabled. It must be a valid local file name of an existing file that contains

terms along with their lemma forms in the following format: Each line lists one term, exactly one tab stop and thereafter the corresponding grammatical root form. Terms and lemma forms must not be multi-token terms that include blank spaces. However, blank spaces in multi-token terms can be replaced with underscores (e.g., "for_example"). Note that capitalization of terms is irrelevant, but the capitalization of lemma forms is retained when replacing the corresponding terms. Comment lines starting with "#" are ignored. Example:

```
# Format: term<TAB>lemmaFormOfTerm
Datenverarbeitungssysteme    Datenverarbeitungssystem
Arbeitsgemeinschaften Arbeitsgemeinschaft
Formaten      Format
Deetz  Deetz
Fanartikel    Fanartikel
Biographien   Biographie
```

Unknown Lemma Forms: The name of this temporary file must be set, if *Look Up Lemma Form in List* is enabled. It must be a valid local file name of either a new or an existing file that is extended by the module. This file is created or extended by DIAsDEM Workbench and includes terms occuring in the collection that are not listed in the file of *Known Lemma Forms* as well as the context of their occurrence (i.e., the sentence) separated by exactly one tab stop. This file could be used to update the file *Known Lemma Forms* with new terms. Example:

```
# Format: unknownTerm<TAB>correspondingTokenizedTextUnit
lit.   1. Der An - und Verkauf von Immobilien sowie die Beteiligung an ...
Art.   Der Gesellschaftsvertrag ist am <<1>> abgeschlossen und am <<2>> ...
Dip.   Dip. - <<26>> und Dr. jur. <<27>> , sind zu Geschäftsführern bestellt .
Dipl.-Kaufm.  Dipl.-Kaufm. <<47>> , ist zum Geschäftsführern bestellt .
```

4.4 Iterative Clustering

4.4.1 Module: Vectorize Text Units

Vector File Format: DIAsDEM Workbench supports the export of three file formats: comma separated values (CSV files), fixed width values (TXT files) and ARFF files in the Weka-specific format described in [WF99]. See below an example of a text unit vector file in comma separated values format:

```
DocumentType,Document,TextUnit,D1_Aktie,D2_Gesellschafter,D3_Inhaber,...,D73_Anspruch
"null","/home/.../volume100000.xml:0",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
```

```
"null","/home/.../volume100984.xml:0",7,0,0,0,0,0,0,0.879,0,1.407,0,0,0,0,0,0,0,0,...
"null","/home/.../volume100984.xml:0",8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
"null","/home/.../volume100984.xml:0",9,0,0,0,0,0,0,0,0,0,2.500,0,2.539,0,0,0,0,0,...
```

In the current version of DIAsDEM Workbench, “DocumentType” is always “null” due to legacy reasons. The attribute “Document” contains the DIAsDEM document ID. Values of the attribute “TextUnit” uniquely identify text units within a given document by their sequence number. Note that “Document” and “TextUnit” constitute a composite primary key for text units in the scope of the respective *Collection File*. The following meta-data file summarize information about attributes of the above CSV file:

```
DocumentType
Document
TextUnit
D1_Aktie = Aktie; Descriptor Frequency = 37; Descriptor Weight = 5.521
...
D73_Anspruch = Anspruch; Descriptor Frequency = 9; Descriptor Weight = 6.935
```

As of DIAsDEM Workbench 2.1, text unit vectors should not be exported as TXT files with fixed width values. The module *Vectorize Text Units* cannot properly process DIAsDEM documents whose IDs comprise more than 25 characters. However, below is an example of a text unit vector file in fixed width values format:

```
DocumentType␣␣␣␣␣␣␣␣Document␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣TextUnit␣␣D1_Aktie␣␣␣␣␣␣␣␣␣␣␣D2_Gese...
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100000.xml:0␣␣␣␣␣␣0␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣...
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100984.xml:0␣␣␣␣␣␣7␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣...
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100984.xml:0␣␣␣␣␣␣8␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣...
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100984.xml:0␣␣␣␣␣␣9␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0␣␣␣␣␣␣...
```

As indicated above, simple blank spaces separate attribute values from each others. The following meta-data file corresponds to the TXT-file above and additionally contains information about the width of each attribute. Note again that file names of intermediate XML files cannot exceed 25 characters and the width of attributes cannot be modified.

```
1-20   DocumentType
21-45  Document
46-55  TextUnit
56-75  D1_Aktie = Aktie; Descriptor Frequency = 37; Descriptor Weight = 5.521
...
```

See below an example of a Weka-specific text unit vector file in ARFF-format:

```
@relation 'DIAsDEM'
@attribute DocumentType string
@attribute Document string
@attribute TextUnit string
@attribute D1_Aktie real ...
@attribute D73_Anspruch real
@data
null,/home/.../volume100000.xml:0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
null,/home/.../volume100984.xml:0,7,0,0,0,0,0,0,0.879,0,1.407,0,0,0,0,0,0,0,0,0,0,...
null,/home/.../volume100984.xml:0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...
null,/home/.../volume100984.xml:0,9,0,0,0,0,0,0,0,0,0,2.500,0,2.539,0,0,0,0,0,0,0,...
```

The following meta-data file corresponds to the ARFF-file above:

```
DocumentType
Document
TextUnit
D1_Aktie = Aktie; Descriptor Frequency = 37; Descriptor Weight = 5.521
...
D73_Anspruch = Anspruch; Descriptor Frequency = 9; Descriptor Weight = 6.935
```

Thesaurus File: The existing DIAsDEM-specific thesaurus file must be identified by a valid local file name. Except for comment lines starting with "#", each line corresponds to exactly one thesaurus entry that can either be a descriptor (i.e., preferred term) or a non-descriptor (i.e., non-preferred term). Non-descriptor terms must always point to an associated descriptor in the same thesaurus file that should be used for indexing and term frequency counting instead. Note that DIAsDEM-specific thesauri must only include grammatical root forms of terms (i.e., their so-called lemma forms) as determined by the module *Actions* → *Prepare Data Set* → *Vectorize Text Units*. Thesauri can be created by *Actions* → *Understand Domain* → *Establish Initial Thesaurus* and modified by *Tools* → *Thesaurus Editor*. Example:

```
# Terms of Thesaurus /home/.../data/parameters/thesauri/de/Case123Thesaurus.dth
19535 "<<company>>" 1 "TY=D" "HL=-" "SY=-" "BT=-" "NT=-" "UD=-" "SN=Case1"
19534 "<<person>>" 1 "TY=D" "HL=-" "SY=-" "BT=-" "NT=-" "UD=-" "SN=Case1"
19461 "Ablehnung" 1 "TY=D" "HL=-" "SY=-" "BT=-" "NT=-" "UD=-" "SN=Case2"
10628 "abschließen" 1 "TY=D" "HL=-" "SY=-" "BT=-" "NT=-" "UD=-" "SN=Case1 Case2" ...
12299 "Änderung" 1 "TY=D" "HL=-" "SY=-" "BT=-" "NT=-" "UD=-" "SN=Case1 Case2" ...
10859 "Zwecke" 1 "TY=N" "HL=-" "SY=-" "BT=-" "NT=-" "UD=Tätigkeit" "SN=-"
10797 "ändern" 1 "TY=N" "HL=-" "SY=-" "BT=-" "NT=-" "UD=Änderung" "SN=-"
```

Thesaurus terms can either be lemma forms of words (e.g., "Ablehnung" and "ändern") or named entity type placeholders such as "<<company>>" and "<<person>>". Consider for example the thesaurus entry "Ablehnung": "19461" is a unique term identifier

within the thesaurus, the term type field "TY=D" denotes that "Ablehnung" is a descriptor term and the scope notes field "SN=Case2" can be used to filter valid descriptors in different case studies and clustering iterations, respectively. The use descriptor field ("UD=-") remains empty for descriptor terms for obvious reasons. Furthermore, consider the thesaurus entry "ändern" which is a non-descriptor ("TY=N"). The descriptor term "Änderung" should be used instead of "ändern", because of the use descriptor field "UD=Tätigkeit". Note, hierarchy level ("HL=-"), synonyms ("SY=-"), broader term ("BT=-") and narrower term ("NT=-") are not used in DIAsDEM Workbench 2.1.

4.4.2 Module: Cluster Text Unit Vectors (Weka)

Text Unit Vectors File: Valid local file name containing text unit vectors to be clustered in ARFF format as specified above in section 4.4.1. Note that the internal Weka-based clustering algorithms cannot process other file formats.

Clustering Results File: Valid local file name of a file to be created or replaced by DIAsDEM Workbench, which contains the mappings of text units onto clusters in CSV format as specified below in section 4.4.3. Note that all internal Weka-based clustering algorithms cannot output other file formats.

Text Unit Clusterer File: Valid local file name of a file to be created or replaced by DIAsDEM Workbench, which contains a serialized instance of the Java class corresponding to *Clustering Algorithm*. *Text Unit Clusterer File* is an output parameter in clustering mode, but an input parameter in application mode. Note, there must be a correspondance between the specified *Clustering Algorithm* in clustering and application mode. In the latter phase, an instance of the respective text unit clusterer is created as follows:

```
modelInputStream = new ObjectInputStream( new FileInputStream(
  CastParameter.getClusterModelFileName()));
switch (CastParameter.getClusteringAlgorithm()) {
  case ClusterTextUnitVectorsWekaParameter.WEKA_SIMPLE_KMEANS: {
    clusterer = (SimpleKMeans)modelInputStream.readObject();
    break;
  }
  case ClusterTextUnitVectorsWekaParameter.WEKA_COBWEB: {
    clusterer = (Cobweb)modelInputStream.readObject();
    break;
  }
  case ClusterTextUnitVectorsWekaParameter.WEKA_EM: {
    clusterer = (EM)modelInputStream.readObject();
    break;
  }
}
modelInputStream.close();
```

4.4.3 Module: Monitor Cluster Quality

Result File Format: *Cluster Result File* maps text unit vectors onto their respective clusters that are identified by integers. Currently, each text unit vector can only be assigned to exactly one cluster. DIAsDEM Workbench can import result files in the following two formats: comma separated values (CSV files) and fixed width values (TXT files). In both cases, *Cluster Result File* must contain exactly three attributes for each text unit vector. The DIAsDEM document ID is the first attribute. It is followed by the text unit identifier as the second and the cluster ID associated with the respective text unit vector as the third attribute. The first two attributes (i.e., file name and text unit identifier) must exactly correspond to the attributes "Document" and "TextUnit" of text unit vector files as described in section 4.4.1. Valid cluster IDs are integers being greater than zero. Text units vectors in *Cluster Result File* should be ordered as in the corresponding *Text Unit Vectors File*.

Note, DIAsDEM Workbench can only process files that completely conform to the syntax exemplified by the following two file excerpts. Hence, clustering algorithms must either be configured to create appropriate result files or intermediate output files must be post-processed by for example Perl scripts. See below an example of a cluster result file in comma separated values format:

```
/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0,0,25
/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0,1,25
/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0,2,9
/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0,3,48
/home/.../tutorial/trainingProject/inputCollection/volume100000.xml:0,4,34
```

As of DIAsDEM Workbench 2.1, clustering results should not be exported as TXT files with fixed width values. The modules *Monitor Cluster Quality* and *Tag Text Units* cannot properly process DIAsDEM documents whose IDs comprise more than 25 characters. However, below is an example of a text unit vector file in fixed width values format:

```
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100000.xml:0␣␣␣␣␣␣1␣␣␣␣␣␣␣␣␣25
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100000.xml:0␣␣␣␣␣␣2␣␣␣␣␣␣␣␣␣25
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100000.xml:0␣␣␣␣␣␣3␣␣␣␣␣␣␣␣␣9
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100000.xml:0␣␣␣␣␣␣4␣␣␣␣␣␣␣␣␣48
null␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/volume100000.xml:0␣␣␣␣␣␣5␣␣␣␣␣␣␣␣␣34
```

As indicated above, only blank spaces are allowed to separate attribute values from each others. The following meta-data file corresponds to the TXT file above and contains information about the width of each attribute. Note that file names of intermediate XML files cannot exceed 25 characters. Currently, the width of attributes cannot be changed

by the user. In the current version of DIAsDEM Workbench, "DocumentType" has always the "null" value due to legacy reasons. In contrast to fixed width files, CSV files must not contain the attribute "DocumentType".

```
1-20   DocumentType
21-45  Document
46-55  TextUnit
56-58  ClusterID
```

Cluster Result File: Valid local file name of a file to be created or replaced by DIAsDEM Workbench that conforms to *Result File Format*.

Thesaurus File: Valid local file name of existing DIAsDEM-specific thesaurus file as described in section 4.4.1.

4.4.4 Module: Tag Text Units

Result File Format: One of two result file formats (i.e., comma separated values and fixed width values), which are supported by DIAsDEM Workbench and described in section 4.4.1.

Cluster Result File: Valid local file name of a file to be created or replaced by DIAsDEM Workbench. *Cluster Result File* must conform to *Result File Format*.

Cluster Label File: Valid local file name of existing file created by DIAsDEM Workbench in *Actions* → *Postprocess Patterns* → *Monitor Cluster Quality* and possibly modified by *Tools* → *Cluster Label Editor*.

4.5 XML Tagging of Texts

4.5.1 Module: Derive Preliminary DTD

Preliminyra DTD File: Valid local file name of file to be created or replaced by DIAsDEM Workbench, which contains meta-data about the preliminary XML document type definition, its XML tags and their attribues in a DIAsDEM-specific format. The following *Preliminary DTD File* has been created in this case study:

```
#This is an automatically created file: Please do not edit this file manually!
#Mon Sep 01 23:21:46 CEST 2003
PRELIMINARY_DTD_REMARKS=Created Mon Sep 01 23\:21\:46 CEST 2003
NUMBER_OF_UNTAGGED_TEXT_UNITS=745
ELEMENTS_FILE_NAME=preliminaryDtd.dpd.elements
MIN_ATTRIBUTE_REL_SUPPORT=0.1
ROOT_ELEMENT=CommercialRegisterEntry
NUMBER_OF_TEXT_UNITS=9254
```

```
XML_DTD_FILE_NAME=preliminaryDtd.dpd.xml
NUMBER_OF_TAGGED_TEXT_UNITS=8509
NUMBER_OF_DOCUMENTS=985
TRAINING_COLLECTION_FILE_NAME=/home/.../tutorial/trainingProject/collection.dcf
ATTRIBUTES_FILE_NAME=preliminaryDtd.dpd.attributes
```

The file `/home/.../tutorial/trainingProject/preliminaryDtd.dpd.elements` contains meta-data about DTD elements (i.e., XML tags):

```
# UnstructuredDtdElement,AbsoluteSupport,RelativeSupport
"AppointmentOfManagingDirector" 831 0.8436548223350254
"ChangeOfFirmName" 57 0.05786802030456853
"CommencementOfPartnership" 113 0.11472081218274112
...
"SolePowerToRepresent_PowerToContractWithOneself" 535 0.5431472081218274
```

The file `/home/.../tutorial/trainingProject/preliminaryDtd.attributes` contains meta-data about attributes of DTD elements (i.e., XML tags) as exemplified below:

```
# PreliminaryDtdElement,UnstructuredDtdAttribute,MostProbableType,
# AbsoluteSupport,RelativeSupport
"AppointmentOfManagingDirector" "Date" "Date" 134 0.16125150421179302
"AppointmentOfManagingDirector" "Person" "Person" 775 0.9326113116726835
"AppointmentOfManagingDirector" "Place" "Place" 145 0.17448856799037304
...
"ShareCapital" "AmountOfMoney" "AmountOfMoney" 775 0.9948652118100129
```

4.5.2 Module: Tag Documents

Preliminary DTD File: Valid local file name of existing file, which is created by DIAsDEM Workbench and contains meta-data about the preliminary XML document type definition, its XML tags and their attribues in a DIAsDEM-specific format.

Random Sample File: Valid local file name of file to be created or replaced by DIAsDEM Workbench, which contains a random sample from all text units (i.e., both tagged and untagged ones) in a DIAsDEM-specific format. Along with *Preliminary DTD File*, this file is input to the module *Tools* → *Assessment of Tagging Quality*. For example, see below three lines of *Random Sample File* as created in this case study. Note that the first three line correspond semantically annotated sentences, whereas the fourth ones contains an untagged sentences.

```
/home/.../tutorial/trainingProject/inputCollection/volume100002.xml:0
  <IfAppointmentOfOneManagingDirector_SolePowerToRepresent>Ist nur ein
  Geschäftsführer bestellt, so vertritt er die Gesellschaft einzeln.
  </IfAppointmentOfOneManagingDirector_SolePowerToRepresent>
/home/.../tutorial/trainingProject/inputCollection/volume100003.xml:0
  <PurposeOfCompany>(Gegenstand: Durchführung von vermessungstechnischen
  Arbeiten).</PurposeOfCompany>
/home/.../tutorial/trainingProject/inputCollection/volume100005.xml:0
  <NameOfMerchant Person="3; Stefan Thümmler; null; null; null; null;
  null; null; null; null">Inhaber: Stefan Thümmler, Kaufmann, Wustermark.
  </NameOfMerchant>
/home/.../tutorial/trainingProject/inputCollection/volume100135.xml:0
  Die Gründer der Gesellschaft, die sämtliche Aktien übernommen haben ist ECC
  Treuhand- und Verwaltungsgesellschaft mbH mit Sitz in München.
```

List of Abbreviations

DFG	Deutsche Forschungsgemeinschaft (German Research Society)
DIAsDEM	Datenintegration von Altlastdaten und semistrukturierten Dokumenten mit Mining Verfahren (German project acronym that stands for "integration of legacy data and semi-structured documents with data mining techniques")
DTD	Document Type Definition
FN	false negative
FP	false positive
geb.	geboren (born on a date)
HHL	Handelshochschule Leipzig (Leipzig Graduate School of Management)
ID	identifier
KDD	Knowledge Discovery in Databases
KDT	Knowledge Discovery in Textual Databases
NE	named entity
NEEX	Named Entity Extractor (module of DIAsDEM Workbench)
POS	part-of-speech
Regex	regular expression
TF	term frequency
TFxIDF	term frequency multiplied by inverse document frequency
TN	true negative
TP	true positive
Weka	Waikato Environment for Knowledge Analysis
XML	Extensible Markup Language

List of Relevant German Vocabulary

The following list contains German nouns and verbs that might be useful to understand the meaning of Commercial Register entries in this case study. This list is based on a translation of the German Commercial Code by Peltzer, Doyle and Voight which includes a concise introduction to the German Commercial Code as well [PDV00, pp. 1–32].

Aktiengesellschaft (AG) German joint stock corporation

Aktionär (Aktionäre) shareholder of German joint stock corporation (AG)

Amtsgericht District Court in Germany; a local Commercial Register is usually maintained by the respective District Court

Änderung change or modification of sth. (e.g., modification of partnership agreement)

Anspruch legal claim against sb.

Bauvorhaben building project; here: purpose of certain companies

Beginn here: commencement of operations

beginnen (beginnt) here: to commence with operations

Bekanntmachung (Bekanntmachungen) information that has to be officially published by companies according to the German Commercial Code

Bundesanzeiger official German newspaper that weekly publishes Commercial Register entries and corporate news

bestellen (bestellt) here: to appoint sb. to a position of responsibility (e.g., to appoint sb. as managing director of a German limited liability company)

eingetragen here: (e.g., legal facts) to be registered with the Commercial Register

Einzelvertretungsbefugnis sole power to legally represent a company (in contrast to joint power to represent a company)

erfolgen (erfolgt) here: to publish information according to the German Commercial Code

Erhöhung increase in sth. (e.g., increase in share capital)

erteilen (erteilt) here: to confer (e.g., Prokura or power to represent a company)

Firma here: legal name of a company as registered in the respective Commercial Register; legal name under which a merchant transacts business and executes agreements; a merchant may sue and may be sued under his firm name

Geschäftsführer, Geschäftsführerin managing director of German limited liability company (GmbH)

Gesellschafter, Gesellschafterin partner in German commercial partnership (e.g., OHG and KG) or in German limited liability company (GmbH)

Gesellschaft here: (commercial) partnership and company, respectively

Gesellschaft mit beschränkter Haftung (GmbH) German limited liability company

Gesellschafterversammlung meeting of (commercial) partners and share holders, respectively

Gesellschaftsvertrag commercial partnership agreement

Handel mit Waren trading of goods

Kommanditist (Kommanditisten) fully liable partner in German limited partnership (KG)

Kommanditgesellschaft (KG) German limited partnership

Offene Handelsgesellschaft (OHG) German commercial partnership

Prokura power to legally represent a company regulated by the German Commercial Code; Prokura includes all judicial and non-judicial transactions that are related to the operations of a commercial business; Prokura might be conferred with either sole or joint power of representation

Stammkapital share capital of German limited liability company (GmbH)

Tätigkeit here: purpose of company

vertreten (vertritt) here: to legally represent a company

Vorstand managing board of German joint stock company (AG)

Zweigniederlassung branch office of a company

Bibliography

[GSW01] Henner Graubitz, Myra Spiliopoulou, and Karsten Winkler. The DIAsDEM framework for converting domain-specific texts into XML documents with data mining techniques. In *Proceedings of the First IEEE International Conference on Data Mining*, pages 171–178, San Jose, CA, USA, November/December 2001.

[GWS01] Henner Graubitz, Karsten Winkler, and Myra Spiliopoulou. Semantic tagging of domain-specific text documents with DIAsDEM. In *Proceeding of the 1st International Workshop on Databases, Documents, and Information Fusion (DBFusion 2001)*, pages 61–72, Magdeburg, Germany, May 2001.

[ISO86] ISO. Documentation: Guidelines for the establishment and development of monolingual thesauri. Technical Report ISO 2788-1986 (E), International Organisation for Standardization, 1986.

[PDV00] Martin Peltzer, Jonathan J. Doyle, and Elizabeth A. Voight. *German Commercial Code: German-English Text with an Introduction in English.* Verlag Dr. Otto Schmidt, Köln, 4th revised edition, 2000.

[Sch94] Helmut Schmid. Probabilistic part–of–speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK, September 1994. TreeTagger is available at http://www.ims.uni-stuttgart.de/∼schmid, accessed 2003-09-01.

[Sul01] Dan Sullivan. *Document Warehousing and Text Mining.* John Wiley & Sons, New York, Chichester, Weinheim, 2001.

[SW02] Myra Spiliopoulou and Karsten Winkler. Text Mining auf Handelsregistereinträgen: Der SAS Enterprise Miner im Einsatz. In Klaus D. Wilde, Hajo Hippner, and Melanie Merzenich, editors, *Data Mining: Mehr Gewinn aus Ihren Kundendaten*, pages 117–124. Verlagsgruppe Handelsblatt, Düsseldorf, 2002.

[WF99] Ian H. Witten and Eibe Frank. *Data Mining: Practical Mechine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, San

Francisco, 1999. Weka is available at http://www.cs.waikato.ac.nz/∼ml/weka, accessed 2003-09-01.

[Win03] Karsten Winkler. Getting started with DIAsDEM Workbench 2.0: A case-based approach. HHL Working Paper No. 58, HHL – Leipzig Graduate School of Management, Leipzig, Germany, 2003. DIAsDEM Workbench 2.0 is available at http://www.hypknowsys.org/diasdem, accessed 2003-09-01.

[WS01a] Karsten Winkler and Myra Spiliopoulou. Extraction of semantic XML DTDs from texts using data mining techniques. In *Proceedings of the K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation*, pages 59–68, Victoria, BC, Canada, October 2001.

[WS01b] Karsten Winkler and Myra Spiliopoulou. Integrating data and probabilistically structured text documents. In *Proceedings des 5. Workshops "Föderierte Datenbanken" und GI Arbeitstreffen "Konzepte des Data Warehousing" (FDBS 2001)*, pages 16–29, Berlin, Germany, October 2001.

[WS01c] Karsten Winkler and Myra Spiliopoulou. Semi-automated XML tagging of public text archives: A case study. In *Proceedings of EuroWeb 2001 "The Web in Public Administration"*, pages 271–285, Pisa, Italy, December 2001.

[WS02a] Karsten Winkler and Myra Spiliopoulou. Employing text mining for semantic tagging in DIAsDEM. *KI – Künstliche Intelligenz*, 16(2):27–29, 2002.

[WS02b] Karsten Winkler and Myra Spiliopoulou. Structuring domain-specific text archives by deriving a probabilistic XML DTD. In *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02)*, pages 461–474, Helsinki, Finland, August 2002.

[WS02c] Karsten Winkler and Myra Spiliopoulou. Text Mining in der Wettbewerberanalyse: Konvertierung von Textarchiven in XML-Dokumente. In *Data Mining und Statistik in Hochschule und Wirtschaft: Proceedings der 6. Konferenz der SAS-Anwender in Forschung und Entwicklung (KSFE)*, pages 347–363, Dortmund, Germany, February/March 2002.