BBCheck:

BoB Chatterbot Checker

v. 001

1 Main Program Information

1.1 Program identification

1.1.1 **BBCheck** (BoB checker) GUI application.

1.1.2 Version 1.0

1.2 Brief description of an application

1.2.1 **BBCheck** is a tool with a GUI interface for checking the BoB chatterbot's topic-tree with the given set of test questions. It is able to identify if the question was matched to the correct topic-tree node and as a result it returns the percent of successfully matched (and therefore responded) questions and a detailed report in excel format.

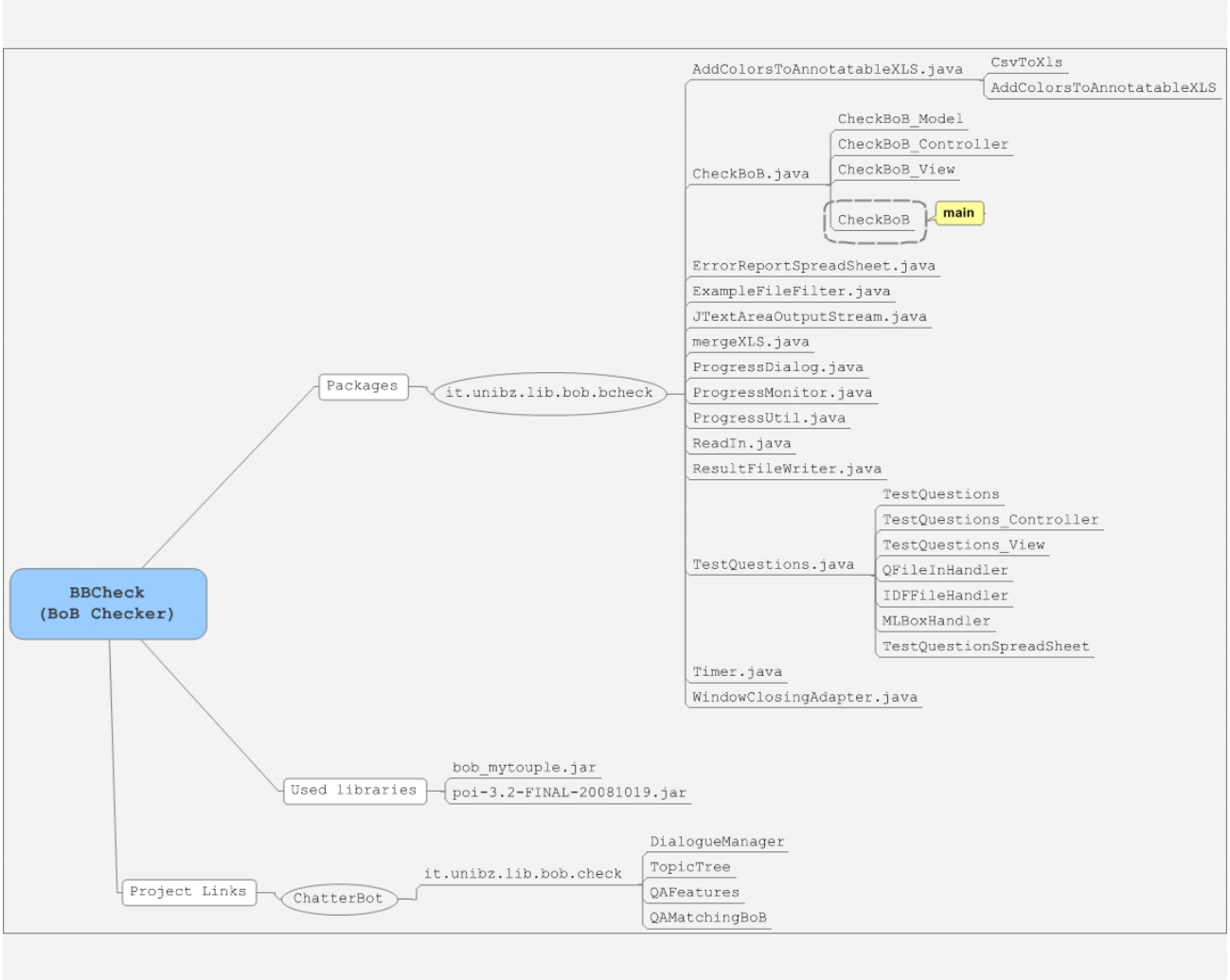
1.2.2 **BBCheck** loads specified topic-tree file into a tree memory structure and normalize it using abbreviation file (substitute abbreviations and copies linked topics) and loads question-answer file (a list of pairs: question, topic-tree ID). Then using `DalogueManager,` class from chatterbot project checks all topic-tree matches for each question from the list: `getAllPossibleNormalResponses` function that returns a vector of possible solutions. If there is more than one solution for current test question **BBCheck** performs answers re-ranking with `rerankAnswers` functions from `QAMatchingBob` class (this is done by using TF/IDF string similarity metric and loaded text corpus).

2 Program functions

BBCheck allows the user to perform only one action, notably checking the topic-tree with the test questions file, which contains the list of (test question)/ (topic-tree ID of the answer) pairs

3 Program Layout

3.1 The major part of the **BBCheck** logic is implemented in `CheckBoB_Model` class, which provides algorithm framework for reading and loading input files, enumerating test questions and writing the output report. To find out a particular question answers `CheckBoB_Model` uses `DialogueManager` class provided by link from the **chatterbot** project. `QAMatchingBob` class is used for answers re-ranking. BBCheck classes are organized in a way to support MVC paradigm, so "view" classes are responsible for visualization, "model" classes contains data and implement algorithms and "controllers" link them together and attach action listeners.



Classes:

`CheckBoB` contains the main function where `CheckBoB_Model` and `CheckBoB_View` are created and then passed to `CheckBoB_Controller`.

`CheckBoB_View` **"**view" class extends the standard Jframe. Contains swing elements for visualization and dialogs for the input files selection.

`CheckBoB_Controller` implements the java ActionListener. It holds the links for “model” and “view” classes. When the checking action is performed, `CheckBoB_Controller` creates the second “check” window by initializing `TestQuestions_View` and `TestQuestions_Controller`.

`CheckBoB_Model` is a thread that implements main program logic. It supports asynchronous call by exposing `execAsyncron` function (it runs new thread attached to current checkbob_model class). `CheckBoB_Model` contains all the links to input files and instances of DialogueManager and `QAMatching` classes.

3.2 Program structure (see attached bob_bbcheck_classes_v002.jpeg)

3.2.1 Packages and classes

`it.unibz.lib.bob.bcheck` contains following classes (look the classes diagrams):

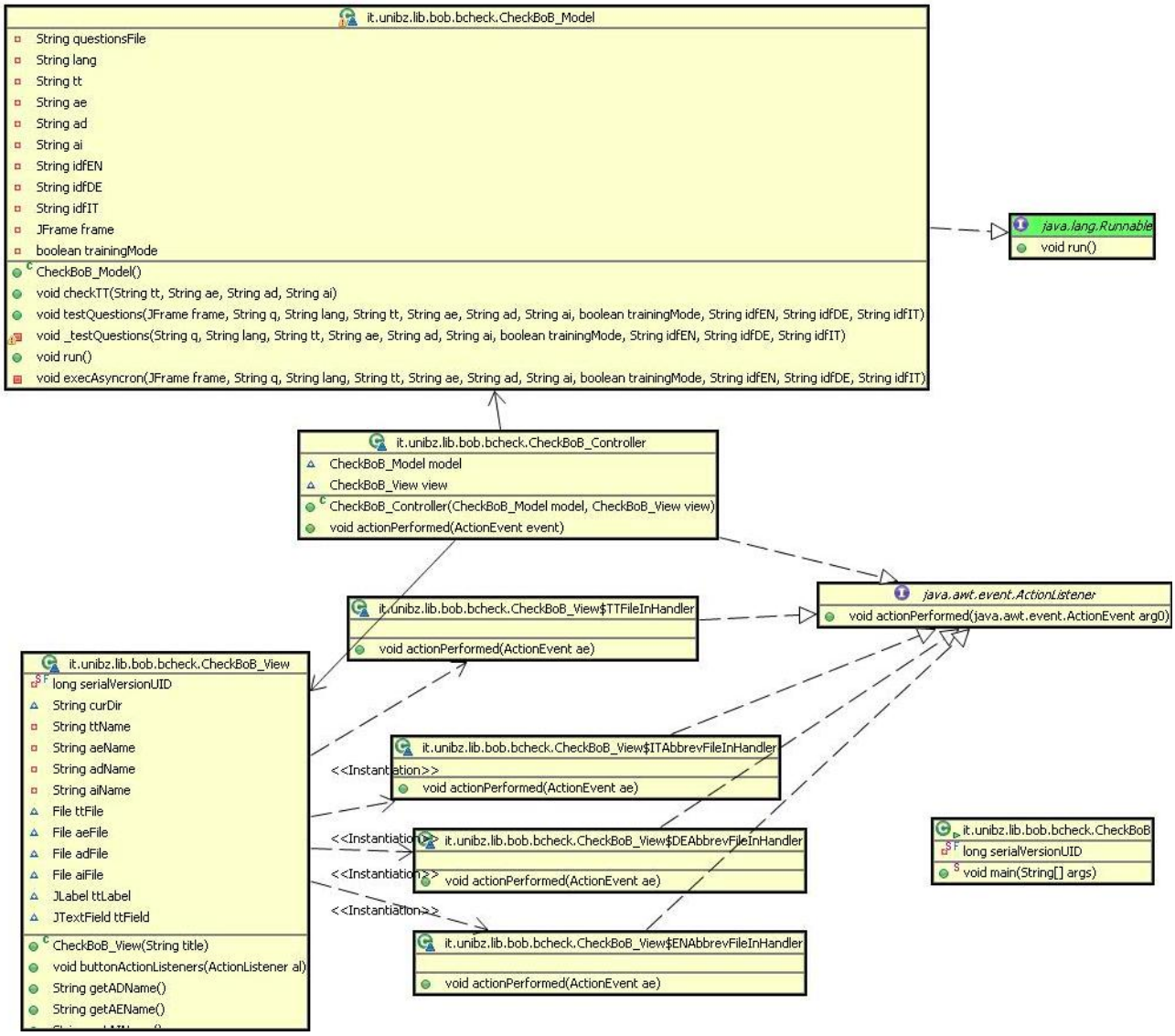
- `AddColorsToAnnotatableXLS` Loads an XLS sheet containing annotatable bob logs and color-code the annotatable cells to make hand-annotation easier. Is not used in current version of BBCheck.
- `CheckBoB`
- `ErrorReportSpreadSheet` XLS report writer class. used to define XLS file formatting and output rows.
- ExampleFileFilter A convenience implementation of FileFilter that filters out all files except for those type extensions that it knows about. Used in `TestQuestions_View` and `CheckBoB_View` views to filter files in user dialogs.
- `JTextAreaOutputStream` A lightweight component provides source compatibility with the `java.awt.TextArea` class and customize OutputStreem to do output into JTextArea, which is a multi-line area that displays plain text. Is used in `TestQuestions_View` to redirect the standard output into JTextArea.
- `mergeXLS` is used for merging several XLS documents in one by placing them into different excel sheets.
- `ProgressDialog` Class from MySwing: Advanced Swing Utilities Copyright (C) 2005 Santhosh Kumar. ProgressDialog has a JLabel, which shows the status message and

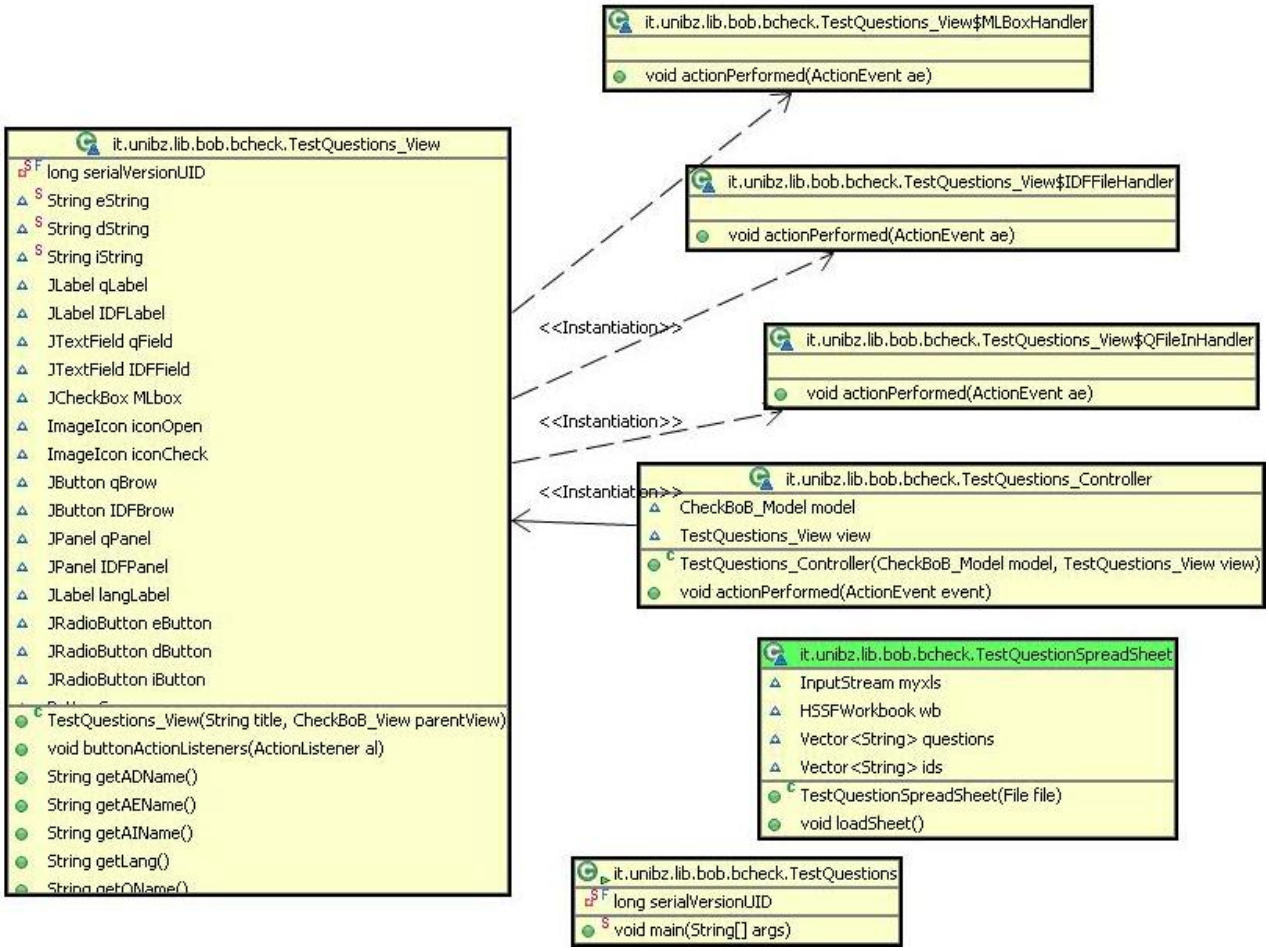
JProgressBar to show amount of progress done. It listens to ChangeEvents from ProgressMonitor and updates the JLabel and JProgressBar. When ProgressMonitor's value hits its total value, dialog is disposed. It ensures that GUI is updated in EDT thread, because ChangeEvents from ProgressMonitor are always fired in Non- EDT thread (see stateChanged method).

- `ProgressMonitor` Class from MySwing: Advanced Swing Utilities Copyright (C) 2005 Santhosh Kumar. A model for progress of task. Progress always starts from zero. User specifies the total units count and whether it is indeterminate or not. The arguments milliSecondsToWait tells, after how much time after the task started, the progress dialog should be shown. This value defaults to half second. Because if GUI does not respond within half second, user will clearly fees that GUI is slow. If the task completes before the milliSecondsToWait is elapsed, then progress dialog is not shown. ProgressMonitor fires ChangeEvent that can be listened by some other class to show progress dialog. Thus, GUI is decoupled from model. ProgressMonitor can be used to show a modal progress dialog or event show the progress in status bar of the application, because we decoupled how progress is shown from its model.
- `ProgressUtil` Class from MySwing: Advanced Swing Utilities Copyright (C) 2005 Santhosh Kumar. Until class for creating Progressmonitor and link it to progressDialog.

- `ReadIn` reads input data of various types from stdin, file, URL using defined char set encoding. Used only once in AddColorsToAnnotatableXLS class for Converting CSV (UTF-8 format) into XLS.
- `ResultFileWriter` Write results to ARFF (Attribute-Relation File Format) or CSV. At some point should provide a nice API for writing correct ARFF files.
- `TestQuestions`
- `Timer` Simple helper class that remember system time on creation and the print the elapsed time on request.
- `WindowClosingAdapter` Extends awt WindowAdapter class, which is an abstract adapter class for receiving window events. The methods in this class are empty. Window closing

adapter is a closing event listener, which we add to `TestQuestions_View` and `CheckBoB_View` frames with addWindowListener routine. WindowAdapter has a constructor flag that specify if the whole program should be terminated on window close or not.





3.2.2 External JAR libraries:

`Bob_mytuple.jar`: mallardsoft open source library, that presents classes for working with ordered multi-value collections.

`poi-3.2-FINAL-20081019.jar`: Apache POI - Java API to access Microsoft format files. POI (Poor Obfuscation Implementation) API is a way to access Microsoft document formats from Java.

3.2.3 Project dependences:

`Chatterbot`: DialogueManager and QAMatchingBob classes for question answering and answer re-ranking respectively.

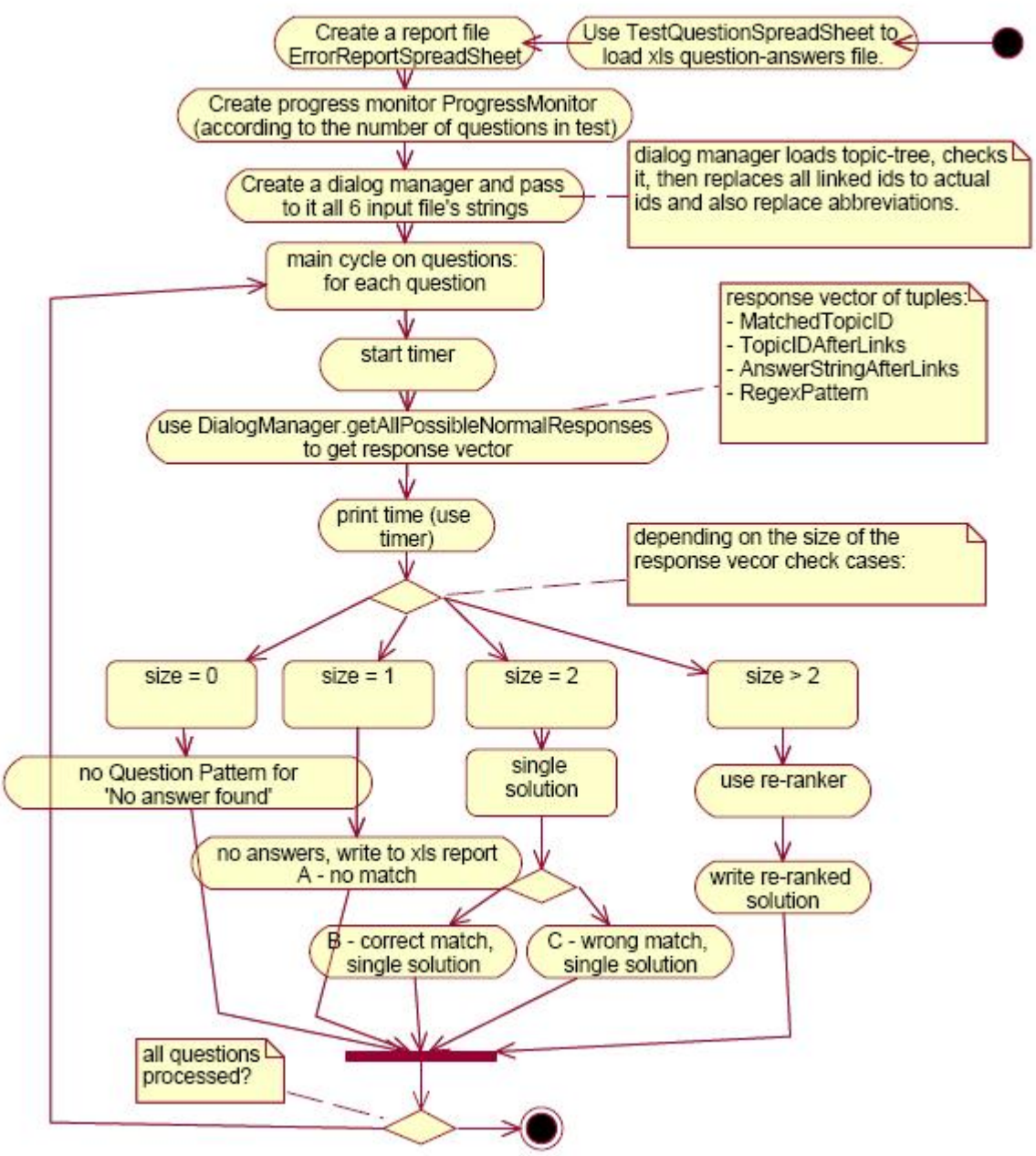
3.3 Compiling and linking

Linking: export as a single runnable jar file.

4 Program Flow

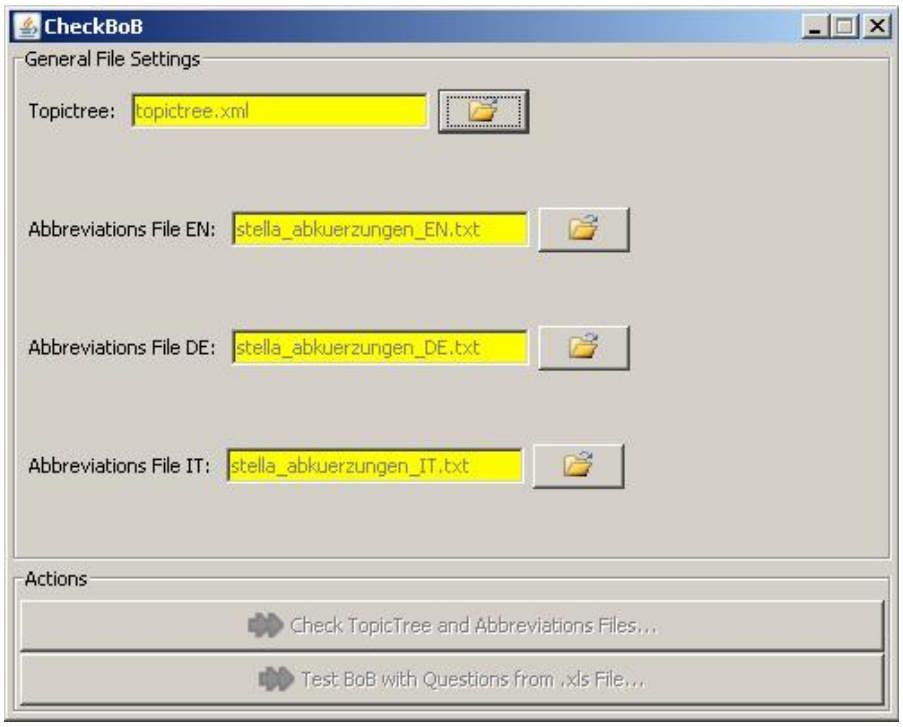
4.1 Program flow diagram

Hereby is the program flow diagram of `_testQuestions` procedure of `CheckBoB_Model` class, which is the main algorithmic part of **bbCheck**.



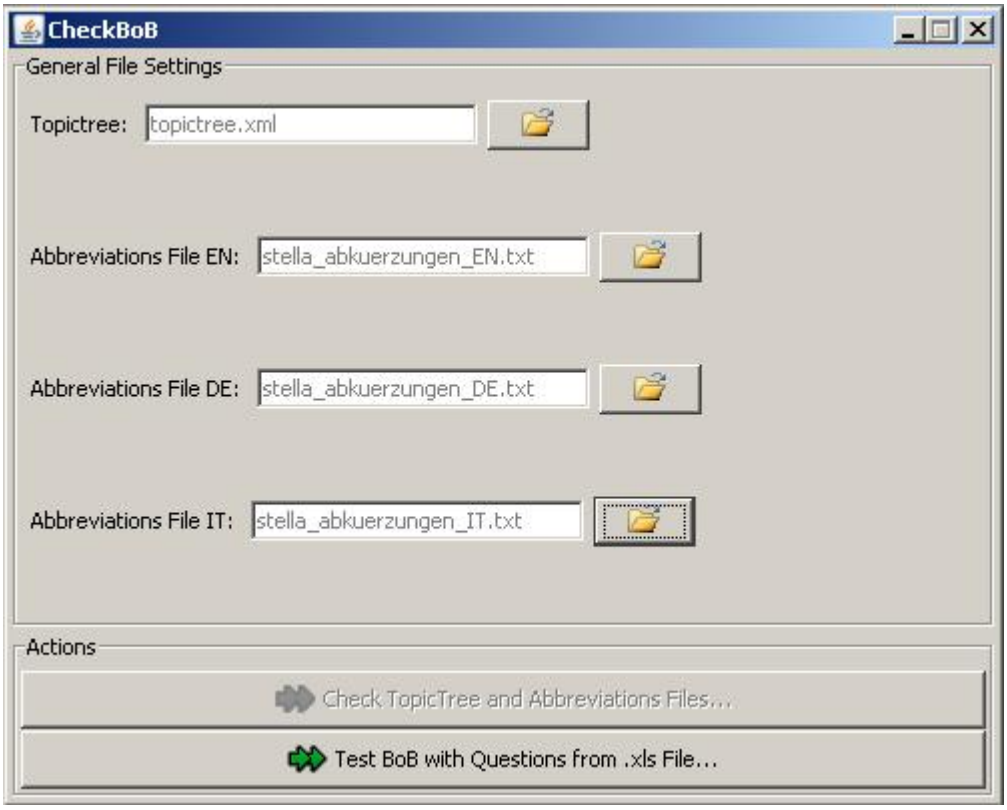
4.2 Dialogs description

"Main view" allows the user to specify the topic-tree file and 3 abbreviation files (English, German and Italian). All 4 input files are required to proceed the checking.



Select topic-tree and abbreviation file

After all the parameters have been specified test-bob action button becomes active and user can proceed.



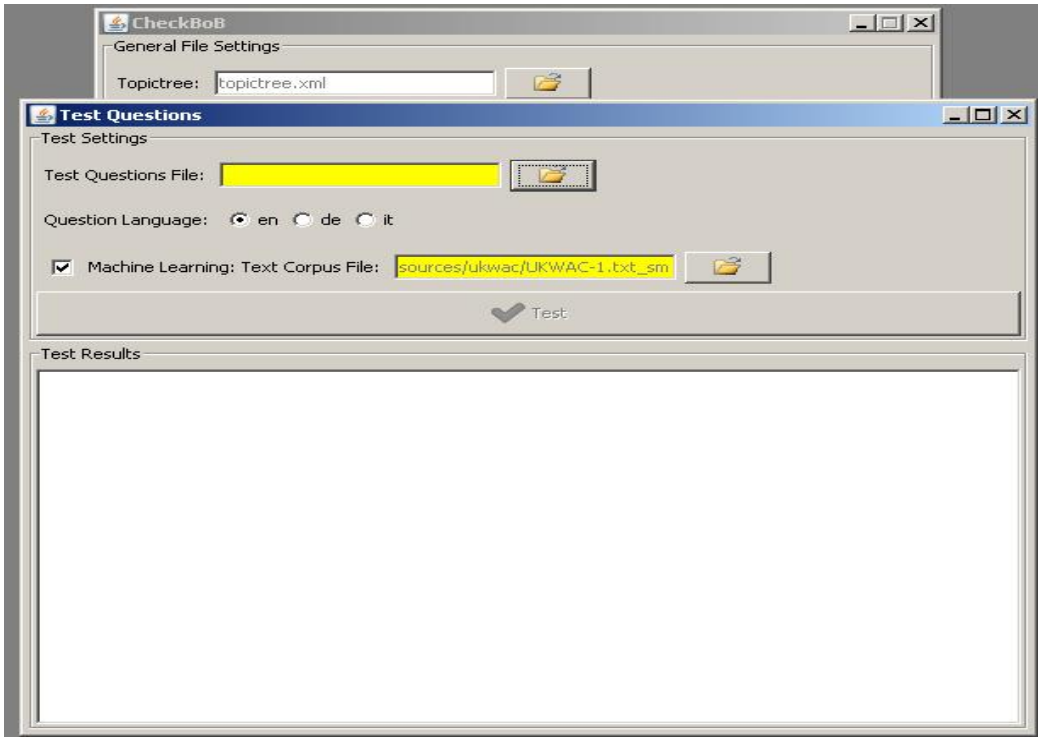
Topic-tree and abbreviations specified

"Test Questions view". After clicking "test bob with Questions" button another modal dialog appears, where user have to select test-questions file, language of testing and machine-learning text corpus for re-ranking (optional). "Test Questions View" also contains output console where outcomes and system messages are printed during the program execution.

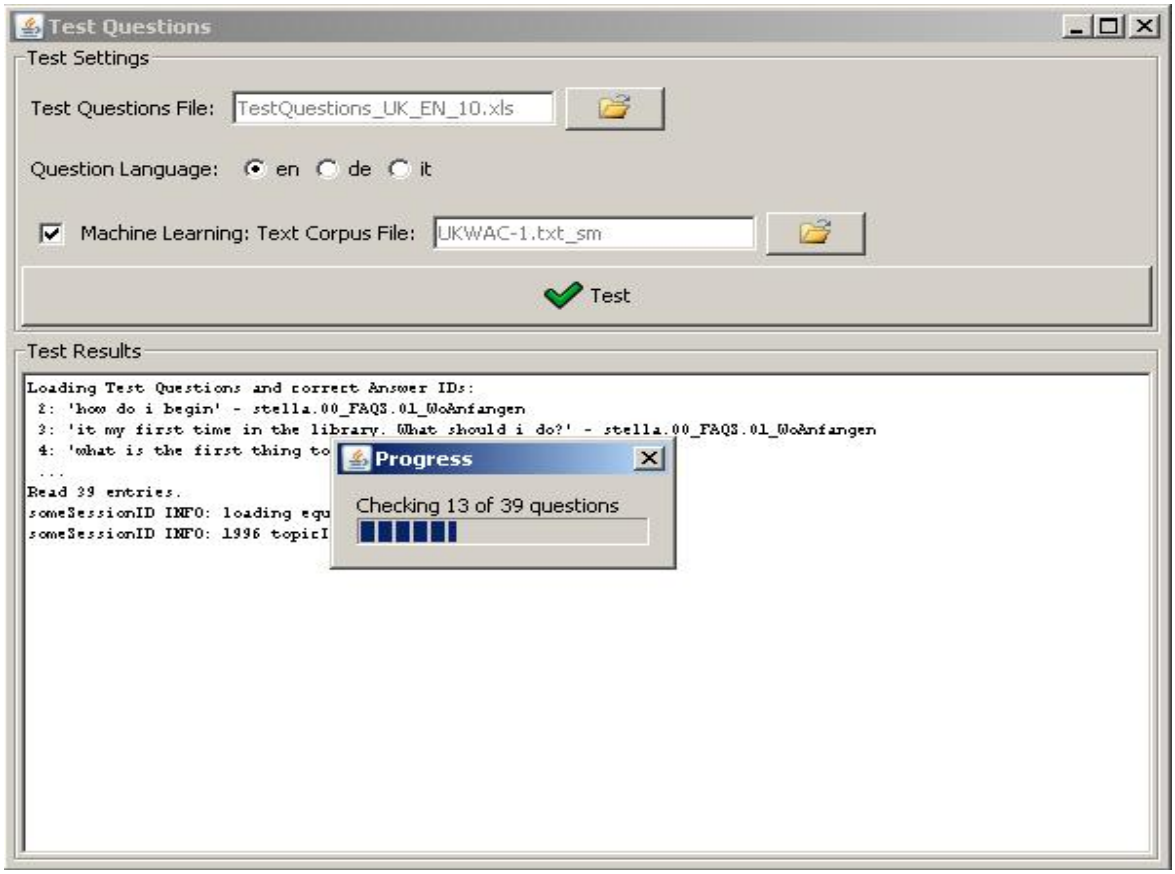
When question-file is specified user can click "test" button to start the process (testing progress figure). During the execution **Bcheck** shows the progress bar with the number of processed questions and the total indicated.

As soon as the work finished progress window disappears and in the console one can see the information resulted and the name of the created report file (test results figure).

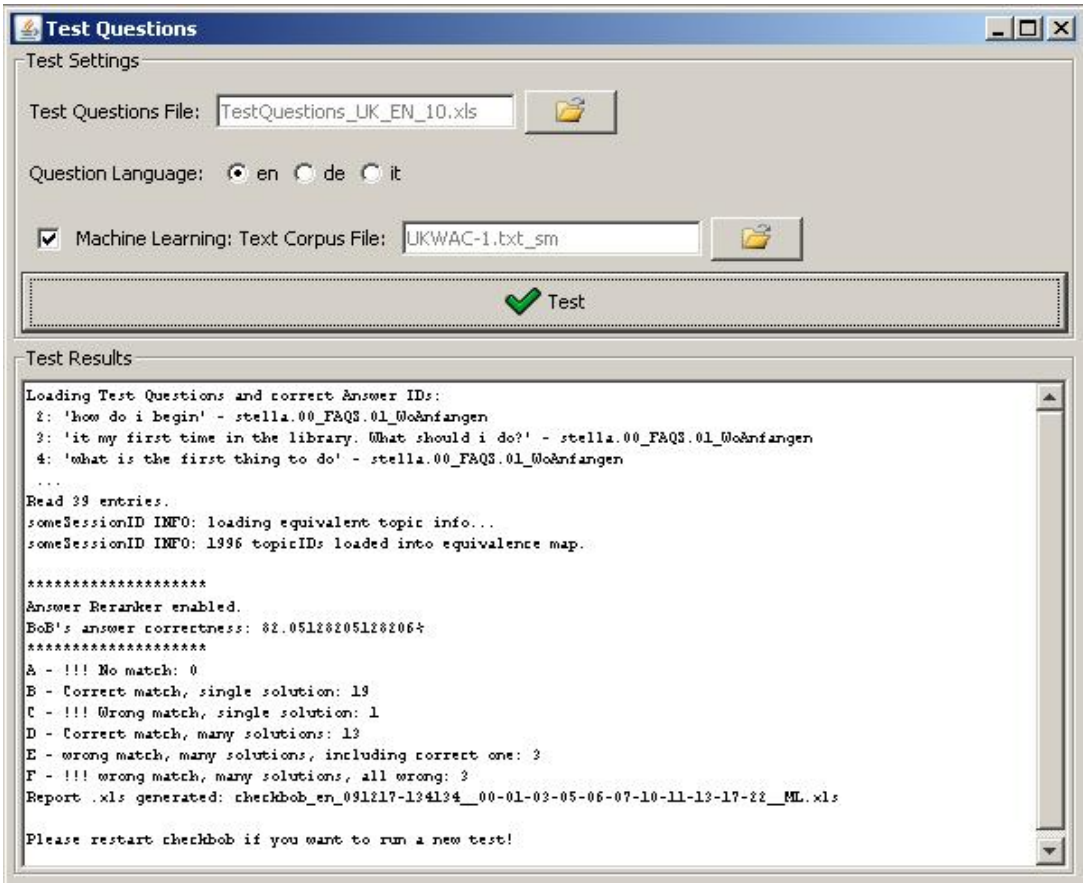
In order to perform the next check it's needed to restart the **Bcheck** application.



Test Questions View

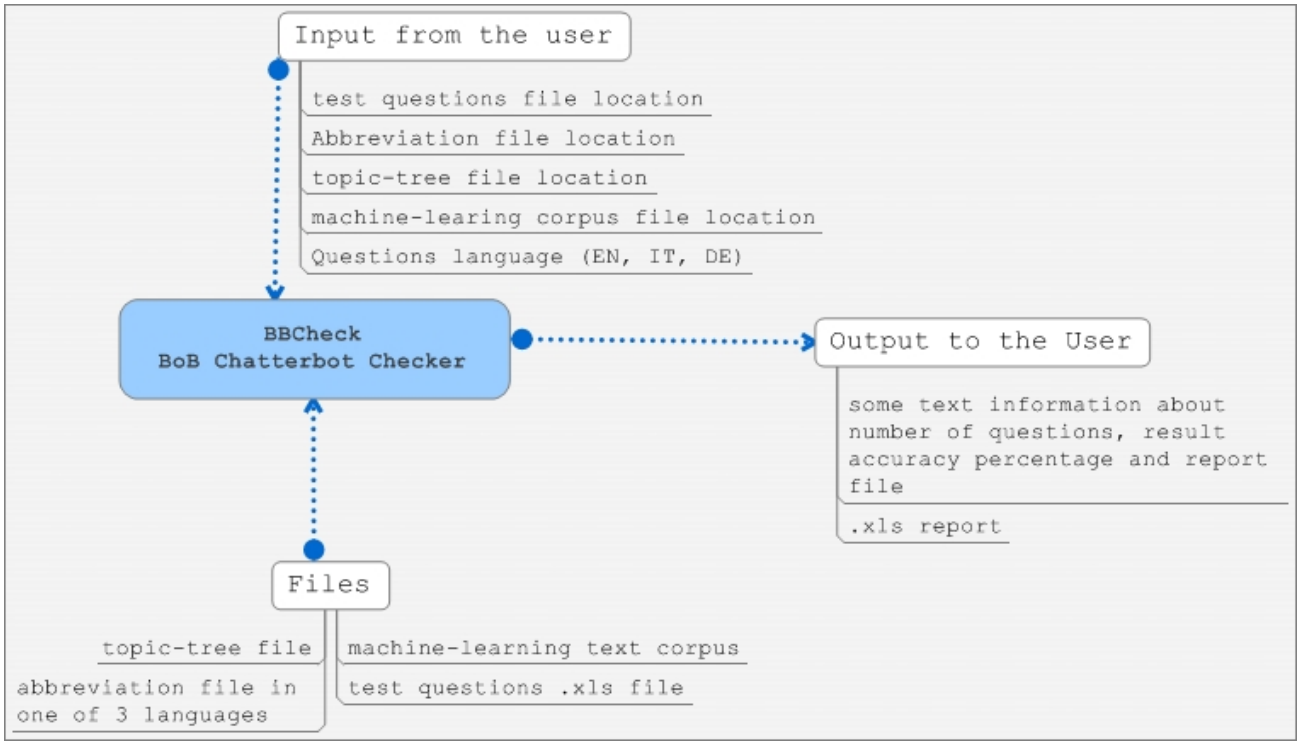


Testing progress



Test results

5 Data Flow



6 Data organization

6.1 Extended regular expressions structure

Extended regular expression syntax is described in `Bob.g` file of chatterbot project (package `it.unibz.lib.bob.check`). It differs from the “ordinal” Perl regular expressions by introducing Boolean operators:

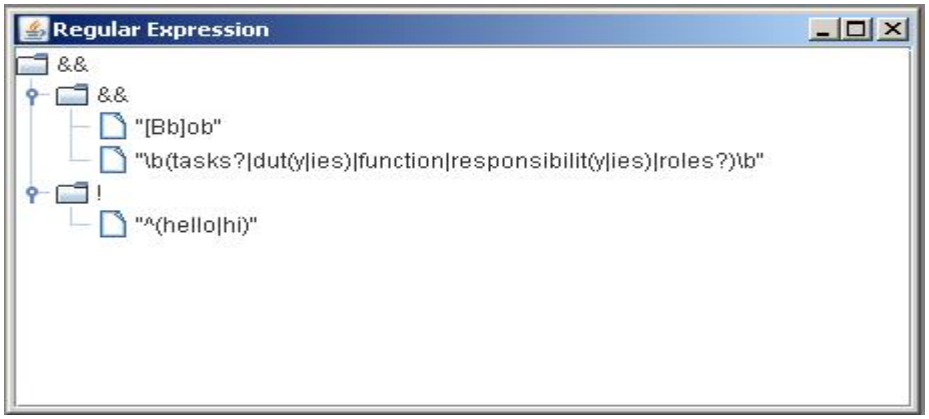
`OR : "||"; AND: "&&"; NOT: "!";` And their parenthesis () grouping.

```
("[Bb]ob" && "#AUFGABEN#") && ! "^(hello|hi)"
```

In this case will be matched all phrases that consist of two parts: first part should contain the words “Bob” or “bob” and the second one should match the “AUFGABEN” abbreviation (which looks like following “(tasks?|dut(y|ies)|function|responsibilit(y|ies)|roles?)"). One more condition: the matched phrase should not start from words “hello” or “hi”.

Order of parts is irrelevant, so both of “tasks bob?” and “bob tasks?” will be matched.

The whole parsing tree structure of an extended regular expression could look like this:



Regular expression tree

6.2 Abbreviation files structure

It is a text file with a list of regular expression’s abbreviations (one per row). Structure of abbreviation is following: ABBREVIATION_NAME = “\b”+ +REGULAR_EXPRESSION+”\b”, where “\b” a word boundary symbol used in RegEx. Example (for English version):

```
WARUM = \b(why|what for|wherefore|for (what|which) reasons?)\b
```

In this case expressions that are matching this abbreviation will be: “why”, “what fore”, “wherefore”, “for what reasons?”, “for which reasons?”

6.3 Topic-tree file structure

Topic-tree is chatterbot’s “knowledge base” represented as an xml file with a particular structure. Each topic within a tree matches some questions with predefined reply and has several **attributes**:

- **`active`:** if the topic is active in current database.
- **`isLocal`:** if the topic is “local”. The idea behind "local" topics is that they can be used by librarians to encode so-called "context-dependent follow-up questions". Example:

  ```
  User Question 1:  Do you offer guided tours?
  System answer 1: We organize guided tours every Wednesday.
  User Question 2:  How do I sign up?
  ```

  For the system to “understand” follow-up questions, it does need to know what the dialogue “was about” in the previous answer. Bob’s search algorithm keeps this kind of information by starting its search for a matching question pattern for user question 2 at that node in the topictree where system answer 1 was found. What is required for the above example to work is a question pattern with the "isLocal" attribute set to "true", being a sister node of the node that contained system answer 1. Topics with "isLocal==true" contain question patterns that are overly general, so that they can match under-specified user questions as user question 2 in the above example. Such question patterns must not be included in the normal search for matching question patterns that spans the whole topictree, or they would be matched in many wrong situations.
- **`isSubDialogue`**: topic is a sub dialogue, i.e. matching the user’s response to the system question.
- **`isSystemInitiative`**(not supported yet): Marks up the system initiative topics, which could be returned to the user for the questions that do not find any match. The idea is that instead of a generic message such as "Sorry, I did not understand", BoB could respond with a message which is more relevant to the topic which was talked about in the previous user question and system answer, such as:

"I did not understand; are you looking for more information about the OPAC?"

Currently the BoB algorithm does not use this feature, and topics with "isSystemInitiative==true" were kept in the topictree only to have them in case we support this feature one day.

- **name**: unique topic's name that identifies it within the tree.

Topic elements:

- **<regex>** regular expressions in three languages (might contain abbreviations, described in their abbreviation files). RegEx element's structure looks like this:

```
<regex>
        <languages>
                <DE>(("^ *$")||("(#JA#|#WEITER#|#FRAGE_WAS# (muss|soll) #ICH# .*machen)"))</DE>
                <EN>(("^ *$")||("(#JA#|#WEITER#|#FRAGE_WAS# (must|should|do) #ICH# .*do)"))</EN>
                <IT>(("^ *$")||( "(#YES#|#GO_ON#|#QUESTION_WHAT# (devo) #I# .*fare)"))</IT>
        </languages>
</regex>
```

- **<answer>** contains predefined system's answer to a matched question (in three languages). Has a languages element with a structure equal to the one in RegEx.

```
<answer>
        <languages>
                <DE>[03]Ja, natuerlich. Alle Medien mit einer gruenen Etikette koennen ausgeliehen werden.</DE>
                <EN>[03]Yes of course. All items with a green sticker can be checked out.</EN>
                <IT>[03]Si, certo: si possono prendere a prestito tutti i libri con l'etichetta verde.</IT>
        </languages>
</answer>
```

- **<link>** element can substitute the "answer", if current topic matches the questions with an answer already specified in another topic. In this case, we could just specify the referent topic's name by giving a link.

```
<link>stella.10_AUSLEIHE.03_AUSWEIS.07a_Ausweis_WieBekommen</link>
```

- **<name-trace number>** element may appear in the scope of a topic element. Its purpose is to keep the history of all the names of the topic element that it has ever had. It is useful for the cases when the topic is needed to be moved to some other location in the topic-tree to

keep the implicit naming convention within the xml topic-tree file (topic names encode the location of a topic by using a path-like notation with dots to denote hierarchy). Name-trace with the highest number attribute is the current topic name.

```
<topic name="bob.CurrentTopicName">
...
        <name-trace number="1">bob.OldTopicName1</name-trace>
        <name-trace number="2">bob.OldTopicName2</name-trace>
        <name-trace number="3">bob.CurrentTopicName</name-trace>
...
</topic>
```

Name-trace within the topic

Tracing mechanism was implemented due to the fact that in the presence of the `link` elements in the topic-tree to provide topic-tree integrity. Bob consider not only the current name IDs of the topics, but also all the old ones, so each topic could be easily renamed and all the links to it will be maintained through the name-trace ID.

For the librarians not to take care of name-tracing there was implemented an SVN post-commit hook, that is executed whenever a topictree.xml file is checked into the SVN repository. This is a python script which must have been installed on the SVN server. This script basically do the following:

1. perform svn lock of the topictree.xml
2. run XSLT transformations on topictree.xml to add new name-trace elements to topic elements if needed (if some topic name was changed since the last commit of topictree.xml)
3. commit topictree.xml

6.4 Test-questions file structure

Test-questions file must be given in Microsoft excel format and have two columns on the first page: Query (question) and Topic_ID_OK (ID of the correct topic in corresponding topic-tree).

Query	topic_ID_OK
can i drink my water in the library	stella.00_FAQS.06_Getraenke_Stabi
can i eat my sandwich in the library	stella.00_FAQS.06_Getraenke_Stabi
i'm hungry. May i eat in the library	stella.00_FAQS.06_Getraenke_Stabi
is one allowed to eat and drink in the library	stella.00_FAQS.06_Getraenke_Stabi
may i bring something to drink	stella.00_FAQS.06_Getraenke_Stabi

Test-Questions XLS file

In this case, the answer for the monolingual English case (i.e. topic in the testing topic-tree) will be this one:

```
<topic active="true" isLocal="false" isSubDialogue="false" isSystemInitiative="false"
        name="stella.00_FAQS.06_Getraenke_Stabi">
        <regex>
                <languages>
                        <EN>("#KANN_ICH#|#MUSS#|#WIE#|#WELCH#|#V_KOENNEN#|#FRAGE_WAS#|forbidden|allowed")
                            &amp;&amp;
                            ("#SOFTDRINKS#|#ALK#|drink|\beat\b|food|beverages?")
                            &amp;&amp;
                            !"where"
                        </EN>
                </languages>
        </regex>
        <answer>
                <languages>
                        <EN>[06]Take only water with you if you visit the Library.
                                Other drinks or meals are to be left outside,
                                because they are definitely not the best friends of books.
                        </EN>
                </languages>
        </answer>
        <name-trace number="1">stella.00_FAQS.06_Getraenke_Stabi</name-trace>
</topic>
```

Topic in a Topic-Tree

6.5 Machine-learning corpus file

Used can indicate a machine-learning corpus (in one of three languages) to in order to use answer re-ranking mechanism. BBCheck assembly includes three prepared corpora of 1.000.000 lines of text taken from WAC (web as corpus).

6.6 Output format

Console output: Shows the information about answer re-ranker, answer correctness (percentage), specification of tested question status-classes (A, B, C, D, E, and F) with number of the representatives in each class and the name of .xls report that was generated.

```
********************
Answer Reranker disabled.
BoB's answer correctness: 60.0%
********************
A - !!! No match: 0
B - Correct match, single solution: 16
C - !!! Wrong match, single solution: 1
D - Correct match, many solutions: 5
E - Wrong match, many solutions, including correct one: 10
F - !!! Wrong match, many solutions, all wrong: 3
```

```
Report .xls generated: checkbob_en_091217-173055__00-01-03-05-06-07-10-11-13-17-22.xls
```

Report excel file:

Report file contains information about all test questions that were given: status-class (was the question matched, and how many solutions did the system produce), matched pattern and wrongly matched topic-tree IDs (if they are presented).

The structure of columns in **BBCheck** report is following:

- Test Question (the one from the input file)
- Correct ID (from the input file)
- Status (A, B, C, D, E or F): Positive B-statuses are highlighted green for better visual perception.
- Matched question pattern (extended RegEx syntax)
- Eight wrongly matched topic IDs with the maintenance of order, in which they were checked.

Test Question	Correct ID	Status
can i drink my water in the library	stella.00_FAQS.06_Getraenke_Stabi	E - wrong match, many solutions, including correct one
can i eat my sandwich in the library	stella.00_FAQS.06_Getraenke_Stabi	B - correct match, single solution
i'm hungry. May i eat in the library	stella.00_FAQS.06_Getraenke_Stabi	B - correct match, single solution
is one allowed to eat and drink in the library	stella.00_FAQS.06_Getraenke_Stabi	B - correct match, single solution
may i bring something to drink	stella.00_FAQS.06_Getraenke_Stabi	B - correct match, single solution

XLS Report

7 Software Licence

In order to provide maximum flexibility as to how the code can be used by customers we distribute the project to the research community under the LGPL (GNU Lesser General Public License) v3 open source licence (http://www.gnu.org/licenses/lgpl-3.0.html). Bob's source code is available for downloading on google code site where other researchers can contribute their changes and improvements: http://code.google.com/p/chatterbot-bob/

Note that we distinguish chatterbot source code, which is published under the LGPL licence via the given link from Bolzano-specific hand-crafted parts of Bob (such as: topic-tree xml file, abbreviation files, BoB's face images). These parts of the project are the property of the "Library of the Free University of Bozen-Bolzano" and must not be passed on to third parties without it's consent.

Why do we use the LGPL:

The Chatterbot project is licensed to everyone under the LGPL. The official name of the LGPL is “GNU Lesser General Public License,” reflecting its heritage as a derivation of the “GNU General Public License,” the GPL. The LGPL is different from the GPL in important ways. Since we don’t license the Chatterbot project under the GPL, we won’t bother explaining those differences or describing the GPL. Only the LGPL license applies to the Chatterbot project software. We use the LGPL for the Chatterbot project because it promotes software freedom without affecting the proprietary software that sits alongside and on top of the Chatterbot project.

The LGPL is an open source project licence:

1. The LGPL allows you to do the following things with the Chatterbot project software: use the Chatterbot project software as a component of your business
   applications in any way you wish, including linking to the Chatterbot project from your own or other proprietary software.
2. Make unlimited copies of the Chatterbot project software without payment of royalties or license fees.
3. Distribute copies of the Chatterbot project software, although it will be much easier to refer anyone who wants copies directly to our website, which we will publish on http://code.google.com/p/chatterbot-bob/
4. Make changes to the Chatterbot project if you need to do so for use within your own company. The LGPL license does not require you to share those internal changes with the rest of the community.
5. Distribute changed versions of the Chatterbot project to others, but if you distribute such changed versions you are required to share those changes with the rest of the community by publishing that changed source code under the LGPL.(see http://www.jboss.com/pdf/Why_We_Use_the_LGPL.pdf for further information)