TTCheck:

BoB topic-tree checker application

v. 001

1. Main Program Information

1.1 Program identification

1.1.1 Program Name

1.1.2 Program Version

1.2 Brief description of an application

1.2.1 Task of application

1.2.2 Main algorithm idea

2. Program Functions

3. Program Layout

3.1 Used types

3.2 Program structure: Modules, External libraries, Packages

3.3 Compiling and linking

4. Data flow

5. Data organization

5.1 Describe input data format: extended regex syntax , topic trees, abbreviation files.

5.2 Output data format

6. Software Licence

1 Main Program Information

1.1 Program identification

1.1.1 **TTCheck** (Topic-tree Checker) command line application.

1.1.2 Version 1.0

1.2 Brief description of an application

1.2.1 **TTCheck** is a command line tool that is able to check the *BoB* question-answer repository topic tree for correct XML and extended regular expressions syntax, using three language-dependant abbreviation files (English, German and Italian).

1.2.2 **TTCheck** takes BoB topic-tree from XML file, validates it with given as a parameter RNG schema, and then using previously loaded language-dependant abbreviation files checks regular expressions within it with the help of parser classes defined in **Chatterbot** project. Validations classes are generated automaticaly from ANTLR parser grammar (ANother Tool for Language Recognition).

2 Program functions

As **TTCheck** themselves is command line checking utility, so it gives the user only one function "Check", which takes as the parameters topic-tree and three abbreviation files and returns "True" in the case of positive validation an "False" in another case.

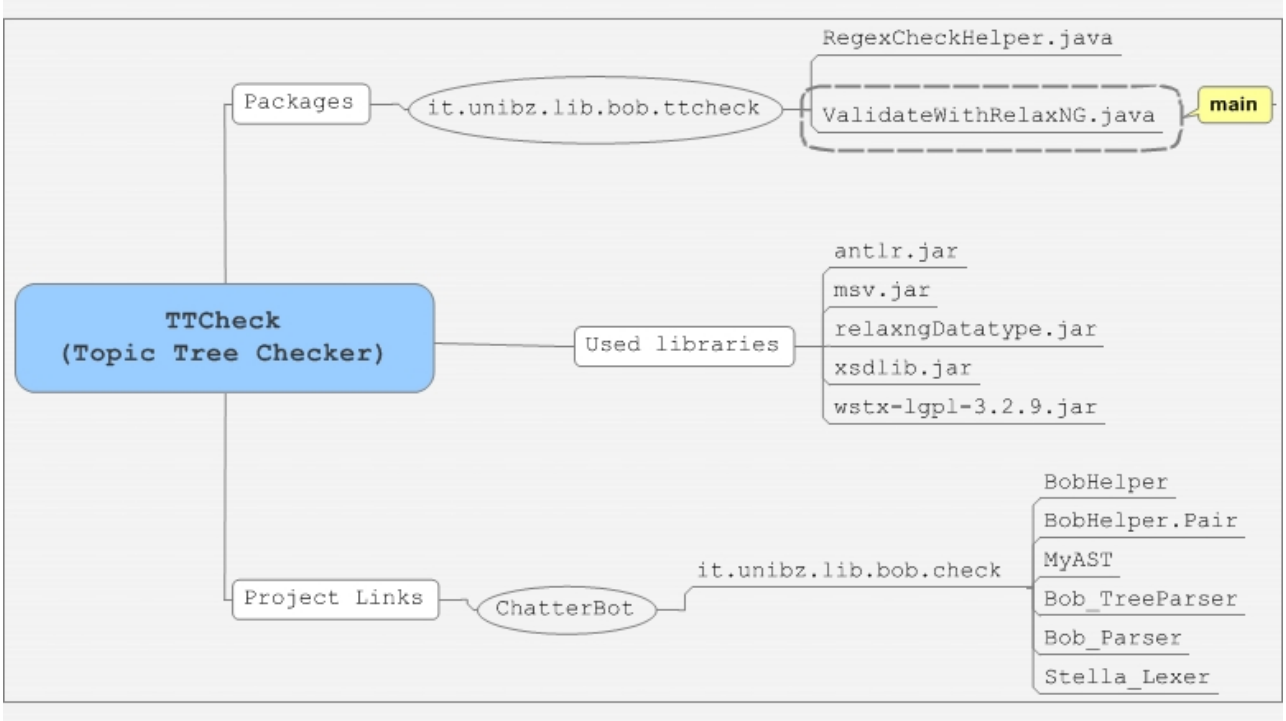
3 Program Layout

3.1 Among the types that are used in **TTCheck**, `BobHelper` is the most significant one. It's provided by **Chatterbot** project and implements the main logic of regular expressions checking by Boolean function `checkAllRegexesInMacroMap`.

`TopicTreeCheckerMain` class defines the main class, loads RNG schema and topic-tree files and performs XML validation against schema and then calls `RegexCheckHelper`, passing to it topic-tree and abbreviation files.

`RegexCheckHelper` is extending `DefaultHandler` and implementing all the methods for general SAX event handling.

3.2 Program structure (see attached bob_ttcheck_v002.jpeg)



3.2.1 Packages and classes

`it.unibz.lib.bob.ttcheck` contains following classes:

- `ValidateWithRelaxNG.java` – main method for RelaxNG validation
- `RegexCheckHelper.java` – helper for checking regular expressions

3.2.2 External JAR libraries:

`antlr.jar`: extended Boolean regular expressions parser

`msv.jar`: Sun Multi-Schema XML Validator (MSV) is a Java technology tool to validate XML documents against several kinds of XML schemata. Used for RelaxNG validation.

`relaxngDatatype.jar`: Datatype library for MSV-based Relax NG validation

`xsdlib.jar`: XML Schema Datatype Library

`wstx-lgpl-3.2.9.jar`: Woodstox XML parser and validator

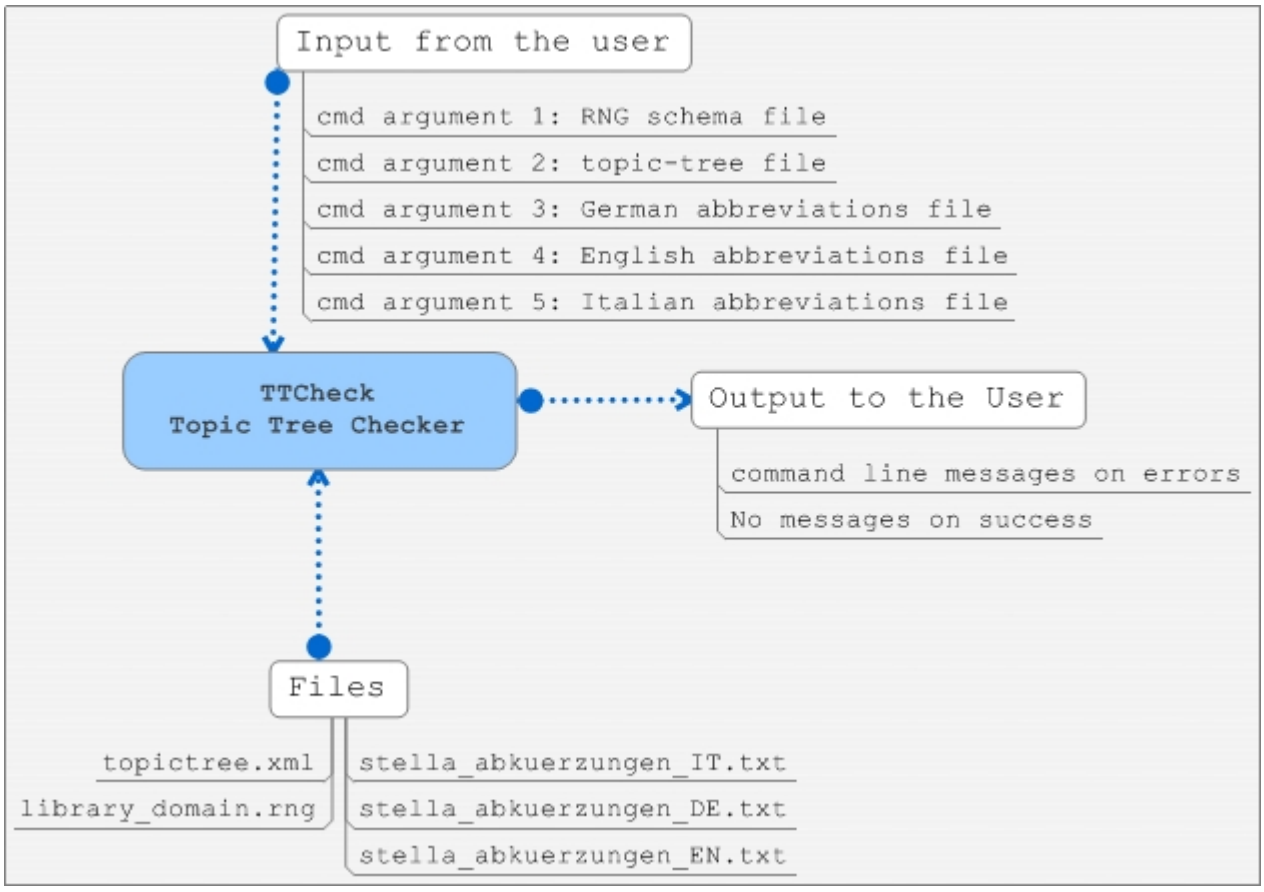
3.2.3 Project dependences:

`Chatterbot`*:* Matching regular expressions against question phrase.

3.3 Compiling and linking

Linking: export as a single runnable jar file.

4 Data Flow



Input: in case of zero command line arguments program tries to read RNG Schema, topic-tree and abbreviation files from the execution folder with the names of the files listed on the data flow diagram. Other possibility is to specify all 5 files manually as program command line arguments in the proper order (1 – RNG, 2 – topic-tree, 3 – German abbreviations, 4 – English abbr., 5 – Italian abbr.).

Output: Command line message in the case of parsing or validation errors, does give nothing on success validation.

5 Data organization

5.1 Topic-tree 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 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): System initiative topics, and which could be returned to the user for the questions which 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 3 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 3 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

5.2 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.

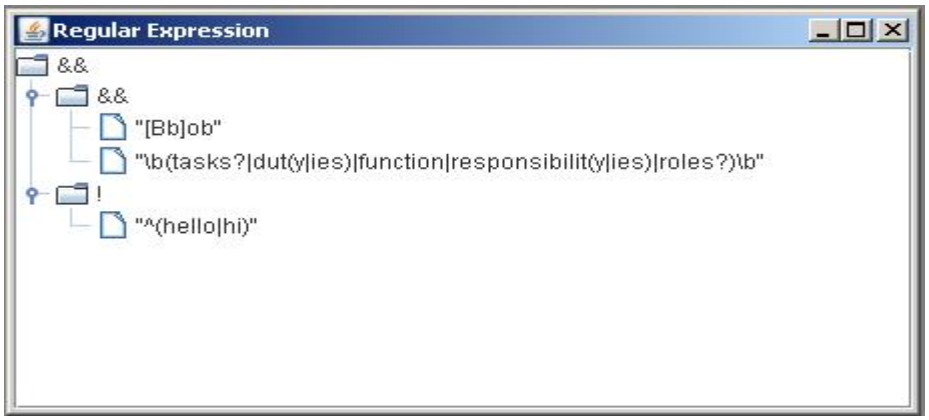
```
("[Bb]ob" && "#AUFGABEN#") && ! "^(hello|hi)"
```

In this case will be matched all phrases that consist of 2 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?)"). And 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:



5.3 Abbreviation files structure

It’s 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 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)