**BoB Chatterbot:**

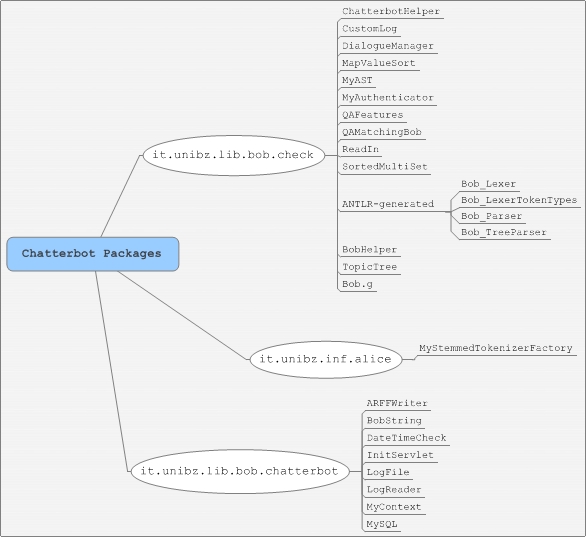
**A 3-lingual web-based**

**interactive question answering system**

**for the University Library**

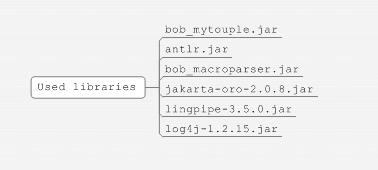
**v. 001**

1. **Main Program Information** 
   1. Program identification
      1. Program Name
      2. Program Version
   2. Brief description of an application
      1. Task of application
      2. Main algorithm idea
      3. Program Functions
2. **Program Structure**
   1. Program Organization
      1. Packages
      2. Classes
      3. External libraries
      4. Web Content
   2. Program Flow
      1. Program flow diagrams
      2. Data flow diagrams
      3. Dialogs description
3. **Data organization**
   1. Input data format description
      1. extended regex syntax
      2. topic trees
      3. abbreviation files
      4. machine learning corpora
   2. Database data format
4. **Application deployment**
   * 1. Setup databse for questions/answers history
     2. Setup knowledge base (topic-tree, abbreviation files, training corpora)
     3. Deployment
5. **Main Program Information**
   1. **Program identification**
      1. **Program Name:** *BoB Chatterbot* is a 3-lingual web-based interactive question answering system for the University Library.
      2. **Program Version:** 1.0
   2. **Brief description of an application**
      1. **Task of application:** *BoB Chatterbot* is an automatic dialogue system with a web interface that is able to provide interactive autonomous question-answering for the number of domain-based (regarded to the university library) questions in 3 languages (English, German and Italian).
      2. **Main algorithm idea:** *BoB Chatterbot* is a java web application with a fronted jsp that is also using JQuery from Google AJAX API library. BoB using as a knowlegde base an XML file with regular expression patterns for user’s questions matching. The XML knowledge base is organized into a topic-tree structure where each topic represents the single answer to a series of questions matched with one regular expression. Bob deals with an “extended” version of regular expression where ORO perl-style expressions are powered with the Boolean operations (ANTLR parser). For answers re-ranking BoB uses NLP-related technology for a machine-learning-based re-ranking, based on a text corpora (LingPipe library for TF/IDF string similarity, tokenizer, stemmer).
      3. **Program Functions:** *BoB Chatterbot*allows the user to address their questions in natural language (English, German or Italian) to the system via web interface and get predefined answers about the university library life.
6. **Program Structure**
   1. **Program Organization**
      1. **Packages:** The BoB *Chatterbot*Javaprojectislogicaly devided into 3 pachages:
         * it.unibz.lib.bob.chatterbot which contains proper chatterbot classes for output, database connection and connection with a web interface part.
         * it.unibz.lib.bob.check the pachage that is accesible from other bob projects and which contains all the logic for the question responding and toic-tree searching.
         * it.unibz.inf.alice the package with the Tokenizer class for answer re-ranker.

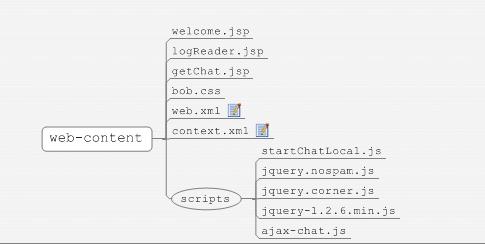
****

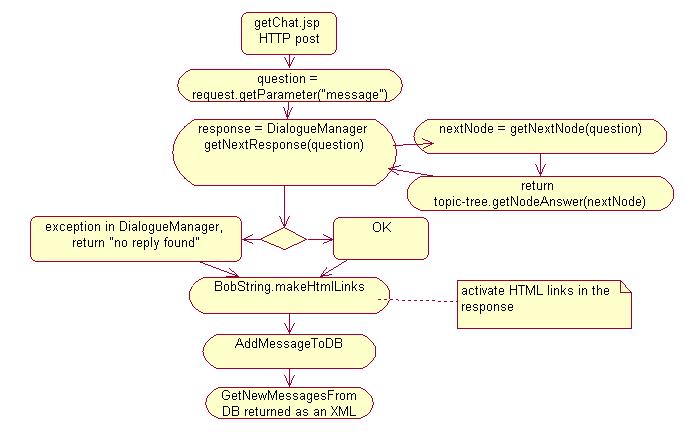
BoB Chatterbot Packages

* + 1. **Classes:**
       - MySteemedTokenizerFactory: implements a factory, that creates creates tokenizers for subsequences of character arrays. The tokenization is based on the following pipe, using existing LingPipe code: (IndoEuropeanTokenizerFactory 🡪 LowerCaseFilterTokenizer 🡪 EnglishStopListFilterTokenizer 🡪 PorterStemmerFilterTokenizer)
       - ARFFWriter: Workpiece. At some point should provide a nice API for writing correct ARFF files, for now works as a file string writer.
       - BoBString: class that is used for post-processing system answers before they are returned to the user. Add 3 types of HTML links to the system answers, for
         * terms highlighted with <option> tags (Replace the <option>-tagged words in a string with corresponding HTML links that trigger a javascript function which supplies the tagged words to BoB)
         * URLs that link to some URL from PHRASES in the topictree
         * normal URLs with regular HTML links
       - DateTimeCheck: provides method to check special time spans (morning, day, evening).
       - InitServlet: Java servlet that runs on a server side without a face and initialize DialogueManager and Logger parametrs according to the contex parametrs given in web.xml (URL of the topic-tree, URLs of the abbreviation files, URLs of training corpus files)
       - LogFile: represents logfile, contains saveDialogue method for logging, which is not used in the current version (instead MySQL database is used).
       - LogReader: used as a bean in a logReader.jsp
       - MyContext: defines context servlet listener of the application. The listener is notified about various events, such as application or session events. Has a method that is invoked when the Web Application is ready to service requests.
       - MySQL: provides access to the MySQL database for question/answer history.
       - ChatterbotHelper: ChatterbotHelper is another modified version of the BobHelper class, used only in BoB web application. Differs form BobHelper by the constructor parameters and the usage of static methods. It provides some helper methods to work with the topic-trees and the abbreviations files.
       - CustomLog: Custom version of log4j.Level class for logging.
       - DialogueManager one of the most important classes, wich implements the entire logic for question answering. Contains getNextNode(query, lang) method, which returns the node containing the next system response for the query.
       - MapValueSort: Class for sorting maps of the type <String, String>. Used in the DialogueManager to get a topicID sorted map <answer, topicID>; of all the answers that can be reached from the topictree's "normal" topic nodes.
       - MyAST: contains a modification abstract syntax tree, which is the output format for ANTLR parse. Modification adds to the basic set of parameters also line number information.
       - MyAuthenticator: The class MyAuthenticator represents an object that knows how to obtain authentication for a network connection by giving credentials. Used by calling setDefault method of Authentificator for the class to be registered.
       - QAFeatures: Representation of the set of features that are used for answers re-ranking.
       - QAMatchingBob: Class that is implementing answers re-ranking mechanizm, able to calculate total score for an answer candidate, represented by feature values.
       - ReadIn: Reads in data of various types from (stdin, file, URL), and than converts it to String.
       - SortedMultiSet: interface that allows multiple "equivalent" objects to be stored. Used in QAMatchingBob to sort re-ranked answer candidate quadruples.
       - BobHelper Helper class created in order to aggregate sequential calls to Lexer/Parser/TreeParser classes and working with abbreviations using macroparserclasses.
       - TopicTree: provides most of the tree searching logic, contains an instance of the XML topic-tree document.
       - Bob.g:The description of an ANTLR grammar. The source Lexer, Parser and TreeParser are generated from.
       - Bob\_Lexer.java:Automatically generated by ANTLR. It scans the extended regular expression string for the relevant tokens.
       - Bob\_Parser.java: Automatically generated from Bob.g by ANTLR. It builds a parsing tree out of tokens, extracted by Bob\_Lexer from regular expression string.
       - Bob\_TreeParser.java is also generated from Bob.g. It evaluates whether some user question are matched by the tree that was built by the parser. Algorithm works as following: it applies the required boolean logic and executs calls to regex.match() for the atomic (non-extended) regular expression patterns with the user question.
    2. **Used Libraries**

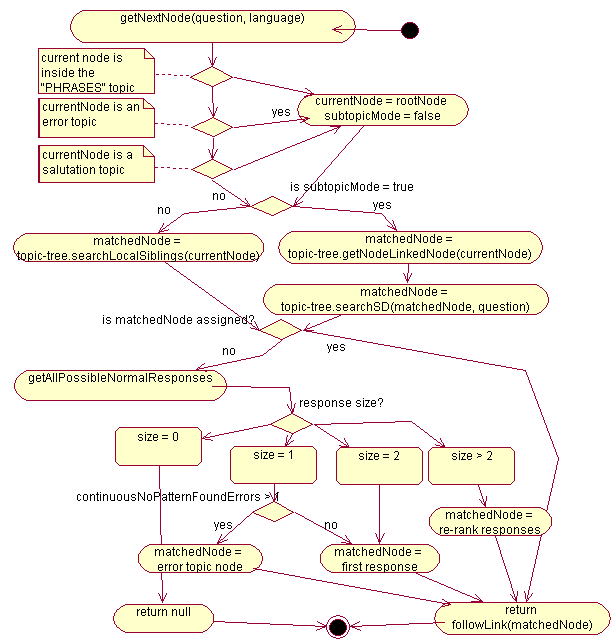
****

* + - * bob\_mytuple.jar: mallardsoft open source library, that presents classes for working with ordered multi-value collections.
      * antlr.jar: extended Boolean regular expressions parser, regular expression tree (AST – Abstract Syntax Tree) classes, AST tree visualization frame.
      * bob\_macroparser.jar Unifies the classes from it.unibz.lib.macroparser used for parsing regular expression macros.
      * Jakarta-oro-2.0.8**:** Provides pearl-syntax regular expressions matching engine.
      * Lingpipe-3.5.0.jar: LingPipe library for TF/IDF string similarity, tokenizer, stemmer. All lingpipe classes are used in the answers re-ranker only.
      * Log4j-1.2.15.jar: standart java logger.
    1. **Web-Content**

****

* + - * getChat.jsp: page that contains nothing but main logic of the chatterbot and calls to the java classes. It is called from ajax-chat script using HTTP post of JQuery returns an string with an XML-document Uses BobString class from the chatterbot package, which process system answers before they return to the user. Add 3 types of HTML links to the system answers, for a) Terms highlighted with <option> tags b) URLs that link to some URL from PHRASES in the topic-tree, c) Normal URLS. Uses beans
        + logFile bean from it.unibz.lib.bob.chatterbot.LogFile class for logging
        + db bean from it.unibz.lib.bob.chatterbot.MySQL from database manipulations
        + dm bean from it.unibz.lib.bob.check.DialogueManager to invoke question answering algorithm.
      * wellcoMe.jsp: main web-page of the project.
        + Contains the markup for all visual elements.
        + Uses Google AJAX API to load JQuery javascript library.
        + Uses ajax-chat.js local javascript file that provide chat functionality (it has startChat function and event messages handlers)
        + Launches "startChat" method from the ajax-chat.jsp or "StartChatLocal" from StartChatLocal.js on the base of the debug parameters.
      * web.xml: Web project deployment descriptor, that defines each servlet and JSP page within the web application and defines their mapping, contains context params (URL of the topic-tree, URLs of the abbreviation files, URLs of the training data corpora), listener class(MyContext.java), configures the web application's entry point (wellcome.jsp) and indicates the link to jdbc data-source object.
      * context.xml: Context container that sets up a DataSource for using the MySQL JDBC driver
      * ajax-chat.js: Javascript file that describes scripts for messages interaction and creates HTTP post request to the getchat.jsp page.
        + blockSubmit function: is linked to onsubmit form event of the chat form in welcome.jsp. Handles the user submitting text via enter key. Instead of submitting the form, send a new message to the server (via sendChatText) and return false.
        + sendChatText function: Add a message to the chat server. Linked to the onclick event of the submit question button from the welcome.jsp
        + StartChat function: gets called upon reset or on a page reload. It restores the GUI (text fields, emotion image), than using "$.post" AJAX JQuery method posts HTTP request to the getChat.jsp file with the language and last message parameters. Assigns to the HTTP post handleReceiveChat callback.
        + handleReceiveChat callback: Function that appends to the chatDiv div block in the bubble the text received in callback XML value $(chatDivId).append(dialogEntry(user, text)); where user is either the name of the user or 'BoB' string (for the system answers). Than it updates BoB's emotion image using updateBobEmotion function,and shows the evaluation link if the last message wasn't already evaluated (showEvaluationLink function).
  1. **Program flow**
     1. **Program flow diagrams**

Main response flow

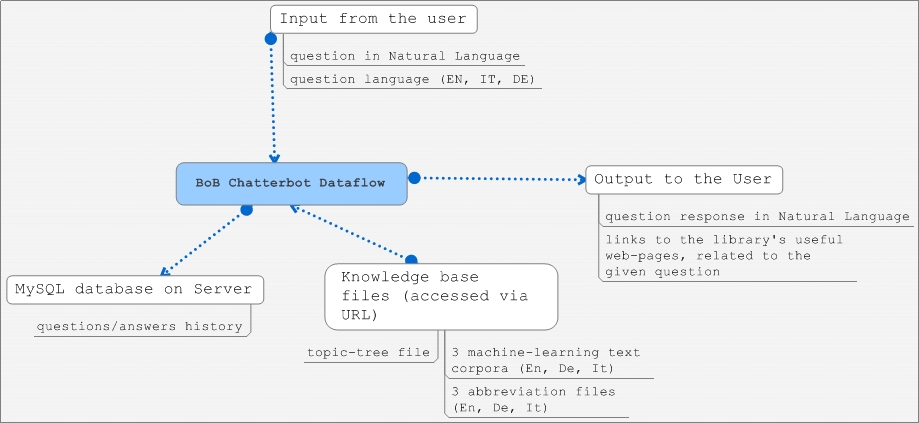
In the main response flow (see diagram): the POST request with the question phrase comes from the welcome.jsp page through ajax-chat script to the getChat.jsp page. Then the dialogueManager bean’s getNextNode function is called in order to get the topic node of the answer. The text of the response is extracted wrom the evalueted node using topic-tree getNodeAnswer function. If some exception is occurred during the execution of DialogueManager – then the system returns “No Answer found” string. At the end respons is being post-processed by the BobString class and added to the MySQL database. getChat page returns the response string in XML format to the callback, defined in the ajax-chat script file.

Response topic-node evaluation

Now let us go deeper into the question matching algorithm (see the response topic-node evaluation diagram), which is introduced by the DialogueManager class. So, first the DialogueManager looks if the current topic-node (used in the previous answer) is a particular one (PHRASES, SALUTATION or EROR). In this case current node is set to root and subDialogue mode flag is dropped. If afterwords the sistem is are still in subDialogue mode then the question is addressed to a topic-tree, which evalueates matchedNode with the searchSD and getNodeLinkedNode functions. If subDialogue mode is turned of – then toic-tree uses searchSiblings method to define matchedNode.

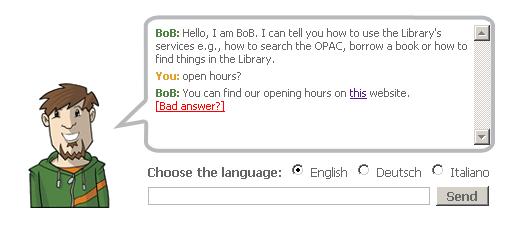
Response topic-node evaluation

If matchedNode is still not assigned then the DialogueManager tries to get all possible “normal” responses and depending on the result set size either returns the matchedNode or (in the case hen we do have multiple answers) launches re-ranker.

* + 1. **Data flow diagrams**

*BoB Chatterbot* takes input question from the user (see dataflow diagram) and using its own knowlegde base (represented by topic-tree file, 3 abbreviations files and 3 machine-learning corpora) returns the proper reply to the user. All question and answers are logged by the system and saved in the MySQL database, so one could leverage from learning these logs and improve the system.

* + 1. **Dialogs description**



*Bob’s* web interface is represented with one single page with a textbox for new questions and a “message bubble”, where all posed questions and system’s answers are displayed. In the message bubble user might also click the links given by *BoB* to visit related to the question pages or evaluate the bad answer.

1. **Data Organization**
   1. **Input data format description**
      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:

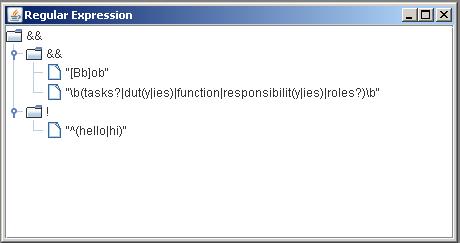
OR : "||"; AND: "&&"; NOT: "!"; And their parenthesis () grouping.

bob_qcheck_exregex.JPG

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:



Regular expression tree

* + 1. **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): bob_abbrev.JPG

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

* + 1. **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 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.
* **isSystemInitiative**(not supported yet):System initative 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:

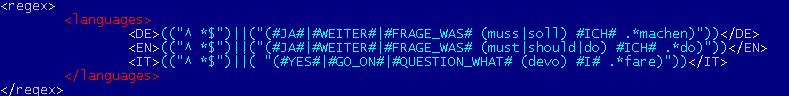
"I did not understand; are you looking for more information about the OPAC?"

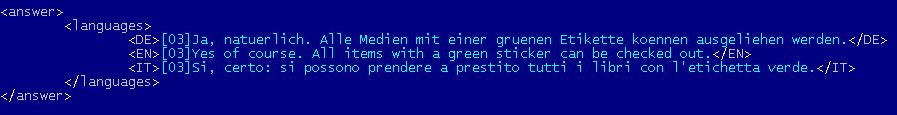
Currently the BoB algorithm does not use this feature, and topics with "isSystemInitiative==true" were kept in the topictree only to have them in case we support this feature one day.

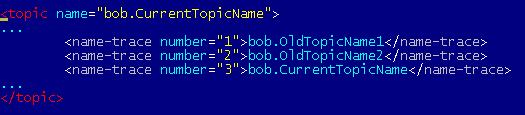
* **name**: uniquetopic’s name that identifies it within the tree.

**Topic elements:**

* **<regex>** regular expressions in 3 languages (might contain abbreviations, described in their abbreviation files). regex element’s structure looks like this:



* **<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.
* **bob_topictree_link.JPG<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.
* **<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.



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

**PHRASES topics**:

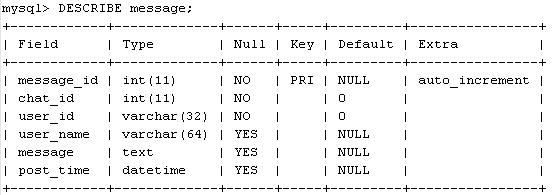
Some of the topic name attributes start with the string "PHRASES." As our convention, we have used such names for system answers consisting only of a URL (e.g., the library page with its opening times). The BoB search algorithm treats links to topics whose names start with "PHRASES." differently from other topics. Such links does NOT trigger sub-dialogue mode, as it would have occure for any other topic whose name starts with "stella."

* + 1. **Machine-learning corpus file**

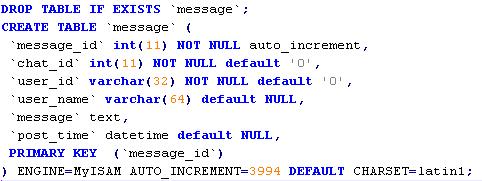
Used can indicate a machine-learning corpus (in one of 3 languages) to in order to use answer re-ranking mechanism. BBCheck assembly includes 3 prepared corpora of 1.000.000 lines of text taken from WAC (web as corpus).

* 1. **Database data format**

Bob’s database for messages history logging consists of one table “message” with the following columns: autoincremental message\_id, chat\_id, user\_id, user\_name, message and post\_time.



The table could be created using this kind of script:



1. **Application Deployment**
   1. **Setup databse for questions/answers history**

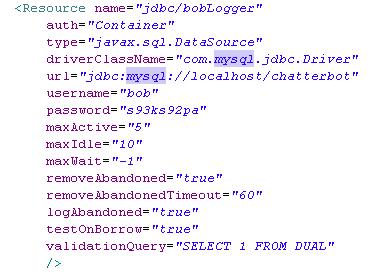
In order to setup database for loggin the history of questions/answer one should

1. create MySQL database named *chatterbot* (“CREATE DATABASE chatterbot;”)
2. Create the user bob (“CREATE USER ‘bob’;”)
3. Change password of the *bob* user (“update user set password=('s93ks92pa') where user='bob';”)
4. Create *message* table as it written in below:

DROP TABLE IF EXISTS `message`;  
CREATE TABLE `message` (  
 `message\_id` int(11) NOT NULL auto\_increment,  
 `chat\_id` int(11) NOT NULL default '0',  
 `user\_id` varchar(32) NOT NULL default '0',  
 `user\_name` varchar(64) default NULL,  
 `message` text,  
 `post\_time` datetime default NULL,  
 PRIMARY KEY  (`message\_id`)  
) ENGINE=MyISAM AUTO\_INCREMENT=3994 DEFAULT CHARSET=latin1;

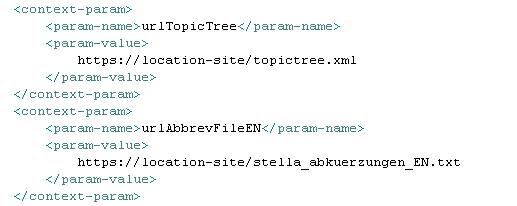
1. Give *bob* user permissions to access the database just created (“GRANT ALL PRIVILEGES ON chatterbot.\* TO 'bob';”).

All the DB connection parameters could be changed in the context.xml file, where jdbc/bobLogger resouce is defined.



* 1. **Setup Knowedge base**

Bob’s knowlegde base consists of 7 files (topic-tree, Enlglish abbreviations, German abbreviations, Italian abreviations, English text corpus for the re-ranker, German text corpus for the re-ranker and an Italian text corpus for the re-ranker). All this resources are listed in web.xml file of the project, so it is needed to put these files online and provide to the Bob a URLs in the configuration file.



* 1. **Deployment**

After all the previous steps have been done, BoB chatterbot cuold be deployed on the Tomcat server using eclips or manually (in this case all the used jar libraries also have to be added by hand)