Mathematical Expression Input Tool

A user friendly tool to type in LaTeX codes

W. M. Saroad
Department of Computer Science and Engineering, Faculty of Engineering, University of Moratuwa
Moratuwa, Sri Lanka
melanka.13@cse.mrt.ac.lk

***Abstract*— LaTeX is a high quality typesetting system, used to prepare scientific and technical documents. It has its own syntax and scripting style to produce mathematical and scientific text/symbols, on any LaTeX interpretable text editor. To use LaTeX, it is important for the user to strictly adhere to its syntax and hence be conversant with it, which is quite inconvenient. This project is focused on producing a user-friendly software tool, which would take in common mathematical text/symbols based on natural language as user input, then present a list of matching suggestions for that input and finally return the LaTeX code corresponding to the user choice out of the list of suggestions. The user-friendliness depends on how well both the natural input processing algorithms and the user input-to-LaTeX mapped database are tuned and optimized. Hence this project would always produce new updates and progress would be made as refinements and optimizations to the algorithms and database is inevitable.**

Keywords—LaTeX; strings; processing; algorithm; user; input; suggestions

I. INTRODUCTION

LaTeX is a high quality typesetting system, used to prepare scientific and technical documents. It has its own syntax and scripting style to produce mathematical and scientific text/symbols on any LaTeX interpretable text editor. To use LaTeX, it is important for the user to strictly adhere to its syntax and hence be conversant with it, which is quite inconvenient. Hence the focus of this project is to develop a user-friendly software solution to overcome this inconvenience encountered in using LaTeX.

As mentioned above the system provides means of a user-friendly interface between natural language input and the standard LaTeX styles. This would help users to write LaTeX codes without being familiar with LaTeX. In addition, this application would also serve as means of preview, by providing a set of suggestions for each user input, hence avoiding any errors at the time of LaTeX interpretation.

The project was then focused on producing a user-friendly software tool, which would take in common mathematical text/symbols based on natural language as user input, and then present a list of matching suggestions for that input and finally return the correct LaTeX code corresponding to the user choice out of the list of suggestions. The 'Google Input Tools' developed by the google in an open source project, which provides a user-friendly interface to type in non-English Unicode characters, served as a model-example for this software solution.

The remainder of this paper is organized as follows: First, Section II describes related work and literature. Next, Section III describes the basic architecture of the system including the system requirements and design, providing diagrams where necessary. The details of implementation of the system is described in Sections IV, along with a brief illustration of the algorithms. Section V then discusses about system testing and analysis. Finally, we conclude the paper in Section VI.

II. LITERATURE REVIEW

A. *Study on LaTeX*

"LaTeX is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents. LaTeX is available as free software." – LaTeX project site [1]. According to the LaTeX project site [1], it is a widely used documentation system, and had been releasing new updates since its first release by Leslie Lamport in 1994. The latest release was on the February 11, 2016.

A tutorial by JACEK POLEWCZAK [2] explains the fundamental and technological aspects of scientific/mathematical formatting systems including LaTeX. It describes the first steps to start LaTeX, installation of LaTeX interpreters and best practices in the use of LaTeX syntax.

Apart from formal mathematical and scientific documentation, LaTeX is widely used in online scientific/mathematical forums such as Art of Problem Solving [3]. Also, there are many user friendly online tools such as HostMath [4], that help users to easily build up and preview LaTeX codes.

B. *Study on similar existing systems*

1) Google Input Tools: Google Input Tools [5] is a tool that provides a user friendly interface and input method to type in Unicode characters for over 80 non-English languages including Sinhala and Tamil. It is installable on Windows environment, and functions as a background tool. It supports any Unicode recognizable text area including Notepad, Microsoft Office applications, online applications such as Facebook, Gmail, Yahoo etc. on any internet browser and frankly speaking on any known application. This tool is easily manipulated by the keyboard, hence this tool provides user friendly and efficient means of typing text allowing the user of involuntary use of the tool. This tool gets customized to the user, by remembering the user's corrections, and maintaining a custom dictionary according to experience with the user. Hence it is a smart tool that learns as it continues to work with the user. This tool was developed in an open source project initiated by the Google, and is available on GitHub [6]. The project is still in progress and expects to improve further with bug fixes and feature addition. The primary motivation and example for this proposed tool is the Google Input Tools.

2) Microsoft Equation 3.0: The Equation editor freely available on Microsoft office software applications, is a user-friendly Math symbol/expression inserter. This editor uses a markup language called MathML, as its syntax to format and present mathematical symbols. The user-friendliness of this editor is due to the availability of a GUI from which the user is free to choose his/her required forms of symbols/expression. The use of the cursor in such a GUI is inevitable, hence the speed, rhythm and consistency of typing declines with switching between keypad and mouse/touchpad. In this proposed system such difficulties are reduced immensely. The use of this proposed system, could entirely be done using the keyboard, hence continuing consistency while typing. In fact the user would use this proposed tool with much less effort and to a certain extent, involuntarily.

3) Texmaker: Is a system that incoperates tools that are needed to develop documents using LaTeX. It is a separate document preparation tool and cannot be used for petty LaTeX editing purposes on other LaTeX interpretable editors. In contrast, the proposed system functions as a background service tool with very low usage of resources and supports the user on minor LaTeX editing purposes on any LaTeX supported editor. The Texmaker requires the user to be familiar with LaTeX syntax. But the proposed system do not expect much user familiarity with LaTeX hence is much more user friendly and convenient than Texmaker. In fact this tool could even be used to type in LaTeX scripts on Texmaker, without being conversant with LaTeX scripting.

III. System Models

A. System Requirement

A user should be able to activate the tool whenever necessary through simple means, for example by pressing a couple of shortcut keys. Once it is activated, the user should have the option of deactivating it whenever desired at any stage, by a similarly simple method. Once activated, it should be able to catch the user keyboard text input on any desired application text area at real-time. Next, the application should process the input and propose a list of related suggestions. Finally, it should replace the user input text by the LaTeX code corresponding to the user selection from the suggestions list, on the same text area that the user used to type in the input.

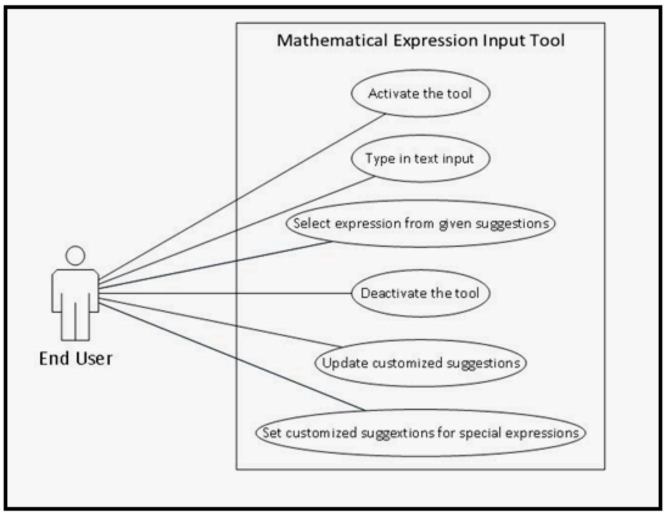
Mathematical Expression Input Tool
Activate the tool
Type in text input
Select expression from given suggestions
Deactivate the tool
Update customized suggestions
Set customized suggextions for special expressions
End User

Figure 2 Usecase Diagram

Smooth communication with the operating system, and simultaneous operation with any other major applications was a fundamental requirement in providing smooth and convenient user experience. Efficient execution of algorithms is required as it is a real-time application and a considerable amount of processing is required to be done. A light weight database is required that could be embedded within the application used to store simple input-to-LaTeX mappings and hence without giving special consideration to the capacity of the database.

B. System Design

The basic architecture consists of four main components. At the top most level or the user end is the User Interface. At middle level is the Input/Output Manager, responsible for coordinating and handling the system, and synchronizing the user interactions with the lower level components. At the lowest level is the Processer Engine and the Database, responsible for processing the user input by executing algorithms accordingly. These components communicate hierarchically.

The Processer Engine Package consists mainly of three classes. The DataAccess class is responsible for accessing the database and loading the data into the program. The TokenLibrary is responsible for efficient storage, manipulation, organization and presenting of necessary data (that was retrieved from the database) for processing. The Expression class is the most vital in terms of processing. The fundamental algorithm that understands and maps natural user input to well defined LaTeX script through efficient processing at real time is encapsulated in this class. Hence the Expression class can be considered the core of the application.

The IO_Center is a singleton class which acts as the coordinator between the user and the core of the application. It is responsible for handling input/output, directing input for processing accordingly, handling the GUI and at the same time maintaining synchronized communication between each component. The ExpressionList class is the one and only simple GUI, which displays the suggested list of outputs to a user for each user input.

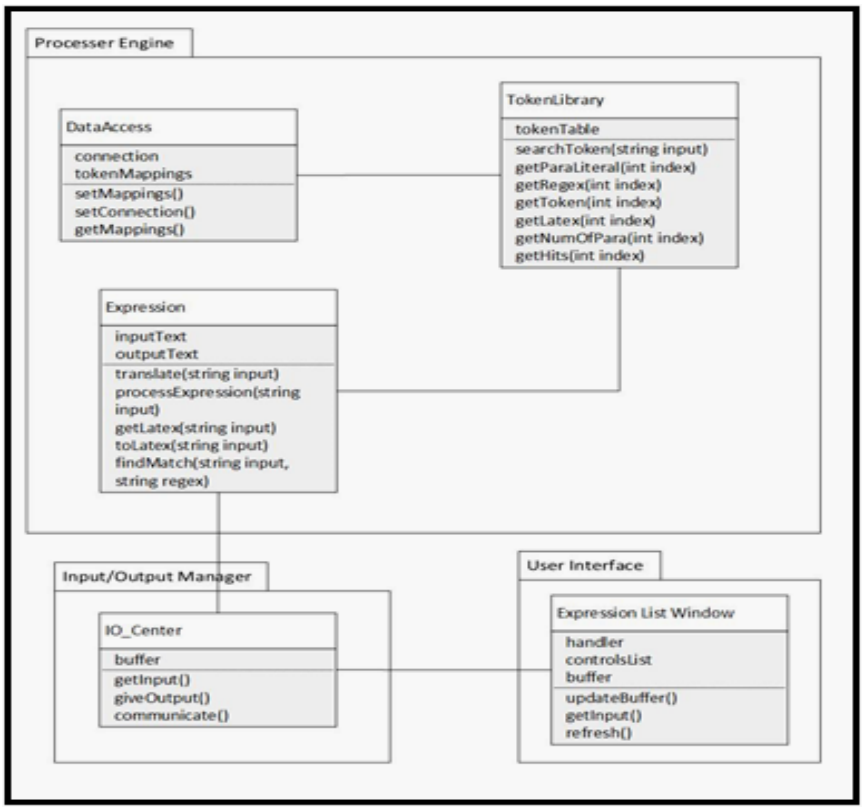

Figure 3 Class Diagram

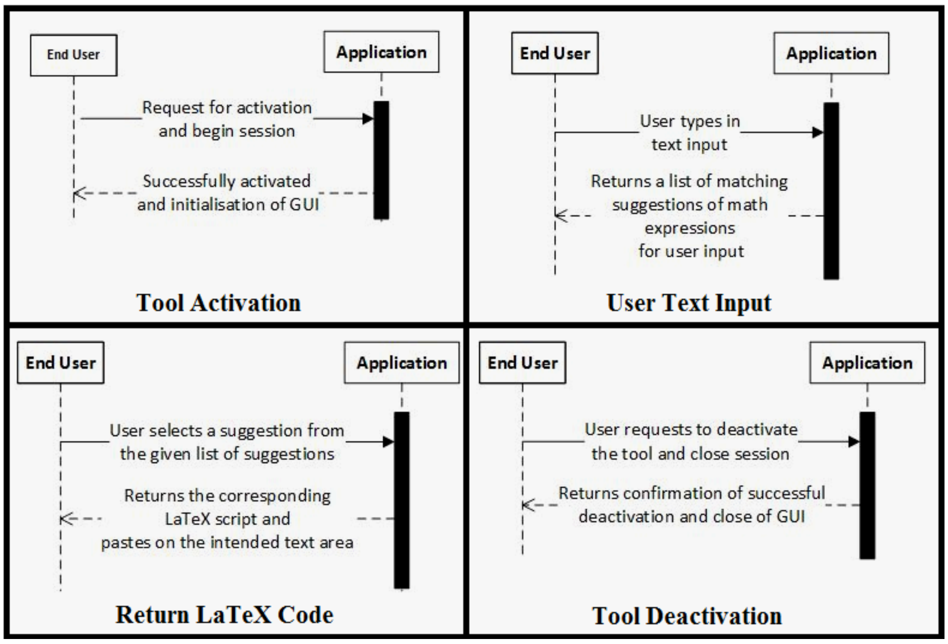
End User
Application
Request for activation and begin session
Successfully activated and initialisation of GUI
Tool Activation
End User
Application
User types in text input
Returns a list of matching suggestions of math expressions for user input
User Text Input
End User
Application
User selects a suggestion from the given list of suggestions
Returns the corresponding LaTeX script and pastes on the intended text area
Return LaTeX Code
End User
Application
User requests to deactivate the tool and close session
Returns confirmation of successful deactivation and close of GUI
Tool Deactivation

Figure 4 Basic Sequence Diagrams

Figure 2 shows logical class structure and Figure 3 illustrates the basic activity sequences of the application.

The database consists of only one table to store the mappings between user input strings, the corresponding LaTeX equivalent script segment and other information vital to process the user input string.

IV. System Implementation

A. *Implementation Procedure*

The main programming language used was C++. Its well-known efficient processing ability suited well for this purpose. In addition, the Qt4.8 library [7] which is a C++ based GUI library was used to implement the user interface. To establish multi-threading facilities, necessary classes were made to inherit from the Qt based QThread class [8].

To render LaTeX scripts as Qt supported images, the KLFBackend library [9] which is based on Qt4.8 was used. Qt supported images were those that were presented to the user on the GUI.

SqLite3 [10] was used as the Database Management System for the database. Its light weight property, ability to be embedded in the application, and the availability of C-libraries to conveniently handle SqLite3 made it highly suitable for this application.

AutoIt [11] which is a windows scripting language was used, to address the OS and other background applications. It was mainly involved in user input/output. The main application based on C++ and the AutoIt application exchange data via an established TCP connection based on a client-server model [12], in which the main C++ application acts as the server while the latter as the client.

The development cycle was divided into two stages, each of them being completely independent from the other. Development of the user input/output system and establishing the database connections were done at the first stage. Mainly, the development of the GUI, establishing multi-threading facilities, developing the AutoIt scripts and integrating it with the main code via TCP connection was done at this stage. Hence at this stage all the infrastructure needed for the proper functioning of the system was established.

Secondly, the core of the application, that is the processing algorithms were developed completely independent from the first stage. At this stage it was completely focused on optimized string manipulations, data manipulation and intelligent programming. Devising algorithms, coding, testing, refining was seen mainly at this stage. Each vital part of the main algorithm was broken down into several sub-parts, and organized under different functions, and each function tested independent from the others. A separate class with static methods was created to implement general string applications. Apart from that, almost all the crucial algorithms were concentrated within a single class.

At the end of the two independent stages, the two products were successfully integrated together. Finally finishing touches on the complete system was done, to produce the concluding result.

The database was populated with LaTeX script codes and mapped to expected user input. Expected/predicted common user input substrings were first listed at the brainstorming stage. For example, generally a user would type in "2/3" to indicate 2/3, so "/" could be considered as an expected substring. Next the corresponding LaTeX code segments were found by browsing through LaTeX documentation. As for the above example, the appropriate LaTeX code segment would be "\frac{2}{3}", where 2 and 3 are parameters to be evaluated by analyzing the entire input string.

There may be many variations of expected user inputs, hence it is a matter of tuning and expanding the database upon continuous usage by various users. It is evident that the nature of user inputs has no limit and could vary indefinitely with various probabilities. Hence the database created and deployed at the development stage is very primitive and could be considered as a test database with one table of around 200 records.

Understanding the user input and producing a list of predicted LaTeX equivalents was viewed as a translation from the domain of natural language to the domain of logical language (LaTeX). The main processing algorithm is basically composed of two main components namely translate1 and translate2.

```
translate1(string input){ // takes in a single string as parameter

  check_For_Exceptions(input); // check for marginal conditions such as empty string

  list<string> result = search_replace(input, 'operator'); // search and replace for tokens of type 'operator'

  result = search_replace(result, 'operator'); // search and replace for tokens of type 'letter'

  return result; // returns a partially processed list of possible LaTeX interpretations of input

  }
```

Figure 5 Pseudocode - translate1

```
translate2(string input){ // takes in a single string as parameter

  list<string> result; // declare the list to be returned

  list<string> tokens = search_Tokens(input, 'constructs'); // search any tokens of type 'constructs' in input

  result.add(input); // initialise the output result

  for each token in tokens:

    list<string> para = extract_Parameters(result, token); // extract parameters for token in each string in result

    list<string> latexList = insert_Parameters(para, token); // by inserting each parameter in token produces a list of latex segments corresponding to each element in result

    result = replace_By_Latex(latexList, result); //the result is refined by replacing the original substring by the corresponding string in latexList

  loop;

  return   result; //returns a processed list of possible LaTeX interpretations of input
 }
```

Figure 6 Pseudocode - translate2

To start with, the user input string would be subjected to a sequence of formatting for the sake of convenience in processing.

Next the formatted string would be fed as input to translate1. In which the string would be searched for substrings suspected to indicate tokens of type ‘letters’ and ‘operators’. Upon finding of such tokens, they would be replaced by the corresponding LaTeX equivalent(s). The pseudocode for translate1 is given in Figure 4.

Each string in the result returned from translate1 is then fed into the next main component that is the translate2. Each of the returned list of strings from each call of translate2 is then appended together to form the final output result of the basic algorithm.

The translate2 is basically focused on processing those tokens of type ‘constructs’. Tokens are searched and matched with corresponding LaTeX equivalents. For each of those tokens, the string is again analyzed to extract the required parameters and insert them in the standard LaTeX code. The LaTeX script segment thus prepared would then replace the corresponding natural expression segment in the original string. Pseudocode for translate2 is given in Figure 5.

B. *Main Interface*

The application consists of only a single simple user interface. This interface is responsible for displaying the list of suggestions for each user input. The images are displayed in a scrollable image list on a QMainWindow (a Qt Widget) [13]. The user is able to move up and down the list of images using the arrow keys. A particular suggestion can be selected by the user by selecting Enter key. The following figure shows the simple interface with a list of suggestions loaded, out of which three are currently visible and the selected item outlined in blue.

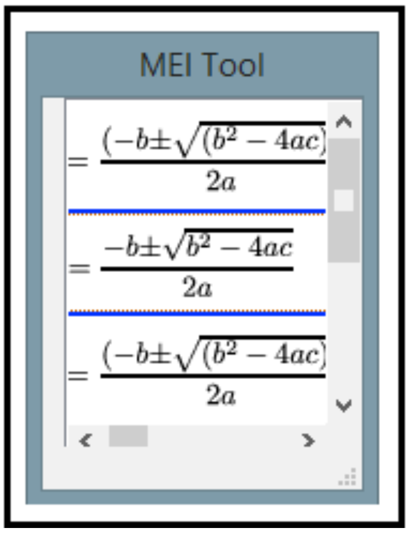
MEI Tool
$= \frac{(-b\pm\sqrt{(b^2-4ac)}}{2a}$
$= \frac{-b\pm\sqrt{b^2-4ac}}{2a}$
$= \frac{(-b\pm\sqrt{(b^2-4ac)}}{2a}$

Figure 7 User Interface

V. System Testing And analysis

Unit testing was done using a C++ unit testing framework module called CppUnit [14]. The most vital functions were unit tested. For each function, a set of test cases were tested. The test cases were selected to cover up almost all cases, and marginal test conditions such as empty strings and null objects.

Performance of the algorithms was tested by measuring the time taken to process a given input string. This test was performed repeatedly for varying lengths and complexities of inputs.

Resource (CPU and memory) usage was monitored through the Task Manager. This was monitored at different phases of the application such as the application activating phase, normal functioning phase, processing phase and the deactivating phase. Also this test was conducted under different background conditions (with and without major resource consuming applications running).

The unit tests ran with 100% success rate. None of them ended in failure. This was due to the careful and systematic procedures adopted at the development phase. Even during development informal tests were carried out momentarily and at random, by printing partial results on console. Careful scrutiny of code at development itself was the main fact behind the extraordinary success during unit testing.

As mentioned above the performance of algorithms were tested using time measurements, in milliseconds. The average time taken to process a given input was found to be 375ms. The maximum time recorded was 550ms and the minimum recorded was 125ms. These time recordings are most welcome. Half second processing time at worst case, for a considerable amount of string processing is quite appreciable. Use of C++ and optimized algorithms may have contributed to this fine performance ability of the application. Hence testing for performance was concluded with appreciable results.

Resource usage was at acceptable level- for a background tool. The tool would operate without much trouble even when high resource consuming applications were simultaneously running with it. During activation a little peak is observed due to several startup operations, such as loading the database and initializing objects and other resources within the application. During process phase no significant change in resource consumption was recorded. Also during other phases of operation the application did not show any considerable increase in resource consumption. Generally, the use of resources was at a minimized level. Hence the test for resource utilization concluded with appreciable results.

Failure testing was carried out by performing unexpected user activity and erroneous inputs. During these tests few failures were reported. The causes of those failures were identified and properly handled and remedial procedures were added to the final product. The tests were carried out again, and refinements were done based on those results. Also recovery after failures were at appreciable levels. Since this is a light weight tool, sudden failures are not expensive, which upon restart of the tool could be easily recovered. These aspects were tested out and turned out to be as expected.

VI. Conclusion

This paper presents the design and development of a user friendly tool to insert LaTeX script codes in any editable area. This tool enables users with little or no knowledge on LaTeX syntax to use LaTeX for documenting purposes with much convenience. While consuming little resources, it is able to coexist with other applications and operate as a user assistive background tool. Hence the tool presented in this paper has multiple advantages over conventional LaTeX editors that require extensive knowledge on LaTeX syntax and utilizes considerable amount of resources and are only focused on specific text editing environments. In conclusion, this paper provides proof of a successful application of Semantic Analysis with high performance and optimization.

In future, considerable improvements in performance are expected to be achieved through enhanced use of Natural Language Processing techniques and Semantic Analysis. This tool can be extended to support MathML in Word processing software as well. The Tralics Algorithm proposed by Jose Grimm [15] that converts LaTeX to MathML hints at such successful extensions.

References

[1] "LaTeX – A document preparation system", Latex-project.org, 2016. [Online]. Available: https://www.latex-project.org/. [Accessed: 19- Jun- 2016]

[2] J. POLEWCZAK, LATEX, MATHML, AND TEX4HT: TOOLS FOR CREATING ACCESSIBLE DOCUMENTS (A BRIEF TUTORIAL), 1st ed. Northridge: CSUN [Online]. Available: http://www.csun.edu/~hcmth008/mathml/acc_tutorial.pdf. [Accessed: 24- Jun- 2016]

[3] "Art of Problem Solving", Artofproblemsolving.com, 2016. [Online]. Available: http://www.artofproblemsolving.com/. [Accessed: 24- Jun- 2016]

[4] "HostMath - Online LaTeX formula editor and browser-based math equation editor", Hostmath.com, 2016. [Online]. Available: http://www.hostmath.com/. [Accessed: 19- Jun- 2016]

[5] "Google Input Tools", Google.com, 2016. [Online]. Available: http://www.google.com/inputtools/. [Accessed: 24- Jun- 2016]

[6] "googlei18n/google-input-tools", GitHub, 2016. [Online]. Available: https://github.com/googlei18n/google-input-tools. [Accessed: 19- Jun- 2016]

[7] "Qt 4.8", Doc.qt.io, 2016. [Online]. Available: http://doc.qt.io/qt-4.8/. [Accessed: 19- Jun- 2016]

[8] "QThread Class | Qt 4.8", Doc.qt.io, 2016. [Online]. Available: http://doc.qt.io/qt-4.8/qthread.html. [Accessed: 24- Jun- 2016]

[9] "KLFBackend Library: Main Page", Klatexformula.sourceforge.net, 2016. [Online]. Available: http://klatexformula.sourceforge.net/doc/apidoc/klfbackend/html/. [Accessed: 19- Jun- 2016]

[10] "SQLite Home Page", Sqlite.org, 2016. [Online]. Available: https://www.sqlite.org/. [Accessed: 24- Jun- 2016]

[11] "AutoIt - AutoIt", AutoIt, 2016. [Online]. Available: https://www.autoitscript.com/site/autoit/. [Accessed: 19- Jun- 2016]

[12] "Programming Windows TCP Sockets in C++ for the Beginner - CodeProject", Codeproject.com, 2007. [Online]. Available: http://www.codeproject.com/Articles/13071/Programming-Windows-TCP-Sockets-in-C-for-the-Begin. [Accessed: 24- Jun- 2016]

[13] "QMainWindow Class | Qt 4.8", Doc.qt.io, 2016. [Online]. Available: http://doc.qt.io/qt-4.8/qmainwindow.html. [Accessed: 24- Jun- 2016]

[14] "CppUnit - The Unit Testing Library", Cppunit.sourceforge.net, 2016. [Online]. Available: http://cppunit.sourceforge.net/doc/cvs/cppunit_cookbook.html. [Accessed: 24- Jun- 2016]

[15] J. Grimm, Tralics, a LATEX to XML Translator, 1st ed. 06902 Sophia Antipolis CEDEX, 2002 [Online]. Available: https://www.tug.org/TUGboat/tb24-3/grimm.pdf. [Accessed: 24- Jun- 2016]