TryMyLanguage: An Extensible, Web-based Software Development Environment

by

Amir Hossein Fouladi & Omid Zafari Khorvaj

LIU-IDA/LITH-EX-A--14/035--SE

2014-06-20
Final Thesis

TryMyLanguage: An Extensible, Web-based Software Development Environment

by

Amir Hossein Fouladi & Omid Zafari Khorvaj

LIU-IDA/LITH-EX-A--14/035--SE

2014-06-20

Supervisor: David Broman
Examiner: Peter Fritzson
Abstract

Introducing a new programming language can be a challenge to language designers and developers. This is usually due to difficulties related to setting up the environment and going through the documentation for people who want to start using and learning the language. In this thesis, we developed a web-based software system called TryMyLanguage which can be used to alleviate this issue by eliminating the need for users to install and configure related software tools such as compiler, interpreter, etc., on their computers. TryMyLanguage can be configured with different programming languages and it is not limited to a set of predefined languages. We evaluated the system by simulating a production environment and showing how it would behave in such an environment. At the end a case study is presented to show how the system can be used with a programming language named Modelyze as an example.
# Contents

1 Introduction ........................................ 1  
  1.1 Background ...................................... 1  
  1.2 Motivation ....................................... 2  
  1.3 Problem Formulation ............................ 2  
  1.4 Contributions .................................... 3  

2 User Perspective .................................. 4  
  2.1 Requirements .................................... 4  
    2.1.1 Use-cases .................................... 5  
    2.1.2 Functional Requirements .................... 6  
    2.1.3 Non-Functional Requirements ............... 6  

3 Design Considerations ........................... 8  
  3.1 System Overview .................................. 8  
  3.2 Design Space .................................... 8  
  3.3 Architecture .................................... 10  

4 Implementation .................................. 13  
  4.1 IDEView .......................................... 15  
  4.2 NewLangUI ....................................... 16  
  4.3 Code Editor ....................................... 16  
  4.4 Sandboxing ....................................... 17  
  4.5 Client-Server Communication ..................... 18  
  4.6 Plotting ......................................... 18  

5 Evaluation ....................................... 20  
  5.1 Method .......................................... 20  
  5.2 Results .......................................... 21  

6 Case Study ....................................... 26  

7 Related Work ..................................... 29  

8 Conclusion and Future Work ...................... 31
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 Code</td>
<td>33</td>
</tr>
<tr>
<td>A System Useability Scale Questionnaire</td>
<td>52</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

The traditional software development life cycle includes the following activities: Requirement Analysis, Design, Implementation, Verification and maintenance. Among those activities, implementation is the part where software engineers actually start programming and write the code. In this stage, software is born and goes through its life cycle which includes writing the program in a programming language, compile it to machine executable code and testing it to see if it is working as intended. Depending on type of the programming language, the program might be executed directly using an interpreter. At the end, if the program doesn’t behave correctly, one shall go through the process iteratively until achieving the intended result.

To do these tasks, programmers usually use various pieces of software to assist them. The bare minimum software usually includes a simple text editor in which you write the program, a standalone compiler or interpreter to compile and execute the program and a debugger to find the possible flaws. At the other end of the spectrum, a fully featured Integrated development Environment (IDE), with lots of capabilities such as syntax highlighting, real-time error reporting, code completion, etc., can be used. This set of tools along with the Operating System in which they function, comprises the environment in which the program is written and executed.

Traditionally this environment exists in a standalone architecture which means that the environment is set up and running on a workstation which is accessed by a single user. Nowadays, with more and more devices getting connected to the Internet, software is increasingly moving towards a more networked architecture in which software is available as a service on the web and it is accessible from any device which is connected to the Internet [7].

The aim of this work is to utilize this opportunity to create a simple and easy to use environment for introducing and teaching a new programming language which is accessible through the web and therefore making it easier for
1.2. MOTIVATION

CHAPTER 1. INTRODUCTION

someone who wants to try to learn a programming language. The software can also be used by researchers to introduce new programming languages to the world faster and with less effort.

1.2 Motivation

One of the great barriers for someone who wants to start programming is to prepare the environment. For a beginner programmer or someone who simply wants to try out a programming language, there is too much hassle to find right tools and set them up. First, a compiler/interpreter for the language should be installed. The problem is that, a compatible compiler might not even exist for a particular platform. For example, suppose that the programmer has access to a Linux machine but the compiler is only available for Windows operating system. In such a case, the only option would be to either install a new operating system from scratch or to use another machine which is not feasible most of the times. Another problem is that for widely-used programming languages like Java, there is an enormous amount of tools available which might simply confuse beginners to choose from and cause them to eventually end up choosing wrong ones.

Another important issue for someone who begins programming is to study the documentation of the language. Apart from some IDEs that might help you with this matter, programmers usually end up searching for the documentation on the Internet themselves. A lot of times, resources that they find are not reliable enough for beginners and may lead them on a wrong direction.

These limitations and drawbacks are also considered as barriers from the perspective of language designers and developers. Suppose that a new programming language is designed and its developers want to introduce it to the market. Pursuing other people to go through the cumbersome process of environment setup just to try out the language is hard and thus decreases the market penetration.

1.3 Problem Formulation

In this thesis, the following challenges shall be addressed:

- Eliminate the need for programmers to set up the environment and necessary tools to start programming.
- Make it easy for programmers to access the documentation of the language
- Make it easy for language designers and developers to introduce a new programming language and reach their audience more effectively
1.4 Contributions

In this thesis, we develop a web-based system for programming online. The system is extensible and configurable to be used with different languages and it is easy to configure and deploy for language designer and developers. It also makes it easy for programmers to access the language documentation. The programs are compiled and executed in an isolated and secured manner to ensure the security of the whole system as it is exposed on the web.
Chapter 2

User Perspective

2.1 Requirements

This section begins with describing a set of usage scenarios and their corresponding use-cases with the aim of clarifying how the system can be used in different situations. Next, all requirements for the system are listed and explained. Based on their characteristics, these requirements are divided into two categories of functional and non-functional requirements.
2.1.1 Use-cases

- Description of use-case *Try a new language*: A user hears about a new language in a conference and decides to try it. He reads the paper and goes to the language site. The user then starts to write a program in the code-editor. Optionally, the user can browse the documentation section to go through tutorial and choose executable sample codes. After the user is finished writing the program, he/she chooses to compile the program by clicking on run button. The result is shown in a console area. If there is any error in the code, the user is presented with information about the error which can be further used to find the location of the error in the source code.

- Description of use-case *Define a new language*: When a language de-
2.1. REQUIREMENTS CHAPTER 2. USER PERSPECTIVE

signer sets up the system for the first time, he/she would be directed to a view of the system for configuring the language alongside other system properties. This view is only accessible from the local machine, i.e., the same computer as the server, and not to remote users. The mandatory fields must be filled with proper values according to language specifications. There are help buttons with comments to clarify what is allowed in each field. The language designer can then use the system to demo the new language. The application can also be used for educational purposes that is, a teacher can use it to teach a desired programming language to students. To support this scenario, a tutorial for the language can be set up with the system.

2.1.2 Functional Requirements

Functional requirements describe the intended operational behavior of a system. The goal of each requirement is to define a part of functionality that the system supposed to perform. These requirements are listed below along with a brief description for each:

• The system shall act independently from the programming language it is used for. In other words, the system must be configurable with the language it is intended to be used for.

• The system shall provide users with a code editor which supports syntax highlighting. It means that the editor shall recognize different constructs of the language such as keywords and comments and color-code them with different colors to make the source code more readable.

• Compilation with error feedback: Users must be able to compile and run the program after they finish coding and see the results of compilation process. In case there is an error in the source code, there should be an easy way for the user to find the cause and the location of the error in the source code.

• 2-D plotting: System shall be able to draw 2-dimensional plots based on predefined data structures. Users should be able to write programs that produce data which is subsequently used by the plotting engine to draw the plot.

• Documentation of the language shall be available to the users in the form of a tutorial next to the code editor. The user shall be able to easily run the existing code samples in the documentation.

2.1.3 Non-Functional Requirements

As the name suggests, non-functional requirements are not directly representing different functions of the system. Rather, they are defined to insure
2.1. REQUIREMENTS

CHAPTER 2. USER PERSPECTIVE

a certain level of quality or to put some constrains on the system based on users’ needs. For example, these requirements can be related to Maintainability and manageability of the system, compatibility with other software and scalability and performance of the system. The list of non-functional requirements are given below:

- Licensing Requirements: The software shall be released under GNU General Public license version 3 GPLv3 [22]. This gives users the freedom to use the software for any purpose, study the source code and redistribute intact and modified versions of the software [23].

- Maintainability Requirements: To insure further development of the system, the code should be maintainable that is, it should be flexible enough to be easily adapted to future changes in requirements and the dynamic nature of the web.

- Compatibility Requirements: The system shall function independently of the platform it is running on to ensure the accessibility from different platforms such as Unix and Microsoft Windows.

- Scalability Requirements: The system should be able to scale proportional to the number of concurrent users. In other words, users should not perceive major performance degradation in periods of peak usage.

- Security Requirements: Users should not be able to compile and execute malicious code. In addition, users should not be allowed to compile and execute code which degrades the performance of the system. Further more, only privileged users should be able to configure the language and system properties.

- Manageability Requirements: Setting up and configuring the system shall be fairly easy for the language designer and the system administrator. No knowledge of internal parts of the system should be necessary to do the setup and configuration.
Chapter 3

Design Considerations

3.1 System Overview

TryMyLanguage TML is a software system with the aim of making programming with a new language a pleasant experience. To achieve this, TML is designed in a way that minimizes the need for users to do extra configuration and let them focus more on the task at hand which is developing software. Moreover, by using Web as the platform of delivery, the only requirement to use TML is to have access to the Internet and a modern web browser. TML officially supports Firefox version 3.5 and above, and all version of Google Chrome.

3.2 Design Space

The nature of TML requires a high level of interactivity which users usually experience while working with typical software development environments. For example, a text editor which supports syntax highlighting and a responsive console to display program’s output. This requirement makes it almost impossible to develop TML using traditional web technologies such as HTML and HTTP protocol. The nature of HTTP protocol requires the browser to refresh the page for every request the user makes [21]. This makes the application highly unresponsive to users’ actions. In addition, you need to manipulate components of the interface on the fly in order to achieve rich interactions and HTML is not designed for that purpose [21]. A number of technologies have been developed to remedy this problem. Web applications developed by these technologies are usually referred to as Rich Internet Applications (RIA). There are three different alternatives to develop RIAs. First, there is the plug-in base approach in which the application runs as a standalone application or embedded in the browser. An example for this approach is Adobe Flash and Microsoft Silverlight. This approach requires installation of extra software on the client machine. The second
method is to employ a set of technologies which mainly include JavaScript, HTML/XHTML, and CSS to develop web applications. These technologies are usually referred to as *Asynchronous JavaScript and XML (Ajax)*. Asynchronous communication between client and server, makes applications developed by Ajax techniques, resembling the look and feel of desktop applications. The third method for developing RIAs is to use technologies which are natively supported by the browser. An example for this is XML User Interface Language (XUL) developed by Mozilla Project [24]. The problem with this approach is that it is dependent on the browser and thus makes your application not portable among different browsers [21]. According to the requirements, users should be able to run TML without installing and configuring any extra software on their computers. In addition, dependency on a specific platform or software is not an option. Therefore among the solutions mentioned above, the Ajax technique have been chosen to fulfill these requirements. Although Ajax is supported by almost all modern browsers, different implementation of JavaScript makes it tricky to make the application behave in the same way on all of them. Several frameworks have been developed to make life easier for developers by bringing consistency among different browsers. Examples are Dojo, Prototype and Google Web Toolkit (GWT).

Another requirement for TML is that it should be independent of the programming language it is intended to be used with. One way to achieve this, is to use a compiler-compiler to automatically generate a compiler from the formal description of the programming language [25]. Although in theory, this approach would make TML a very sophisticated and easy to use case-tool, unfortunately these tools are not mature enough yet and they only can generate the parser part of the compiler. Another more practical solution is to assume that the compiler/interpreter of the language already exists and then configure TML to use it. However, for this to work, the compiler/interpreter should be already available for the platform that TML is running on. As mentioned several times before, one of the more important goals of TML is ease of use. To align more with that goal, TML does not provide any sort if authentication and users are able to just point their browsers to the TML server and start programming. However, this poses a security challenge that is, if users submit malicious code, they can potentially carry out different security attacks such as Distributed Denial of Service. To prevent this, the whole process of Compilation/Execution of the code is done in a controlled environment with limited access to system resources. This technique is called sandboxing. In this way the administrator of the system can define which resources are available to the programs that are running through TML. In addition, an execution timeout can be specified to avoid depleting resources by a single user.

In addition to functional requirement which are discussed above, there are non-functional requirements that should also be fulfilled by TML. The first requirement is that TML should be operational in face of growing
3.3 Architecture

A high-level view of the architecture of TML is depicted on figure 3.1 using UML component diagram [38]. As you can see in the diagram, TML consists of three main components. The front-end component consists of a text editor, a console which displays the output of the program, and a section displaying the documentation of the language. It also enables the system administrator to configure the system with a specific programming language.

Another component is the compile/execute service which is responsible for taking compile requests from the front-end and to provide it with the output of the program. In addition, it should manage TML configuration and in that, it should be able to read the configuration and do compile/execute tasks based on it. If the front-end requests that the output of the program should be plotted, the compile/execute service sends the output to the plot converter service and then redirects the formatted output to the font-end. This data is then used on the front-end to draw a chart.

The main module of the front-end is the code editor. According to the
### 3.3. ARCHITECTURE

#### CHAPTER 3. DESIGN CONSIDERATIONS

Figure 3.1: High-level Architecture of TML

![High-level Architecture of TML](image)

One of the security requirements is that users should not be able to run malicious code. There are several approaches one can take to meet this requirement. For example, the source code can be analyzed prior to compilation/execution to spot potential malicious parts. Several static source-code analysis tools [14] have been developed to automate this process. The problem with these tools is that they are very limited in how many security problems they can find and in most cases they need additional human review of the results. Moreover, there is the problem of false positives and false negatives that is, the tool might report a perfectly secure code as malicious or neglect an obvious security issue. The downside from TML’s prospective, is that the code analysis adds another layer of processing which increases complexity and degrades performance of the system. To keep things simple, instead of looking for vulnerabilities in the code, access to system resources gets limited at runtime. With this approach, the system tries to minimize the impact of possible attacks, rather than taking a preventive action. For example, by limiting the amount of time a program has for its execution, the impact of a malicious behavior can be greatly reduced. For this purpose, the administrator of the system can set a timeout for execution of programs run...
by TML so that if a program takes more time to execute than the specified timeout, it will be terminated by the system.

In order to limit access to other system resources, the program should be executed in an isolated environment. Two techniques have been discussed in [26] namely Sandbox and Virtualization. As the name suggests, a Sandbox is a contained and safe environment in which programs can be run. Access to system resources such as storage and I/O is restricted by the sandbox. This means that the programs cannot use all the features of the system and they are limited in what they can do. On the other hand, Virtualization provides a complete environment with all resources and services that a program might need to do its task. This environment is totally isolated from the hosting platform and can not affect it in any way. In a web-based application like TML, users have to have their own virtual environment in order for them not to be able to affect each others programs. Since each virtual machine takes a lot of resources form the host platform, assigning a virtual machine for each user is a wasteful choice and takes additional effort to manage effectively. For these reasons, Sandboxing has been chosen as the technique to improve the security of TML. Figure 3.2 is an execution view of the back-end showing the setting of sandbox relative to other parts of the system. As you can see in this figure, TML back-

![Figure 3.2: Sandbox](image)

end, perform both compilation and execution of the code in the Sandbox so any possible malicious code is isolated and can not affect other parts of the system. Obviously, there is a trade of between the security of the system and capability of the programs being run with it and sometime, one has to loosen the security of sandbox in order to allow more features.

TML back-end includes, compile/execute service as shown in figure 3.1. The whole back-end part runs on the host operating system on the server side which allows the centralized operation of sandbox.
Chapter 4

Implementation

The first decision that has to be made for implementing TML is the choice of platform and programming language. There are several factors affecting the decision ranging from the actual requirements to the personal preference of the developers. Since TML is a web-base application, the choice of technologies on the client side is straightforward. Being a standard system, the web is promised to be available for all platforms with a standard web browser and a connection to the Internet. As discussed in Chapter 2, Ajax is used on the client-side to deliver a rich user experience alongside other web technologies such as HTML and the JavaScript. On the server side, choices are much wider. Any platform and programming language that can process text and communicate using TCP/IP protocol can virtually be used as the web platform on the server side. In the case of this project, Java has been used for the following reasons.

First of all, Java is a platform independent language. It has been around since 1996 and has about 9 million active developers worldwide [33]. This makes Java a robust and reliable platform. Secondly there are a lot of open source third-party libraries written for Java that makes it usable for almost any application from embedded systems to enterprise applications. Last but not least, developers of TML has the most experience in Java and are quite comfortable around it. As mentioned above, to develop a web application one should deal with a lot of different technologies and programming languages such as HTML, CSS and JavaScript. This poses a challenge for developers that is, they should be familiar with all those languages to be able to deliver a web application. This particularly manifests itself in the difference between the language that is used to develop the client side, i.e., JavaScript, and the one that is used to develop server software. To remedy this problem one should either use JavaScript on the server side or use whatever language that is used on the server side for the client side too. Although JavaScript hasn’t been used traditionally to program server side application, platforms such as Node.js [28] uses it to do so and it’s been proved to be very
successfu in the industry in terms of delivering the expected robustness for a web or distributed application. However, even if products like Node.js are used to bring language uniformity across the client and the server, we are still left with CSS and HTML on the client side. In an attempt to unify all languages across the client and the server, Google introduced Google Web Toolkit or GWT. When developing with GWT, one uses only Java to develop software both on the client and the server. GWT works by translating the Java code to JavaScript, CSS and HTML to be rendered by the web browser. However, if your are not satisfied with the produced code, you still have to touch what is under the hood which is ultimately written in those languages. TML is using GWT to bring that uniformity to the development process. Although at some point it was necessary to deal with JavaScript, for the most part, GWT greatly reduced the time and effort needed to develop TML. Above all, GWT is and open source software which makes it an ideal candidate for TML licensing requirements.

Figure 4.1 is a box-and-arrow diagram of different components of the system and how they interact with each other. It shows the control flow of the system and what services each component provides to other components. The direction of arrows is from the provider to the consumer of the service. At the heart of the system, is the TMLService which is responsible for providing different services to the front-end. IDEView is the component through which users interact with TML. It comprises a code editor, a console and a section for showing the documentation. NewLangUI is used by
the administrator of the system to configure TML with a programming language. TMLService relies on the CompileService to compile and execute the programs. If the program involves drawing a plot, PlotDataConverter service is used to convert the data format of the program’s output to the format used by TML to draw charts. In other words, the CompileService provides the output of the program execution to the PlotDataConverter service and gives back the result of conversion to the TMLService to be consumed by IDEView subsequently. Both NewLangUI and IDEView are implemented using GWT.

4.1 IDEView

Figure 4.2 shows a screenshot of the IDEView in user’s browser. This is the main part of the user interface of TML and users spend most of the time on this page. The components of this view are fairly straightforward and anyone who has a little bit of experience with Integrated Development Environments or IDEs can recognize different parts of it. You can see the code editor at the center of the view. This is where the program is written. It shows line numbers on the left and the file name on top-left of the editor. On top of the editor, there are buttons for running the program and opening new files in a new tab. There is also an option to enable chart drawing feature. Right below the code editor is the Console area. This area shows the output of the program to be executed. If there is a syntax error in the program, the console shows the details of the error and the place it happened. If the users clicks on the line number which is shown in the console, the cursor
in the editor jumps to the line where the error has happened. If there is no error in the program, the console area simply shows the output of the execution of the program. TML is designed to run console-based programs and currently it is not possible to use it to develop programs with graphical user interfaces. Nevertheless, TML has the ability to draw graphical charts based on the output of the program. If the user chooses the option for drawing charts, the console area shows a chart instead of the text produced by the program. This is explained in more detail in section 4.6.

On the left side, there is an area for showing documentation. It can be used for example by teachers to show a tutorial of the language to the students or it can be used to provide the language reference. HTML is used to build the material for this section. There is also a feature to include code snippets in the documentation and automatically copy the code to the editor by clicking on a button. Detailed instructions on how to write the documentation is included with the TML software.

4.2 NewLangUI

This is the page in which the system administrator configures TML. As you can see in figure 4.3, it consists of two main parts. On the left side, one can set up the characteristics of the programming language to be used with TML. These characteristics, include: keywords of the language such as if, while, . . . , and how to comment out text. On the right side, compiler options such as file type, command line instruction for compiling and executing programs, and execution timeout is configured. It is also possible to define a regular expression for interpreting the output of the compiler in case of possible errors. In addition, there are a set of help buttons to assist the process of setup and configuration. When the user clicks on a help button, a small window pops up showing extra information about the particular item designated for that button.

4.3 Code Editor

Code editors are relatively complex pieces of software and when it comes to web the situation get even worse given the complexities you face such as cross-browser compatibility. Fortunately, there already exists a number of feature-rich sophisticated and stable code editors for web which are written in JavaScript. TML uses CodeMirror [2] mainly because it is stable, actively developed, has comprehensive documentation and distributed as open source software. CodeMirror has the notion of modes which is used to extend its functionalities to different programming languages. There are already tons of modes for different languages but the problem is that there is no way you can add a new language without writing some JavaScript code. To tackle this issue, we introduced a general mode that can be configured with
4.4. SANDBOXING

Since sandboxing is about limiting resources available to programs, it is highly dependent on the operating system on which the program runs. Being written in Java, TML can be theoretically run on any platform that supports Java. However for the sandboxing to work, there should be specific implementation for different operating systems. To make matters simpler, we assume that TML is deployed on a Linux machine for which there are already a number of sandboxing solutions available. Schreuders, Megill and Payne [36] did a comparison between three popular Unix-based sandboxing solutions namely SELinux, AppArmor and FBAC-LSM in terms of usability and security. While FBAC-LSM provides the best usability and ease-of use, AppArmor manages to keep the balance between security, ease-of use and stability. In addition AppArmor is available for Ubuntu which is the preferred distribution of Linux for deploying TML. AppArmor has a feature called learning mode which is used to generate security policies based on
4.5 Client-Server Communication

Since TML is a web application, HTTP would be the natural protocol for communication between client and the server. A simple scenario would be sending an HTTP request containing the source code that should be compiled and then receive the output of program execution as the HTTP response. This requires TML to halt and wait for the output before it can send in the response. This can be problematic in situations where the result of the program execution is not immediately available. For example, if the program waits for user input or if it prints the output bit by bit periodically. Since for each HTTP response there should be a corresponding HTTP request, it is not possible to send the output bit by bit using multiple responses. To push information to the client whenever it is needed, one can use a technique called Comet or Reverse Ajax. Bozdag, Mesbah and Deursen [8] did a study on different ways of implementing reverse Ajax. They did a comparison between push and pull communication patterns. According to their study, push has the advantage of high network performance but the downside is that it consumes a lot of system resources and therefore hard to scale. On the other hand, pull only works well when the pattern of communication is predictable. We employed a hybrid-approach to implement reverse Ajax in order to achieve balance between performance and scalability.

4.6 Plotting

This feature is implemented in order to make TML more useful for languages that do scientific simulation such as Modelyze [11]. Modelyze is a host language for embedding equation-based modeling languages [10]. In chapter 6, we show how plotting can be used with Modelyze. Another candidate for using this feature is Modelica language [5]. Although we have not tested Modelica with TML in this thesis, since TML can be used with different languages, it should be possible to do it. To implement the plotting feature, the following requirements should be considered. First we needed to draw plots on the client side and in the web browser. Second, it should support different chart types such as Line chart, Pie chart, etc. It should also support simple user interactions such as getting extra information by pointing to different parts of the chart. Finally to make development easier, it should be possible to integrate it with GWT. Harger and Crossno [27] did a comparison between Open Source Visualization Analytics toolkits that can be used for visualizing information by drawing charts. In their study, they
4.6. PLOTTING

CHAPTER 4. IMPLEMENTATION

compared different toolkits based on available features, and the development environment. Among the Wb-based toolkits, Google Visualization Toolkit has the best support for GWT and it satisfies all the other requirements for drawing charts for TML. Another challenge that we faced in implementing plotting in TML is the data structure that is used to deliver information to the plotting engine. Google Visualization Toolkits uses a JSON [18] to represent plot data. The problem is that, one might want to use a different data structure to represent plot data. For example, one might want to write programs in TML that produce plot data in columnar format like spreadsheet programs. In that case, the data should be converted to the format usable by Google Visualization Toolkit, otherwise TML shows and error indicating data format is invalid. PlotDataConverter module is used to solve this problem. It uses Java Service Provider Interface [37] to be able to extend the functionality of TML by adding Java modules. This makes it possible to add extra functionality without the need to manipulate the source code of TML. To be able to use this feature, TML provides a simple Java interface for data conversion. This interface is meant to be implemented by the system administrator in order for the plotting feature to work. The implementation is simply a String manipulation function which takes the output of programs and reformat it for Google Visualization Toolkit to use. The interface file includes all the documentation needed for Implementation.
Chapter 5

Evaluation

5.1 Method

Evaluation of TML has been done in three stages. In the first stage, we have arranged a walk-through evaluation based on IEEE 1028-2008 standard for software reviews and audits [3] with a team of four domain experts to assess the quality of TML. The walk-through objective is to evaluate TML and particularly the language setup phase in order to detect anomalies and improve the software quality. In the beginning of the evaluation session, the walk-through leader gives a brief presentation of TML followed by a general discussion about the application and tasks that the team is going to perform to ensure the walk-through is directed properly. Next, the team is asked to configure TML with a programming language of their choice within a reasonable amount of time. Each team member performs the walk-through individually to examine the application and find possible anomalies. The session continues with a discussion about issues that has been raised by the team members during installation and configuration. Afterwards, a detailed demonstration of application is performed by the walk-through leader. Finally, the walk-through leader gathers all the comments and suggestions made by the team for further analysis. Walk-through participants are chosen from master students of computer science in Linköping university. They can be considered as domain experts and it was relatively easy to reach them for the purpose of the research. Table 5.1 shows demographic information about the members.

<table>
<thead>
<tr>
<th>Number of Participants</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
<td>2 Male - 2 Female</td>
</tr>
<tr>
<td>Education</td>
<td>Master Students of Computer Science</td>
</tr>
<tr>
<td>Age Range</td>
<td>25-30</td>
</tr>
</tbody>
</table>

Table 5.1: Demographic data of walk-through participants
5.2 Results

The walk-through evaluation has resulted in comments and suggestion about various parts of the system, namely: configuration page (NewLangUI), the Readme file and the main page (IDEView). All the participants thought that items in the configuration page are ambiguous and the provided documentation is not enough to understand them. For instance, in the process of configuration, they thought that it is not clear how to specify keywords of the language. A possible solution which is suggested by one of the evaluators, is to provide example values for each field form widely known languages such as C or Java. Another problem is with the Readme file which is used to instruct users to install TML. It also contains instructions on how to set up customized HTML-based documentation and how to setup plotting feature. Evaluators suggested that instead of being a simple text file, it is better for the Readme file to contain step-by-step screenshots of the installation process. All the participants manged to install and configure TML with the language of their choice and run an example program. On average, it took approximately 30 minutes for participants to do the walk-through. Two of the participants chose C++ to do the evaluation and the other two did the evaluation using Java and Python.

When it comes to usability evaluation, SUS scores are not in any sense an absolute measure and they should be compared to scores gained by similar systems. This presents a great challenge in evaluating TML’s usability using SUS. In an attempt to interpret SUS scores in a more empirical manner, Bangor, Kortum and Miller analyzed SUS scores of various types of systems over a ten year period [6]. At the end, they presented a 7-point scale of the overall usability of the system based on the mean value of the scores gathered form all systems being evaluated. This includes Web-based systems similar to TML. Table 5.2 shows points of the scale and their corresponding values.

TML gained the average score of 62.5 which according the table 5.2 is somewhere between good and OK. This shows that although there is no severe problem with TML in terms of usability, there is certainly room for
5.2. RESULTS

Table 5.2: 7-point SUS scale with their corresponding mean values from 212 evaluated systems

<table>
<thead>
<tr>
<th>Rating</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Imaginable</td>
<td>100</td>
</tr>
<tr>
<td>Excellent</td>
<td>85.58</td>
</tr>
<tr>
<td>Good</td>
<td>72.75</td>
</tr>
<tr>
<td>OK</td>
<td>52.01</td>
</tr>
<tr>
<td>Awful</td>
<td>NA*</td>
</tr>
<tr>
<td>Worst Imaginable</td>
<td>39.17</td>
</tr>
</tbody>
</table>

*NA = not applicable

Figure 5.1: The diagram shows SUS scores corresponding to each participant
5.2. RESULTS  

CHAPTER 5. EVALUATION

Table 5.3: Test server system specification

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>8GB 667 MHz</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu Linux version 12.04 x64</td>
</tr>
<tr>
<td>Storage</td>
<td>Solid-state Drive</td>
</tr>
</tbody>
</table>

improvement. It is worth noting that the score can only be viewed as a preliminary measure of usability of the system and it is not to be considered as a holistic and complete way of usability evaluation. Figure 5.1 shows SUS scores obtained from each individual participant.

Next, is the performance evaluation. There are several approaches one can take to do performance measurement of a software system. In a web application, a typical indication of good performance is the number of concurrent users the system can handle. This depends on various factors such as hardware and software configuration of the server and network bandwidth and latency between clients and the server. In case of TML, the program being compiled and executed by the system, and performance characteristics of the programming language are added to these factors. Table 5.3 shows detailed specification of the server being used to do performance test.

In order to capture the actual performance characteristics of TML and minimize performance costs associated with programs being executed, the simplest form of a program is used to simulate users’ behavior. The program is a single line of code which prints out a line of text and it is written in Python. The total amount of time it takes for the python interpreter to start up, run the script and shutdown is 0.05 millisecond on the test server which is negligible comparing to response times associated with end-user experience of web applications which are usually measured in terms of seconds [31].

To simulate the situation of the system under load, a software tool called JMeter [29] is used. We defined a test scenario in JMeter for a typical user action which is running a program, i.e., the mentioned Python script. Then, JMeter runs the test scenario automatically with a defined number of virtual users concurrently and at the end shows the response times that are experienced by each user. Of course it is possible to run JMeter test on the same machine as the server but the number of virtual users would be limited to resources of the server machine. For example in our case, the test server has a quad-core CPU, so having more than three concurrent users would make the test results unrealistic. Fortunately, there are a number online services that provide real load testing for web applications. We used Blazemeter [1] which is a hundred percent JMeter compatible online load testing platform. One can simply upload JMeter test scripts and it runs the test on dedicated machines over the Internet. This makes test results close to what one might get with real users.

Figure 5.2 shows a diagram of response times, i.e., the amount of time it takes for the users to see the results on the browser window, when the system
5.2. RESULTS

CHAPTER 5. EVALUATION

Figure 5.2: The diagram shows response times based on the number of concurrent users

Figure 5.3: The diagram shows number of errors based on the number of concurrent users
is under load with 46 concurrent users. \textit{MAX\_Users} is simply the number of active threads reported by JMeter and each thread is representing a single user. The testing system gradually increases the number of concurrent users to capture the effect it has on response times. As you can see in the diagram, the response time is about 2 seconds with 9 concurrent users and it linearly grows to about 7 seconds by the time it reaches 21 users and then it decreases as more users are added. This behavior can be explained by looking at Figure 5.3. This diagram shows the number of errors produced by the server during the test comparing to the number of concurrent users. As you can see in the diagram, the server starts to produces errors when it reaches 21 concurrent users and that is exactly where the response time begins to go down. Blazemeter counts these errors as responses and since errors reach clients faster than correct results, the response time decreases. This continues until we reach 36 users with 4 of them receiving error. This patterns repeats again when the number of users exceeds 38.

The performance evaluation shows that the system is quite usable for small number of concurrent users, for example if TML is used for teaching purposes in a typical class room. However the performance starts to degrade when the system gets to have about 21 concurrent users. This can be acceptable or not, depending on performance requirements of a particular setup. For instance, if TML is to be used on the Internet for a Massively Open Online Course or MOOC which might have thousands of concurrent users, the performance characteristics are clearly not up to the task. It is worth noting that we have only used one server machine for testing TML and it is still possible to scale TML up, to support more concurrent users by utilizing multiple servers.
Chapter 6

Case Study

In this chapter, we go through configuration and running of TML with a particular programming language to demonstrate how the system works in practice. We used Modelyze [11] as the programming language to do the case study. In this case study, TML is configured with Modelyze and then it is used to execute an example program which produces a chart to also show how to use plotting feature.

Figure 6.1 shows TML configuration page with values corresponding to Modelyze language. On the left side, you can see all the syntactic characteristics of Modelyze such as its keywords, and the way comments are specified. On the left side, information related to how programs should be compiled and executed are specified. As you can see in the figure, the Compile Command box is empty. The reason for this is that, Modelyze compiler also takes care of execution of programs so you don’t have to specify separate compile and execution commands. Next, we have the regular expression for identifying the line number of possible errors which might occur during compilation of programs. This regular expression is used to process the output of the compiler to find the line number in the source code where an error occurs. @ is used as a place holder in the regular expression to refer to the line number. Note that you can also use <filename> and <suffix> as place holders to refer to the file name and the suffix of programs being compiled by TML. Another interesting point in this configuration is the timeout for compilation and execution of programs which is zero in our case. This means that there is no timely limitation on execution of programs. Of course this value can be changed to reflect the needs of a particular setup.

Now that we have configured TML, we can go to the home page and start using it to execute Modelyze programs. Figure 6.2, shows a sample program written in Modelyze language together with its output after execution. In this case, the sample program produces some chart data. If the user does not select “draw chart” option, TML simply prints out the text output of the program. On the other hand, if the users chooses to draw chart, TML
CHAPTER 6. CASE STUDY

Figure 6.1: TML configuration page filled with values corresponding to Modelyze

Figure 6.2: A program written in Modelyze language
Figure 6.3: A Modelyze program producing a chart

tries to interpret the output of the program as chart data. If the program produces valid data, the resulting chart will be shown to the user in the console area as depicted in figure 6.3
Chapter 7

Related Work

There are a lot of similar products to TML which can be used with variety of different programming languages. Some of them are tailored to a specific language and others support multiple languages. In this chapter a number of these systems are introduced along with a short description of their features. It is worth noting that none of the mentioned systems have the capability to be configured with a new programming language and we could not find a system with this feature to the date this work is published.

Cloud9 IDE [32] is an IDE that assists writing, running and debugging the code online. Cloud9 supports JavaScript, PHP, HTML, CSS, Java, Ruby and 23 other languages. It supports collaborative development which means that users can work on the same piece of code at the same time and they can also communicate using an integrated chat service. It also provides drag and drop, keyboard shortcuts, syntax highlighting and code completion. You can integrate your project with Git revision control and Bitbucket hosting service. Cloud9 IDE has two standard and premium versions and the standard version is free of charge.

Ling Wu introduced CEclipse [39] as an online IDE for programming in the cloud that supports Java. It is intended to address three main problems existed in similar platforms namely, Function implementation, Security guarantee, Advanced utilization.

F# is a type safe programming language which supports different type of programming paradigms including object-oriented and functional programming. It is ideal for analytical, data-rich and parallel component development [34]. F# is a part of .NET framework [16]. TryFSharp [35] is a learning environment with the aim of teaching fast and simple software development using F# programming language. You can either write your own code and run it or use the predefined executable samples in tutorial which are quite helpful to explore F# features. Compared to our TML, TryFSharp actually executes in the browser using Microsoft Silverlight [17] which is completely in contrast with our client-server architecture. F# is equipped with syntax
CHAPTER 7. RELATED WORK

highlighting and error reporting with line feedback.

WebIDE [20] is a web-based development environment for entry-level programmers. It is designed to make first few weeks of programming courses easier for students. The WebIDE objective is to provide a very simple install-free web development environment. It has a number of simple takes designed for beginner users in C and Java to get more familiar with programming.

AlgoWeb [19] is another platform which is developed to be used for learning introductory programming. The main goal of this tool is to encourage students not to abandon programming when facing difficulties in the beginning. It supports syntax highlighting for structured Portuguese which is a simpler version of programming languages like C.

ACE [4] is a standalone web-based code editor which means that it can be used offline too. ACE is designed to extend the capabilities and performance of current editors like Vim and Eclipse. It is a successor of Mozilla project previously named Bespin. It features Syntax highlighting for 40 languages, auto indentation, highlight matching parentheses, optional command line, multiple cursors and selections, line wrapping and code folding. According to their website, ACE can handle very large document with millions of lines of code.

Coderun Studio [15] is a powerful browser-based free IDE service for developing, debugging, compiling and deploying web application to the Cloud. Coderun supports application development in C#/.NET (ASP.NET, WCF, Silverlight), PHP, JavaScript, HTML and CSS. SQL Server 2005 and Amazon SimpleDB databases are supported. It features syntax highlighting and automatic code-completion in multiple languages.

Ideone [30] is an online developing, compiling and debugging platform which can be used to program with more than 60 programming languages. The code can be shared with other developers to assist collaborative software development. There are plenty of examples to help people writing a proper code. You can either execute the code to see the result or download the code for further modification and investigation.
Chapter 8

Conclusion and Future Work

In this thesis we introduced a simple web-based software development environment: TryMyLanguage or TML for short, which is configurable with different programming languages. The aim of the project is, on one hand, to provide language researchers a platform to easily introduce new programming languages to the world, and on the other hand, to help teachers to set up a web-based learning environment for students to learn and experiment with programming languages. TML provides an environment consisting of a text editor to write the program in, a console to show the output of the program execution and a panel to display documentation. TML is language agnostic, which means, one can set up the system to work with virtually any programming language. It is worth mentioning that TML does not generate compilers or interpreters for a language on the fly. Instead it assumes that the compiler/interpreter already installed on the hosting platform and then simply takes advantage of it to execute programs. We designed TML with a client/server architecture, which means the client sends out source code to the server and the server compiles/interprets the program and returns the result to the client for displaying to the user. In order to achieve a rich user experience, we used Rich Internet Application technologies such as Ajax to deliver a responsive user interface on the web, resembling the look and feel of a desktop application. The most important part of the UI is the text editor. We used CodeMirror as the text editor and extended it with a language independent mode for adding support for new languages without the need to manipulate its source code. We employed two techniques to meet security requirements. First we used sandboxing in order to avoid adverse effects of possibly malicious programs. The entire process of compilation/interpretation and execution of programs can be contained in a sandboxed environment which can be configured to limit available system resources. Second, it is possible to define a timeout for execution of programs.
CHAPTER 8. CONCLUSION AND FUTURE WORK

to prevent attacks based on resource exhaustion of the server. Finally, TML has a feature for drawing charts based on programs’ output. It is possible to extend TML to support different data formats for plotting engine.

Next, we evaluated TML from two different perspectives. First, we arranged a software walk-through based on IEEE 1028 standard for software audits and review. The walk-through was done by a group of experts on the subject to reflect the opinions of programming teachers and language designers. The result of the evaluation provided guidelines to make improvements in terms of functionality as well as usability of TML. Second, we did a usability evaluation to measure ease of use and learnability of TML.

At the end, a case study has been done to show how TML can be used in practice. A programming language named Modelyze is used to do the case study and to show various features of TML.

The following, is a list of recommendations for future works that can be done to improve and extend TML. The list is based on results of evaluation and the thoughts of authors of TML.

- Right now there is no way for users to use third-party libraries with their code. One way to solve this problem is perhaps to add the feature for uploading libraries to the server.
- The ability to save programming sessions for later use so users can reload their previous programs to continue working on them.
- When the user clicks on Run button to execute the program, only the source code that is in the current tab is sent to the server. It is better to be able to define project across multiple files.
- Although this requires great deal of complexity, it would be better from usability perspective to have auto-completion and auto-indentation of source code in the code editor.
- The software audit standard used in chapter 5, can be employed iteratively to improve the quality of TML in future releases.
Chapter 9

Code

This chapter includes some important parts of TML source code. It can be used as an overview of how TML is implemented and can be particularly useful for people who do not want to download the whole source code. The entire source code is available on following address: https://github.com/exjobbliu/TryMyLanguage

Listing 9.1: Generic CodeMirror mode that can be configured for different programming languages

```javascript
CodeMirror.defineMode("basemode", function(config, parserConfig) {
  var keywords = parserConfig.keywords,
  stringCh = parserConfig.stringCh,
  isOperatorChar = new RegExp(parserConfig.isOperatorChar),
  commentSingle = parserConfig.commentSingle,
  commentMStart = parserConfig.commentMStart,
  commentMEnd = parserConfig.commentMEnd,
  escapeCh = parserConfig.escapeCh;
  function tokenize(stream, state) {
    var ch = stream.next();
    var i = 0;
    for (i; i < stringCh.length; i++) {
      var strCh = stringCh[i];
      if (strCh == ch) {
        var escaped = false, next, end = false;
        while ((next = stream.next()) != null) {
          if (next == strCh && !escaped) {end = true; break;}
          escaped = !escaped && next == escapeCh;
        }
        return "string";
      }
    }
    if (/\d/.test(ch)) {
      stream.eatWhile(/\d+/);
    }
  }
});
```
return "number";
}

if((ch == commentSingle.charAt(0)) && (stream.match(commentSingle.substring(1), true, false)) && state.tokenize !="comment"){
    stream.skipToEnd();
    return "comment";
}

if((ch == commentMStart.charAt(0)) && (stream.match(commentMStart.substring(1), true, false))){
    state.tokenize = "comment";
    return "comment";
}

if((ch == commentMEnd.charAt(0)) && (stream.match(commentMEnd.substring(1), true, false)) && state.tokenize == "comment"){
    state.tokenize = null;
    return "comment";
}

if(isOperatorChar.test(ch)) {
    stream.eatWhile(isOperatorChar);
    return "operator";
}

stream.eatWhile(/\w\$/);
var cur = stream.current();
if (containsObject(cur, keywords)) {
    return "keyword";
}

function containsObject(obj, list) {
    var i;
    for (i = 0; i < list.length; i++) {
        if (list[i] === obj) {
            return true;
        }
    }
    return false;
}

return {
    startState : function(){
        return {
            tokenize : null
        };
    },
    token : function(stream, state){
if(stream.eatSpace()) return null;
var style = tokenize(stream, state);
return (state.tokenize || style);
}
CHAPTER 9. CODE

Listing 9.2: Client side code for the main window of TML written in Java using GWT

```java
package com.liu.trymylanguage.client.view;
import java.io.Serializable;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import net.zschech.gwt.comet.client.CometClient;
import net.zschech.gwt.comet.client.CometListener;
import net.zschech.gwt.comet.client.CometSerializer;
import net.zschech.gwt.comet.client.SerialTypes;
import se.liu.gwt.widgets.client.CodeMirror2;
import se.liu.gwt.widgets.client.CodeMirrorConf;
import se.liu.gwt.widgets.client.Cursor;
import com.google.gwt.core.client.GWT;
import com.google.gwt.core.client.JavaScriptException;
import com.google.gwt.core.client.JavaScriptObject;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.json.client.JSONArray;
import com.google.gwt.json.client.JSONObject;
import com.google.gwt.json.client.JSONString;
import com.google.gwt.json.client.JSONValue;
import com.google.gwt.uibinder.client.UiBinder;
import com.google.gwt.uibinder.client.UiField;
import com.google.gwt.uibinder.client.UiHandler;
import com.google.gwt.user.client.History;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.Window.ClosingEvent;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.CheckBox;
import com.google.gwt.user.client.ui.Composite;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.FlowPanel;
import com.google.gwt.user.client.ui.Frame;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.Image;
import com.google.gwt.user.client.ui.InlineHTML;
import com.google.gwt.user.client.ui.PopupPanel;
import com.google.gwt.user.client.ui.RequiresResize;
import com.google.gwt.user.client.ui.ScrollPanel;
import com.google.gwt.user.client.ui.SimplePanel;
import com.google.gwt.user.client.ui.TabLayoutPanel;
import com.google.gwt.user.client.ui.TextArea;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.Widget;
import com.google.gwt.visualization.client.DataTable;
import com.google.gwt.visualization.client.VisualizationUtils;
import com.google.gwt.visualization.client.visualizations.
corechart.AreaChart;
```

35
CHAPTER 9. CODE

```java
import com.google.gwt.visualization.client.visualizations.corechart.BarChart;
import com.google.gwt.visualization.client.visualizations.corechart.ColumnChart;
import com.google.gwt.visualization.client.visualizations.corechart.CoreChart;
import com.google.gwt.visualization.client.visualizations.corechart.LineChart;
import com.google.gwt.visualization.client.visualizations.corechart.Options;
import com.google.gwt.visualization.client.visualizations.corechart.PieChart;
import com.google.gwt.visualization.client.visualizations.corechart.ScatterChart;
import com.liu.trymylanguage.client.ErrorDialog;
import com.liu.trymylanguage.client.TMLService;
import com.liu.trymylanguage.client.TMLServiceAsync;
import com.liu.trymylanguage.client.TabWidget;
import com.liu.trymylanguage.client.exception.LangNotFoundException;
import com.liu.trymylanguage.client.exception.TMLException;
import com.liu.trymylanguage.shared.CodeDTO;
import com.liu.trymylanguage.shared.ConsoleDTO;
import com.liu.trymylanguage.shared.LangParamDTO;
import com.liu.trymylanguage.shared.OutputDTO;
import com.liu.trymylanguage.shared.Plot;

public class IDEView extends Composite {
    @UiField
    Button runButton;
    @UiField
    Button newTabButton;
    @UiField
    TabLayoutPanel tabPanel;
    @UiField
    SimplePanel tutorialArea;
    @UiField
    ScrollPanel consoleArea;
    @UiField
    Button renameButton;
    @UiField
    CheckBox plotCheckBox;
    private static IDEViewUiBinder uiBinder = GWT.create(
        IDEViewUiBinder.class);
    private TMLServiceAsync service;
    private CodeMirrorConf conf;
    private FlowPanel console;
    private LangParamDTO langParam;
    private CometClient cometClient;
    private boolean canRun = true;
    private PopupPanel loadingIndicator = new PopupPanel();
    private boolean cometStarted = false;
    @SerialTypes({
        Boolean.class, String.class, Plot.class,
        TMLException.class })
```
 CHAPTER 9. CODE

```java
public static abstract class InstantMessagingCometSerializer extends CometSerializer {
}

interface IDEViewUiBinder extends UiBinder<Widget, IDEView> {
}

public IDEView() {
  console = new FlowPanel();
  service = GWT.create(TMLService.class);
  service.getLangParam(new AsyncCallback<LangParamDTO>() {

    @Override
    public void onFailure(Throwable caught) {
      if (caught instanceof LangNotFoundException)
        showAddLangErrorDialog();
      else {
        ErrorDialog dialog = new ErrorDialog(caught);
        dialog.center();
        dialog.show();
      }
    }

    @Override
    public void onSuccess(LangParamDTO result) {
      langParam = result;
      conf = getConf(result);
      CodeMirror2 c = new CodeMirror2(conf);
      tabPanel.add(c, new TabWidget(tabPanel, false, c));
      c.onResize();
      ResizeableFrame f = new ResizeableFrame("doc/index.html"");
      tutorialArea.setWidget(f);
      f.setStyleName("tutorialArea");
    }
  });

  exportCopyCode();
  exportSelectLine();
  initWidget(uiBinder.createAndBindUi(this));
  Window.addWindowClosingHandler(new Window.ClosingHandler() {

    @Override
    public void onWindowClosing(ClosingEvent event) {
      doStop();
    }
  });

  consoleArea.setWidget(console);

  // Set custom ids to be able to access elements by DOM
  console.getElementById().setId("console");
  runButton.getElementById().setId("runButton");
  loadingIndicator.add(new Image("image/ajax-loader.gif"));
```

37
CHAPTER 9. CODE

```java
loadingIndicator.setGlassEnabled(true);

@Override
protected void onUnload() {
    doStop();
    super.onUnload();
}

// ************* UI Handlers ***************

@UiHandler("runButton")
void handleRunClick(ClickEvent e) {
    if (canRun) {
        runButton.setEnabled(false);
        console.clear();
        canRun = false;
        final CodeDTO code = new CodeDTO(getSelectedTabTitle(),
                                            langParam.getSuffix(), langParam.getCompileCmd(),
                                            langParam.getRunCmd(), langParam.getFeedbackRegex(),
                                            plotCheckBox.getValue(), langParam.getTimeout());
        code.setCode(getSelectedEditor().getValue());
        service.compile(code, new AsyncCallback<Void>() {
            @Override
            public void onFailure(Throwable caught) {
                canRun = true;
                runButton.setEnabled(true);
                new ErrorDialog(caught).show();
            }
            @Override
            public void onSuccess(Void result) {
                executed();
            }
        });
    }
}

private void executed() {
    if (cometClient == null) {
        CometSerializer serializer = GWT.create(InstantMessagingCometSerializer.class);
        String url = GWT.getModuleBaseURL() + "comet";
        cometClient = new CometClient(url, serializer, new CometListener() {
            @Override
            public void onRefresh() {
                System.out.println("Refresh");
            }
            @Override
            public void onMessage(String message) {
                // Handle message
            }
        });
    }
}
```
CHAPTER 9. CODE

```java
public void onMessage(List<? extends Serializable> messages) {
    for (final Serializable serializable : messages) {
        if (serializable instanceof String) {
            String message = (String) serializable;
            console.add(new InlineHTML(message));
            consoleArea.setWidget(console);
            consoleArea.scrollToBottom();
        } else if (serializable instanceof Plot) {
            final String message = ((Plot) serializable).getChartData();
            try {
                Runnable onLoadCallback = new Runnable() {
                    @Override
                    public void run() {
                        try {
                            drawChart(message);
                        } catch (JavaScriptException e) {
                            new ErrorDialog(e).show();
                        }
                    }
                };
                VisualizationUtils.loadVisualizationApi(
                    onLoadCallback, CoreChart.PACKAGE);
            } catch (Exception e) {
                new ErrorDialog(e).show();
            }
        } else if (serializable instanceof Boolean) {
            canRun = true;
            runButton.setEnabled(true);
        } else if (serializable instanceof TMLException) {
            canRun = true;
            runButton.setEnabled(true);
            new ErrorDialog((TMLException) serializable).show();
        }
    }
}

@Override
public void onHeartbeat() {
    //
}

@Override
public void onError(Throwable exception, boolean connected) {
    new ErrorDialog(exception).show();
    exception.printStackTrace();
}

@Override
public void onDisconnected() {
    System.out.println("disconnected");
}```
CHAPTER 9. CODE

```java
@Override
public void onConnected(int heartbeat) {
    System.out.println("connected: " + heartbeat);
    canRun = true;
    runButton.setEnabled(true);
}
}

doStart();

private void doStop() {
    if (cometClient != null) {
        service.invalidateCometSession(new AsyncCallback<Void>() {
            @Override
            public void onFailure(Throwable caught) {
            }
            @Override
            public void onSuccess(Void result) {
            }
        });
        cometClient.stop();
    }
}

private void doStart() {
    if (cometClient != null && !cometClient.isRunning()) {
        cometClient.start();
    }
}

private void doRestart() {
    doStop();
    doStart();
}

@UiHandler("newTabButton")
void handleAddTabClick(ClickEvent e) {
    CodeMirror2 codeMirror = new CodeMirror2(conf);
    tabPanel.add(codeMirror, new TabWidget(tabPanel, true, codeMirror));
    tabPanel.selectTab(codeMirror);
    if (tabPanel.getWidgetCount() == 2) {
        ((TabWidget) tabPanel.getTabWidget(0)).setCloseable(true);
    }
}

@UiHandler("renameButton")
void handleRenameClick(ClickEvent e) {
```
int selected = tabPanel.getSelectedIndex();
TabWidget w = (TabWidget) tabPanel.getTabWidget(selected);
w.showRenameDialog();
}

// ********************************************
private CodeMirrorConf getConf(LangParamDTO dto) {
String[] keywords = dto.getKeywords().split("\s");
JSONArray array = new JSONArray();
for (int i = 0; i < keywords.length; i++) {
array.set(i, new JSONString(keywords[i]));
}
String[] stringChar = dto.getStringChar().trim().split("\s");
JSONArray sarray = new JSONArray();
for (int i = 0; i < stringChar.length; i++) {
sarray.set(i, new JSONString(stringChar[i]));
}
JSONObject mode = new JSONObject();
mode.put("name", new JSONString("basemode").orElse("basemode"));
mode.put("keywords", array);
mode.put("stringCh", sarray);
mode.put("commentSingle", new JSONString(dto.getCommentSingle().orElse("" )));
mode.put("commentMStart", new JSONString(dto.getCommentMStart().orElse("" )));
mode.put("commentMEnd", new JSONString(dto.getCommentMEnd().orElse("" )));
mode.put("escapeCh", new JSONString(dto.getEscapeChar().orElse("" )));
dto.setOperators(dto.getOperators().replaceAll("\−", "−"));
dto.setOperators(dto.getOperators().replaceAll("\^", "ˆ"));
mode.put("isOperatorChar", new JSONString("\[ " + dto.getOperators().replaceAll("\", ", " ) + " ]" ));
CodeMirrorConf conf = new CodeMirrorConf();
conf.setMode(mode);
conf.setLineNumbers(true);
return conf;
}

private void showAddLangErrorDialog() {
final DialogBox confDialog = new DialogBox();
VerticalPanel content = new VerticalPanel();
HTML message = new HTML("Welcome to TryMyLanguage");
Button close = new Button("Click here to configure a language");
content.add(message);
content.add(close);
close.addClickHandler(new ClickHandler() {
public void onClick(ClickEvent event) {
confDialog.hide();
}
});
confDialog.setGlassEnabled(true);
close.addClickHandler(new ClickHandler() {
@Override
CHAPTER 9. CODE

```java
public void onClick(ClickEvent event) {
    History newItem("configure");
}
confDialog.setWidget(content);
confDialog.center();
confDialog.show();
}
public void copyCode(String code) {
    code = code.replace("<br>", "
");
    code = code.replace("<br />", "\n");
    code = code.replace("
nbsp;", " ");
    CodeMirror2 m = (CodeMirror2) tabPanel.getWidget(tabPanel
    .getSelectedIndex());
m.setValue(code);
}
private native void exportCopyCode() /
    {var that = this;
        $wnd.copyCode = $entry(function (code) {
            that.@com.liu.trymylanguage.client.view.IDEView::copyCode(
            Ljava/lang/String;)(code)
        });
    }-/*
private native void exportSelectLine() /
    {var that = this;
        $wnd.selectLine = $entry(function (event) {
            that.@com.liu.trymylanguage.client.view.IDEView::
            selectLine(Ljava/lang/String;)(event.target.id)
        });
    }-/*
private void setConsoleContent(ConsoleDTO dto, final int
    tabIndex) {
    console.clear();
    List<OutputDTO> out = dto.getConsoleContent();
    for (Iterator<OutputDTO> iterator = out.iterator(); iterator
        .hasNext();) {
        OutputDTO outputDTO = (OutputDTO) iterator.next();
        final int num = outputDTO.getLineNum();
        String content = outputDTO.getContent();
        if (num == 0)
            console.add(new InlineHTML(content));
        else {
            InlineHTML lbl = new InlineHTML(content);
            lbl.setStyleName("lineFeedback");
            lbl.addClickHandler(new ClickHandler {} {
                @Override
                public void onClick(ClickEvent event) {
                    tabPanel.selectTab(tabIndex);
                    tabPanel.forceLayout();
                }
            });
        }
    }
```
CHAPTER 9. CODE

```java
public void selectLine(String lineNumber) {
    int num = Integer.parseInt(lineNumber);
    int tabIndex = tabPanel.getSelectedIndex();
    tabPanel.selectTab(tabIndex);
    CodeMirror2 editor = (CodeMirror2) tabPanel.getWidget(tabIndex);
    Cursor cursor = new Cursor(num - 1, 0);
    editor.setSelection(cursor);
    console.add(lbl);
}

private CodeMirror2 getSelectedEditor() {
    int selected = tabPanel.getSelectedIndex();
    CodeMirror2 cm = (CodeMirror2) tabPanel.getWidget(selected);
    return cm;
}

private String getSelectedTabTitle() {
    int selected = tabPanel.getSelectedIndex();
    TabWidget w = (TabWidget) tabPanel.getTabWidget(selected);
    return w.toString();
}

class ResizeableTextArea extends TextArea implements RequiresResize {
    private Integer lastSelectedLine;

    public ResizeableTextArea() {
        super();
        setReadOnly(true);
    }

    @Override
    public void setText(String text) {
        super.setText(text);
    }

    @Override
    public void onResize() {
        int width = ResizeableTextArea.this.getParent().getOffsetWidth();
    }
```
CHAPTER 9. CODE

```java
    int height = ResizeableTextArea.this.getParent().getOffsetHeight();
    ResizeableTextArea.this.setSize(width, height);
}

public int getSelectedLine() {
    int pos = this.getCursorPosition();
    String content = this.getValue();
    int ln = 1;
    for (int i = 0; i < pos; i++) {
        if (content.charAt(i) == '\n')
            ln++;
    }
    return ln;
}

public Integer getLastSelectedLine() {
    return lastSelectedLine;
}

public void setLastSelectedLine(Integer ln) {
    lastSelectedLine = ln;
}
}

class ResizeableFrame extends Frame implements RequiresResize {
    public ResizeableFrame(String url) {
        super(url);
    }

    @Override
    public void onResize() {
        int width = ResizeableFrame.this.getParent().getOffsetWidth();
        int height = ResizeableFrame.this.getParent().getOffsetHeight();
        ResizeableFrame.this.setSize(width + "px");
        ResizeableFrame.this.setHeight(height + "px");
    }
}

// Create the chart based on the data
private void drawChart(String data) throws JavaScriptException {
    JSONObject chart = null;
    JavaScriptObject dataObj = null;
    Options options = null;
    String type = null;
    chart = new JSONObject(parseJSON(data));
    Set<String> keys = chart.keySet();
    options = Options.create();
    for (Iterator<String> iterator = keys.iterator(); iterator.hasNext();) {
        String string = (String) iterator.next();
```
if (string.equals("data")) {
  dataObj = ((JSONObject) chart.get("data")).getJavaScriptObject();
} else if (string.equals("chartType")) {
  type = ((JSONString) chart.get("chartType")).stringValue();
} else {
  JSONValue v = chart.get(string);
  if (v.isString() != null)
    options.set(string, (v.isString()).stringValue());
  else if (v.isObject() != null)
    options.set(string, (v.isObject()).getJavaScriptObject());
  else if (v.isNumber() != null)
    options.set(string, (v.isNumber()).doubleValue());
  else if (v.isBoolean() != null)
    options.set(string, (v.isBoolean()).booleanValue());
  else if (v.isArray() != null)
    options.set(string, (v.isArray()).getJavaScriptObject());
}

final DataTable dataTable = DataTable.create(dataObj);
switch (CoreChart.Type.valueOf(type)) {
  case LINE:
    consoleArea.setWidget(new LineChart(dataTable, options));
    break;
  case AREA:
    consoleArea.setWidget(new AreaChart(dataTable, options));
    break;
  case BARS:
    consoleArea.setWidget(new BarChart(dataTable, options));
    break;
  case COLUMNS:
    consoleArea.setWidget(new ColumnChart(dataTable, options));
    break;
  case PIE:
    consoleArea.setWidget(new PieChart(dataTable, options));
    break;
  case SCATTER:
    consoleArea.setWidget(new ScatterChart(dataTable, options));
    break;
  default:
    break;
}

private native JavaScriptObject parseJSON(String jsonString)
  throws JavaScriptException { 
  return eval('(' + jsonString + ')');
}
CHAPTER 9. CODE

Listing 9.3: TML service implementation

```java
class TMLServiceImpl extends RemoteServiceServlet implements TMLService {
    private static final long serialVersionUID = 1L;
    private CompileServerImpl server;
    private File tmldir;
    private CometSession cometSession;

    @Override
    public void init() throws ServletException {
        super.init();
        this.tmldir = new File(getServletContext().getContextPath());
        this.server = new CompileServerImpl(PlotDataConvertorService.getInstance(),
                                              new TmlUtil(), tmldir);
        System.out.println(getServletContext().getContextPath());
    }

    public void compile(CodeDTO code) throws TMLException {
        HttpSession httpSession = getThreadLocalRequest().getSession();
        cometSession = CometServlet.getCometSession(httpSession, false);
        if (cometSession == null) {
            cometSession = CometServlet.getCometSession(httpSession);
        }
        server.compile(code, cometSession);
    }
}
```

@Override
public void saveLang(LangParamDTO dto) throws TMLException {
    try {
        Pattern.compile(dto.getFeedbackRegex().replace("@", "(\d+?)"));
        ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream(tmidir.getAbsolutePath() + "WEB-INF/langparam.bin"), false);
        output.writeObject(dto);
        output.flush();
        output.close();
    } catch (IOException e) {
        e.printStackTrace();
        throw new TMLException("Can not save the language configuration: \n" + e.getMessage());
    } catch (PatternSyntaxException e) {
        throw new TMLException("Syntax error in regular expression for line feedback");
    }
}

@override
public LangParamDTO getLangParam() throws TMLException {
    ObjectInput obj = null;
    LangParamDTO dto;
    try {
        obj = new ObjectInputStream(new BufferedInputStream(new FileInputStream(tmidir.getAbsolutePath() + "WEB-INF/langparam.bin")));
        dto = (LangParamDTO)obj.readObject();
        if (dto==null)
            throw new LangNotFoundException();
        obj.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        throw new LangNotFoundException();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        throw new TMLException("There has been an error loading language configuration: \n" + e.getMessage());
    } catch (IOException e) {
        e.printStackTrace();
        throw new TMLException("There has been an error loading language configuration: \n" + e.getMessage());
    } finally {
        if (obj!=null){
            try {
                obj.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
return dto;
}

@Override
public LangParamDTO getLangParamAddLang() throws TMLException{
    if (getThreadLocalRequest().getRemoteAddr().trim().equals("127.0.0.1") ||
         getThreadLocalRequest().getRemoteAddr().trim().equals("0:0:0:0:0:0:0:1"))
        return getLangParam();
    else
        throw new TMLException("Access Denied");
}

@Override
public void InvalidateCometSession() {
    if (cometSession != null && cometSession.isValid()) {
        cometSession.invalidate();
    }
}
Bibliography


Appendix A

System Useability Scale Questionnaire

System Useability Scale


<table>
<thead>
<tr>
<th>Item</th>
<th>Strongly disagree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. I thought the system was easy to use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. I think that I would need the support of a technical person to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>be able to use this system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>very quickly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. I found the system very cumbersome to use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. I felt very confident using the system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with the system</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

52
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Amir Hossein Fouladi – Omid Zafari Khorvaj