Friedrich-Alexander-Universität
Erlangen-Nürnberg

T.cs
Chair for Computer Science 8
Theoretical Computer Science

Interactive web interface for the Loop/While interpreter

Documentation and installation manual

Johannes Kern

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Erlangen, January 8, 2020

Contents

1 Overview

1.1 Introduction

In typical undergraduate courses in theoretical computer science, a major aspect is to discuss different models of computability and their expressiveness. One important result is, that primitive recursion is not Turing complete, i.e. not equivalent to Turing machines or μ-recursion. In [Hof18], the difference in computational power between μ-recursion and primitive-recursive computation is presented by the languages WHILE and LOOP, where WHILE is a Turing-complete imperative language that offers basic control constructs like while loops and if-then-else, and LOOP is a restriction of WHILE that allows only a restricted loop construct ***loop*** x ***do*** ... ***enddo***, which is to be understood as "execute ... as often as the value of x".

In preliminary work, an interpreter for a variation of this two languages was developed [Geb18]. Currently the only available IDE for this languages is IntelliJ in combination with a plugin that is released together with the interpreter. The goal of this project was to develop an web-based interactive interface to that interpreter such that no extra software has to be actually installed on the user's computer. Additionally, the web interface provides an interactive tutorial which helps the user to learn the Loop/While language and documents the usage of the web interface.

1.2 License information and used third-party software

The software that was implemented as part of this project makes use of several CSS and javascript libraries that have to be made available to the user for the web-application to work properly, so licensing issues have to be considered.

The frontend uses a publicly available CSS from the purecss project[1] which is published under the Yahoo BSD license[2].

For user code inputs, the open source javascript-based BSD-licenced editor ace[3] ist used. Its code has been augmented by two source files (mode-LoopWhile.js and snippets/LoopWhile.js) to provide syntax highlighting for the Loop/While language. The new code must again be released under the BSD License.

All these libraries can be redistributed under the same corresponding license terms. The used copyrighted code mentioned above is also clearly separated from the code developed in this project, and since the BSD license is copyleft free, the framework itself is not affected by obligations of those licenses.

On the server-side, a python-based lexer library[4] is used. A comment in the code states that "This code is in the public domain", so there are no obligations with it.

To execute the Loop/While code itself, the Interpreter of Michael Gebhard is used[Geb18]. Its

[1] https://purecss.io/
[2] https://github.com/pure-css/pure-site/blob/master/LICENSE.md
[3] https://ace.c9.io/
[4] https://gist.github.com/eliben/5797351

code is licensed under the terms of the Apache license[5], but not published yet. For this project, the code has been modified such that error messages do not spoil paths or other sensitive information to the user. The Interpreter itself is not linked again the web interface, but is used as a command-line tool. Therefore, the code of the backend is not affected by obligations of that license. The modified interpreter code does not need to be published, unless the interpreter binary itself is made available publicly.(TODO?)

1.3 Software architecture

The main part of the software that was developed in this project is written in Python 3 and relies on the web framework TurboGears[6] in combination with the XML-template engine genshi.

It uses the widely-used model-view-controller (MVC) pattern [GS11] to separate view-related concerns from internal processes like the connection to the interpreter/debugger.

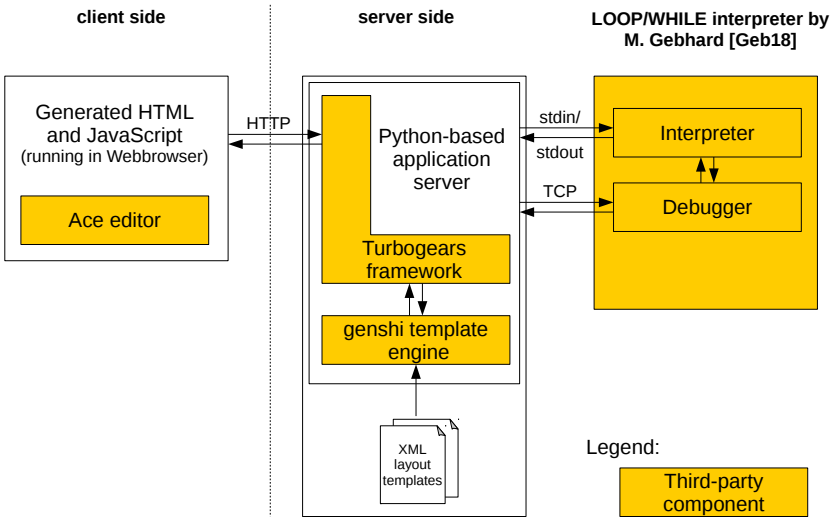

Figure 1.1: Software architecture

The view on the client side has two different modes: the interpreter and the debugger mode. In the interpreter mode, the user sees an editor, realized by the Ace editor library, in which he can edit his code. When the user clicks the run button, the JavaScript in the browser sends a request to the server, containing the user's program, and the server replies a randomly generated session id. This session id is then used by the script for authentication in further requests, e.g. when the contents of the terminal are updated or an user input is transmitted to the server. On the server side, a child process running the lwre binary is spawned to run the user's Loop/While code. The session (and therefore also the child process) is terminated, when either 1) the program terminates itself, 2) the user clicks the stop button or 3) a timeout elapses.

When the user clicks the debug button, the view switches to the debug mode, and the editor

[5] http://www.apache.org/licenses/LICENSE-2.0
[6] https://www.turbogears.org/

is replaced by a HTML-table containing the content of the editor, with the ability to display breakpoints and highlighting of the currently executed line, as it is needed by the debugger. This view is generated on the server side and also the child process is spawned when entering the debug mode, since the connection to it is not only necessary when the program runs, but also to set breakpoints before the execution of the user's code is started.

1.4 Repository structure

`src/`	source for the backend
`templates/`	XML/XHTML templates for turbogears
`web/`	files that will be directly served by the web server
`unit_tests/`	unit tests for the backend
`test_programs/`	test programs that are used by the unit tests
`config/`	sample configuration files

2 Installation and configuration

2.1 Requirements

We assume, that there is a Linux-based system (e.g. Ubuntu or Debian) with at least Python 3.5 installed. TurboGears can be installed by the package-management system of python (depending on your installation, you might need the command pip3 instead of pip for Python 3):

```
pip install TurboGears2 genshi transaction
```

This manual expects also an installed Nginx web server, but usage of other web servers like Apache httpd[1] should be possible without problems.

The Loop/While Interpreter binary is expected to be compiled seperately from the repository `https://git8.cs.fau.de/theses/michael-gebhard-ba/loopwhile-yacc-interpreter`. TODO: add correct repository/branch

2.2 Installation

There is a script `deliver.sh` in the repository, that does most of the necessary steps automatically. The variables used in this script can be adjusted:

`USER`, `GROUP`	the user- and groupname that gets owner of the installation directories and which are running the deamon
`INSTALL_DIR`	directory in which the python-based backend will be installed
`USER_SRC_DIR`	directory used to create temporary files. This directory must be exclusively used by the backend
`HTTPD_DIR`	directory to which all files are copied that will be directly served by Nginx or Apache httpd

After this script is executed, all necessary files are placed to their correct location. The next step is to set up the web server. An example configuration for Nginx can be found in the repository under `config/nginx.example`. This file must be copied to `/etc/nginx/sites_available`, and a symlink to this copy must be placed under `/etc/nginx/sites_enabled`. In this configuration, requests to the backend are delegated to the port 8080.

The last step is to set up a deamon for the backend. An example configuration file for systemd is found in the repository under `config/lw.service`. This file must be copied to `/etc/systemd/system/lw.service` and the settings for `User`, `ExecStart` and `WorkingDirectory` have to be adjusted. The deamon can then be started by

```
sudo systemctl start lw
```

To start the service automatically on boot, you have to enable it:

```
sudo systemctl enable lw
```

[1] https://httpd.apache.org/

2.3 Configuration and run script

The entry point of the backend deamon in the `run.sh` script. In this script, some basic parameters can be set:

`--logile=<PATH>`	Path to the logfile that shall be generated. A RotatingFile-Logger is used
`--loglevel=<LEVEL>`	Level used for logging. The higher the level, the more verbose the logging. Available levels are: CRITICAL, ERROR, WARNING, INFO, DEBUG, NOTSET. Default is INFO
`--host=<HOST_ADDR>`	Address on which the backend listens on. It is strongly recommended to use 127.0.0.1 here, so that the backend itself is only reachable by the upstream web server.
`--port=<PORT>`	Port the backend listens on
`--user_src=<PATH>`	Path to the directory for temporary files
`--max_sessions=<NUMERIC>`	Maximal number of sessions, i.e. concurrent instances of the lwre binary

For testing purposes, the `run.sh` script can also be executed directly without running the install routines.

2.4 Tutorial generator

The tutorial is generated using an own simple markup language, that provides some basic features for text formatting and syntax hightlighting for code snippets. A tutorial template according to this markup language is a sequence of text and commands described in the following table. Commands start with a backslash and cannot be nested. To be used in text, the backslash is escaped by double backslash.

headline { <*text*> }	headline of the tutorial
tableofcontents	a table of contents which is generated automatically
code { <*code*> }	syntax highlighted view of a given LOOP/WHILE code
html { <*text*> }	HTML code that gets directly embedded
link { <target_url> } { <link_text> }	a link to <target_url>
nobr { <*text*> }	A piece of text that should not contain a line break

The other commands are self-explanatory:
chapter { <*text*> } , *section* { <*text*> } ,
bold { <*text*> } , *italic* { <*text*> } , *linebreak*

The Tutorial Generator is started by just running the corresponding python source file, speciying input and output file on the command line:

```
python3 src/TutorialGenerator.py --infile=<TEMPLATE_FILE> --outfile=<OUTPUT_PATH>
```

This script is usually executed automatically by the `deliver.sh` script described in section 2.2, expecting the tutorial template in `templates/tutorial.template`. It will place the output in `templates/tutorial_container.xml`.

Example:

```
\headline{Headline of the Tutorial}
\tableofcontents
\chapter{example chapter}
  \section{example section}
    This is normal text.
    \bold{This is bold text.}
    \italic{This is italic text.}
    \linebreak
    \link{http://example.com}{This is a link.}
    \code{
//This is a Loop/While code snippet
in: i0
out: o0

o0 := i0
    }
```

Bibliography

[Geb18] Michael Gebhard. *Development of a Programming Environment and Interpreter for LOOP and WHILE. Bachelor Thesis in Computer Science.* Friedrich-Alexander Universität Erlangen-Nürnberg, 2018.

[GS11] Peter Hruschka Gernot Starke. *Software-Architektur kompakt - angemessen und zielorientiert.* Springer-Verlag, Berlin Heidelberg New York, 2011.

[Hof18] Dirk W. Hoffmann. *Theoretische Informatik.* Carl Hanser Verlag, München, 2018.