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

Erlangen, February 28, 2022

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. Furthermore, we use

[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

the Python libraries TurboGears[5] and Autobahn[6], which are licensed under MIT license, and genshi[7], which is licensed under a BSD license. Since all there libraries are linked dynamically, the Python source code of the backend has no obligations to be published under some certain license.

To execute the Loop/While code itself, the Interpreter of Michael Gebhard is used[Geb18]. Its code is licensed under the terms of the Apache license[8], 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.

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 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.

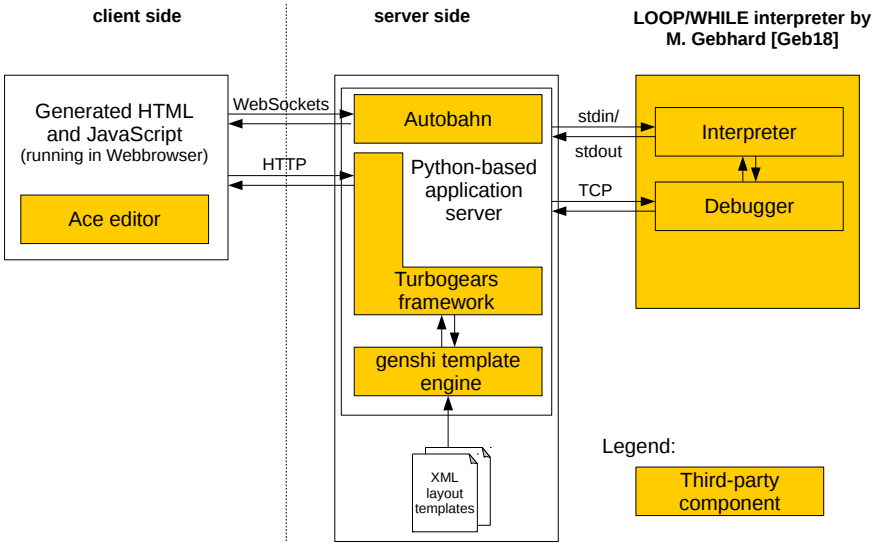

Figure 1.1: Software architecture

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

[5]https://www.turbogears.org/
[6]https://crossbar.io/autobahn/
[7]https://genshi.edgewall.org/
[8]http://www.apache.org/licenses/LICENSE-2.0

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 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 Communication protocols

The communication between the client JavaScript and the Python-based server relies on two different OSI Layer 7 protocols.

The first one used for synchronous communication is based on HTTP. It provides several commands to start and stop sessions, transmit user input in the terminal and to set breakpoints:

relative URL	POST arguments	server response
run	program_code	<SESSION_ID> in case of success, 0 in case of error
stop	session_id	"OK" in case of success, <ERR_MSG> in case of error
shell	input, session_id	<STRING> to print on the terminal
check_termination*	session_id	"running", "terminated", "timeout" or "error"
start_debug_session	program_code	"OK," + <SESSION_ID> in case of success, "FAIL," + <ERR_MSG> otherwise
debugger	session_id	an HTML -tag containing the debugger view
set_breakpoint	session_id, line_no	"OK" or "FAIL"
remove_breakpoint	session_id, line_no	"OK" or "FAIL"
debugger_poll_state*	session_id, line_no	"DIED", "RESTARTED", "FAIL" or a JSON-string containing the last stacktrace
debugger_action	session_id, action**	"OK" or "FAIL"

* deprecated commands that were used before the WebSockets-based protocol was developed.
** the following actions are available:

action	describtion
start	starts the debugger and stops at the first line of the root macro
continue	continues the debugger until the next breakpoint is reached.
stepinto	steps to the next line
stepover	steps to the next line without entering macros
stepout	steps out of the current macro
close	stops the debugger and closes the session. The session id gets invalidated

For asynchronous communication, another protocol based on HTML5 WebSockets is used. When a client has started the interpreter or debugger, it connects to the WebSockets server and transmitts its session id for authentification. By this communication channel, the client gets notified when the status of the lwre changes, a line can be printed on the terminal or a

debugger breakpoint is reached.

1.5 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
`doc/`	sources of the documentation you are reading

2 Local Installation and configuration

☞ Note: all steps described in this chapter are performed automatically by running the script `install.sh`

☞ Recommendation: Instead of local installation, a Docker container should be used (see Section 3).

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 and other prerequisites 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 waitress autobahn[twisted,accelerate]
```

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 separately from the branch 'webinterface' of the repository
`https://git8.cs.fau.de/software/loopwhile-yacc-interpreter`.

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 group name that gets owner of the installation directories and which are running the daemon
`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

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

`/etc/nginx/sites_enabled`. In this configuration, requests to the HTTP backend are delegated to the port 8080 and requests to the WebSockets backend get delegated to port 8081. Furthermore, the configuration ensures, that the original client IP address is transmitted in the request header to the respective backend to enable further request filtering there.

The last step is to set up a daemon 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 daemon 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
```

2.3 Configuration and run script

2.3.1 Overview

The entry point of the backend daemon 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 RotatingFileLogger 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=<LIMIT>`	Maximal number of sessions, i.e. concurrent instances of the lwre binary
`--ws_interface=<URL>`	Address on which the WebSockets service listens on. It is strongly recommended to use 127.0.0.1 here.
`--ws_post=<URL>`	Port the WebSockets service listens on
`--report_file=<PATH>`	File to which report data will be written
`--max_sessions_per_addr=<LIMIT>`	Maximal number of sessions per IP address

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

2.3.2 Security and performance

The nginx settings in `config/` do not use HTTPS at all. However, it is strongly recognized to set up the server such that it is only reachable by HTTPS.

Performance tests on a virtual server with 2 virtual CPUs and 2GB of RAM showed, that the application is able to handle over 128 parallel debugging sessions without recognizable impact to the response times. Nevertheless, the maximal number of parallel sessions is limited by

- the maximal number of processes allowed by the operating system
- the maximal number of open files in the server process
- the limit configured using the `--max_sessions` option (see Section 2.3.1).

The limits of the operating system can be obtained by the shell command `ulimit -a`. For the lw server daemon, they can be adjusted if necessary by adding the following two lines in the systemd service file (`/etc/systemd/system/lw.service`):

```
[Service]
...
LimitNOFILE=20000
LimitNOFILESoft=20000
LimitNPROC=20000
LimitNPROCSoft=20000
```

To reduce the risk of (distributed) DoS attacks, it is possible to establish a limit of parallel sessions per IP address, using the `--max_sessions_per_address` option. To determine a suitable limit, the report generator is a useful tool (see Section 2.3.3).

Another way is to restrict the maximal number of TCP connections per time unit in the firewall (using iptables).

2.3.3 Report generator

The report generator can be used to generate certain usage statistics. It writes status data at runtime to a file that is specified by the `--report_file` argument. The data contained in this file is persistent even after restarts of the server daemon.

To generate statistics, you have to call the respective Python script:

```
python3 src/ReportGenerator.py --infile=<PATH>
```

This script uses different metrics and can easily be extended.

2.4 Tutorial generator

The tutorial is generated using an own simple markup language, that provides some basic features for text formatting and syntax highlighting 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, specifying 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
    }
```

3 Docker container

This chapter describes how the loopwhile interactive interpreter service is installed using a Docker container. The container bundles the application together with an up-to-date nginx which is fully preconfigured.

3.1 Installation

The only requirement is an installation of Docker itself. Docker can be installed on ubuntu and debian using the following command:

```
sudo apt-get install docker.io
```

The application itself is then installed in three steps:

- Clone the git repository of the interactive interpreter to the machine on which the service shall run
- Run `sudo ./build_container.sh` to build the container
- Run `sudo ./run_container.sh` to start the container

The service is then listening on port 80. The container runs in background as a service and will start automatically after reboot.

3.2 Logfiles

In the directory that contains the `./run_container.sh` script, a subdirectory `logs` will be created, which is used to store all relevant logfiles. There you will find one subdirectory for each service:

- `loopwhile`: Logfiles of the lw service
- `nginx`: Logfiles of nginx
- `supervisor`: Logfiles of the supervisord deamon. Additionally, stderr and stdout of all supervised services are captured.

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.