fl0w - a Workflow Optimisation Tool
Philip Trauner, Christoph Heiss, Nico Kratky, Nico Leidenfrost, Sebastian Schaffler, Christine Zeh, Sascha Zemann

HTBLuVA Wiener Neustadt: Philip Trauner <philip.trauner@aol.com>, Christoph Heiss <me@christoph-heiss.me>, Nico Kratky <nico@nicokratky.me>, Nico Leidenfrost <leidenfrost.nico@gmail.com>, Sebastian Schaffler <s.schaffler@outlook.com>, Christine Zeh <zeh.chrisi@gmail.com>, Sascha Zemann <sascha.zemann@gmail.com>

fl0w - a Workflow Optimisation Tool

Abstract
This publication introduces fl0w, a robotics workflow improvement tool designed to save time and be a viable alternative to the newly developed Harrogate. The benefit of fl0w is offsite compiling and basic remote control of Wallaby Controllers. It focuses on the implementation of all the parts required for fl0w to fit its purpose, namely the Server, the network protocol, the Sublime Text plugin, and the Wallaby client. Additionally it describes how fl0w can and should be used and an example workflow is shown. Lastly there are some statistics presented that depict how much time is saved by using fl0w.

1. Introduction
fl0w was developed out of a need for a fast, reliable and wireless workflow solution that can compete with the currently available offerings. Its main objective is to reduce compile time to a minimum by offloading it to a separate computer while also moving all robot programming into Sublime Text.

2. Advantages
Sublime Text is a text editor with widespread use throughout the developer community, as detailed by Package Controls statistics [1, 2]. Unlike Harrogate [3], which is the new web interface and development platform introduced with the Wallaby, it is a native application with versions available for every major operating system. Sublime Text has a package manager (Package Control), syntax highlighting for a variety of languages, autocompletion, a capable plugin API and is very fast compared to similar text editors such as Atom [4, 5]. In contrast to Harrogate and the KISS IDE Sublime Text is also used for non-robotics programming which enables the user to work on all of their projects in the same development environment [6]. Hence it is a prime candidate for robot programming.

fl0w makes use of the plugin API and integrates tightly with Sublime Text. This enables offsite compiling, robot remote control through Sublime's interface and synced projects between multiple Sublime Text clients. By offloading workload from the Wallaby to a faster system compile time is sped up and battery power is saved.

3. Implementation
Python 3.5 was chosen as the main language for fl0w because it is supported by the Wallaby with minor effort and the Sublime Text 3 plugin API, although very recent language features are left

unused for the sake of backwards compatibility [7]. This way code can easily be shared between server and clients which enables the functionality of several internal parts of fl0w. This results in less duplicated code and a smaller codebase that is easier to maintain in addition to less points of failure. Another advantage of Python is its low power usage, which is important when running on battery powered devices such as the Wallaby Controller. Lastly fl0w greatly profits from Python's unconstrained access model. fl0w utilises a modified version of watchdog, a Python library that reports file system events, in the Sublime Text plugin and an unmodified version on the server for file synchronisation [8].

fl0w is distributed as a complete package containing Server, Sublime Text Client and Wallaby Client to avoid version mismatches.

1. Server

The server is the heart of fl0w. It compiles code, synchronizes source files with Sublime Text clients and the simultaneously generated binaries with Wallaby Controllers and its implementation is about as long as both clients combined (excluding shared code). There is no direct communication between Sublime Text clients and Wallaby clients at any time, but data can be shared between them through the server-side. Neither Sublime Text clients nor Wallaby Controller are owned by one another, instead every client is permitted to send information to any other client trough means provided by the server. This simplifies the underlying network protocol when multiple Wallaby Controllers are targeted by a single Sublime Text client and Vice-versa.

The server can talk with specific groups of clients at once via broadcasts. This way a message that's only supposed to be sent to one type of client does not arrive at the other ones.

There is no need for user authentication and permission management in fl0w because it should only be used in a local environment, so the server only has to receive the client type when a connection is established.

The server is made to run on an ARMv7 board with a recent Linux distribution installed because the Wallaby uses the same processor architecture. This allows for compilation on the server with binary-compatibility. It could also run on a Wallaby Controller although the speed benefit gained by compiling on a separate device is diminished by doing so, in consequence, it isn't officially supported nor recommended. During development of fl0w a Raspberry Pi 2 was used [9].

2. Network Protocol

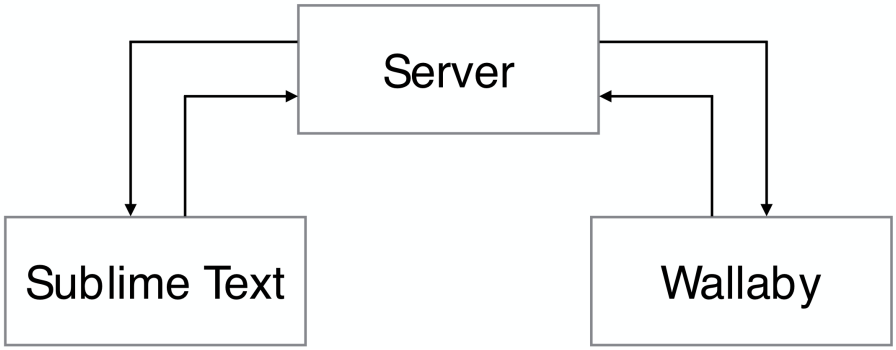

Fig. 1: Overview of fl0w's network paths which demonstrates that all data is routed through the server and no client to client connections exist.

fl0w uses its own TCP based layer-oriented asynchronous networking library. Every connection is handled in a dedicated thread, because scalability to hundreds of users was not a concern when designing fl0w.

It consists of many different network submodules that work independently. This essentially means that there are handlers on top of a router, which is a logical unit controlling data flow. The handlers can be shared between all users or just serve a single one. All handlers are assigned a route, which is best described as a logical data line connecting two handlers, over which they have full control.

Routes are implemented on server- and client-side and are deeply integrated into a custom socket, which hides buffer allocations and data type conversions. It is implemented as a proxy class that overrides the default send and receive methods found in Python's socket implementation.

When a send operation occurs data length, data type and the route are bundled into a header in front of the raw data. A receive call uses the prepended metadata to determine message length (used to do away with TCP packet fragmentation), data type (a hint to speed up automatic data type conversion) and the route.

1	Data Type
3	Padding
8	Data Length
16	Route
x	Data

Fig. 2: fl0w's custom socket packet structure that is required to hide buffer allocations and enable message routing. The numbers on the left represent how many bytes are reserved for each part.

Python's weak typing simplified automatic data type conversion and its overridable attribute lookup made the creation of a proxy class possible. Data structs are used to prepend metadata to messages and collections are transformed to JSON [10] when transferred over fl0w's custom socket.

3. Sublime Text Plugin

Sublime Text by default uses JSON configuration files for its menus but also includes an API call that allows for dynamic menus [10]. fl0w ships with an abstraction layer around that particular call, which allows an object-oriented menu approach with sub-menu support and Back buttons.

As with any user interface, network operations have to be asynchronous to prevent freezes. This is the main reason the Sublime Text plugin includes nearly the same routing based network stack the server does.

In addition to its networking stack the fl0w plugin also comes with a network submodule that synchronises all source code with the server. For this feature to work it was necessary to listen for filesystem events. Sublime Text's API includes functions that can be used to achieve this goal, however modifying watchdog, the library that is used server-side to capture such events, to run inside Sublime Text was chosen as it allows for additional shared code.

By default the Python interpreter inside Sublime Text behaves differently to the regular, non-embedded, version when it comes to imports. Source code can not be imported from the main plugin directory. Through the manual manipulation of import paths it was possible to bypass this behaviour and import modules inside the plugin directory which enabled the use of already existing Python code with minor changes.

Another feature of the plugin makes it possible to run actions on the client remotely. This enables simple remote control capabilities of a robot like starting and stopping a program.

4. Wallaby Client

The Wallaby client receives all compiled binaries from the fl0w server every time a change is made. It never receives source code. The binaries can be executed through Sublime Text and the Wallaby's default user interface. In a competition scenario the client can be terminated without the loss of programs to comply with the Botball ruleset which does not allow any network communication during a competitive run.

Like the Sublime Text client it also integrates tightly with fl0w's network stack.

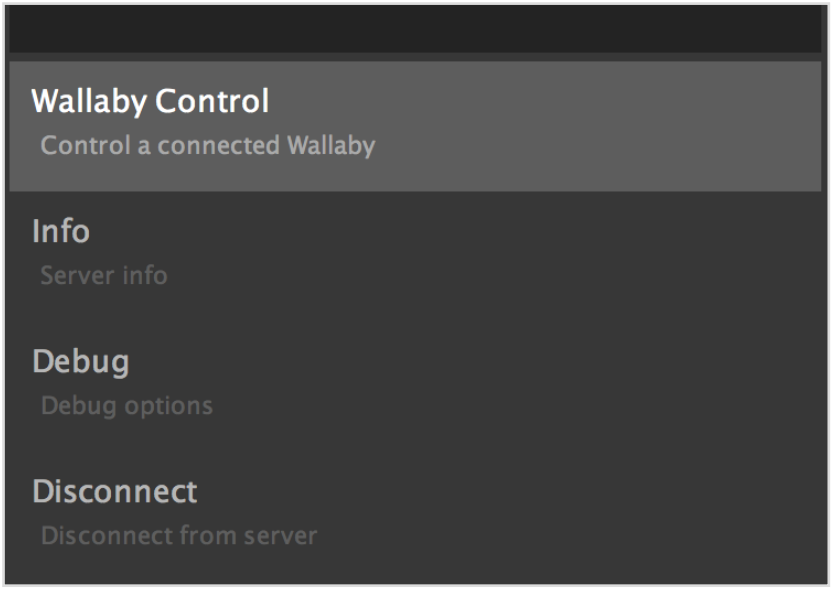

Fig. 3: fl0w's main Sublime Text interface while connected to a server.

4. Setup

1. Prerequisites

A Raspberry Pi 2 running Raspbian Minimal is recommended to run the server because it is the primary development platform of fl0w, is portable and can be powered off a battery-bank [9, 11].

The Sublime Text plugin works on all platforms but unlike Sublime Text itself, is only tested on OS X and Linux regularly.

2. Installation

Installation instructions are located in fl0w's GitHub repo which always contains the latest version of fl0w. [12].

5. Conclusions

1. Authentication

fl0w currently relies on network level security and does not use an authentication system. This was done because credential input on Wallaby Controllers would require a user interface. One possible solution to this problem is the creation of a Harrogate [3] app to modify client settings. Another solution is to require authentications in the Sublime Text client only, which would not prevent Wallaby Controller clients from downloading binaries, but would protect the source code.

2. Version Mismatches

Version mismatches are problematic because newer versions of fl0w usually contain more features than the previous ones. There is currently no plan to mitigate this issue other than an easy to use update system that can be triggered within the Sublime Text plugin.

3. Interpreted Languages

fl0w can not be applied to interpreted languages in its current state as it heavily focuses on binary synchronisation. This is accomplished through a separate file synchronisation route for Wallaby Controllers. In case of non-compiled languages the same file synchronisation route could be used for Wallaby Controllers and Sublime Text clients because the source code is required on both. Unless there is a demand for this feature, it will not be implemented.

4. Environment Variables and Arguments

The possibility to start program from the Wallaby's user interface prevents features such as startup arguments for programs and environment variable editing because they are not available in the Controllers user interface.

5. Installation

fl0w's installation procedure currently consists of many steps, and is not newcomer-friendly at all. It is planned to be improved with prepackaged requirements for the Raspberry Pi 2 and the Wallaby Controller. This way there is no need for the user to compile anything by themselves which at the moment can take up to 30 minutes.

6. Appendix

1. Example Usage

1. Connect to a fl0w server in Sublime Text
   (Tools → Command Pallet → fl0w: Menu → Connect)
2. Connect a Wallaby
3. Create hello_world.c in Sublime Text
4. Content of hello_world.c:

```
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

5. Save.
6. Open Wallaby Control
   (Tools → Command Pallet → fl0w: Menu → Wallaby Control)
   1. Choose Wallaby from list
   2. Use Run
   3. Select hello_world
7. Program will now run on the selected Wallaby and output is piped into Sublime Text

7. Glossary

API - Application Programming Interface, a set of routines, protocols and tools for building software applications

IDE - Integrated Development Environment, a software application that provides comprehensive facilities to computer programmers for software development

ARMv7 - a CPU architecture developed by ARM

TCP - Transmission Control Protocol, a protocol which provides reliable, ordered and error-checking delivery of a stream of octets between applications

JSON - JavaScript Object Notation, a lightweight data-interchange format

Binary - a computer file that is not a text file which is executable by the system

---

Acknowledgment

The author would like to thank the robotics team robot0nfire: Christoph Heiss, Nico Kratky, Nico Leidenfrost, Sebastian Schaffler, Christine Zeh, Sascha Zemann for continued support during

development; Dr. Michael Stifter for making the existence of our team possible; Daniel Maximilian Swoboda for answering all paper related questions and the KIPR development team without whom the Wallaby Controller would not exist.

References

1. Will Bond; “Package Control Statistics”; https://packagecontrol.io/stats; usage statistics page; 2015; accessed April 7th 2016
2. Jon Skinner; „Sublime Text“; https://www.sublimetext.com/; product page; 2014; accessed April 7th 2016
3. Stefan Zeltner and David P. Miller; “Kiss Your Old KISS Goodbye”; http://www.gcer.net/scoring/papers/KISS_Miller_KissYourOldKISSGoodbye.pdf; online paper; 2015; accessed February 26th 2016
4. Will Bond; “Package Control”; https://packagecontrol.io/; online homepage; 2015; accessed February 26th 2016
5. GitHub Inc.; “Atom”; https://atom.io/; product page; 2012; accessed February 26th 2016
6. Nafis Zaman, Braden McDorman, Jorge Villatoro; http://www.kipr.org/products/kisside; source code; 2009; accessed April 7th 2016
7. Guido van Rossum; “Python”; https://www.python.org/downloads/release/python-350/; release page; 2015; accessed February 26th 2016
8. Yesudeep Mangalapilly; “watchdog”; https://github.com/gorakhargosh/watchdog; source code; 2010; accessed February 26th 2016
9. Eben Upton; “Raspberry Pi”; https://www.raspberrypi.org/products/raspberry-pi-2-model-b/; product page; 2015; accessed February 26th 2016
10. Francis Galiegue, Kris Zyp and Gary Court; “JSON Schema: core definitions and terminology”; http://json-schema.org/latest/json-schema-core.html; online paper; 2013; accessed February 26th 2016
11. Diederik de Haas and Toni Spets; “raspbian-ua-netinst”; https://github.com/debian-pi/raspbian-ua-netinst/graphs/contributors; source code; accessed March 17th 2016
12. Philip Trauner; “fl0w”; https://github.com/robot0nfire/fl0w; source code; 2016; accessed February 26th 2016