fl0w - a development environment for the KIPR Wallaby

Philip Trauner, Christoph Heiss, Sebastian Schaffler, Nico Kratky, Nico Leidenfrost, Christine Zeh, Sascha Zemann
Department for Computer Science
Secondary Technical College
Wiener Neustadt, Austria
Email: philip.trauner@arztpraxis.io, me@christoph-heiss.me, se.schaffler@gmail.com,
nico@nicokratky.me, leidenfrost.nico@gmail.com, zeh.chrisi@gmail.com, sascha.zemann@gmail.com

***Abstract*—This publication introduces fl0w, an alternative development environment and monitoring solution for the KIPR Wallaby. The aim of fl0w is to improve the Botball program development experience. It focuses on the components that make up fl0w, namely a file synchronization protocol that maintains a consistent state across all connected controllers, a route based and data type preserving network protocol with peer-to-peer piping capabilities, a discovery protocol that connects all controllers together automatically, a Sublime Text 3 plugin which enables in-line sensor readouts, program execution, program editing, and keyboard shortcuts, and a browser-based management front-end to manage the controller fleet.**

***Index Terms*—file synchronization, networking, LAN discovery, development environment, monitoring**

I. Introduction

fl0w [1] was developed out of the need for a fast, reliable and wireless workflow solution that can compete with the currently available offerings like Harrogate [2]. Its goals are to transform the connected controllers into a redundant fleet of logical units that share the same binaries and source code, real-time in-line sensor readouts, programming as well as robot management inside Sublime Text 3 [3], and a browser-based front-end.

II. Implementation

fl0w is the combination of all its components split into Wallaby [4] client and server. All components emphasize shared code and are therefor written in Python [5] 3.3.6 to remain compatible with the Sublime Text 3 plugin environment.
Python was chosen as the implementation language for fl0w because of its rich standard library and its lightweight virtual machine that can run on a Wallaby controller with acceptable speed. A pre-compiled version of Python is bundled with the fl0w installer, r0adrunner, because it is not present on the Wallaby by default.
All fl0w components are designed as libraries to allow for integration into other projects. fl0w utilizes a client-server networking model with random server assignment. Client-server was chosen because the server can act as a data source for all clients and logic can be clearly separated.

III. Components

A. undergr0und

undergr0und [6] is an asynchronous, route based, and data type preserving network protocol with peer-to-peer piping capabilities. It automatically converts data into a transferable format and prepends binary headers to allow for reconstruction on the other end.

TABLE I
The binary encoding of messages in undergr0und (in bytes)

Data type	Route ID	Data
1	2	*

The Python version is built on top of a fork of ws4py [7] and the JavaScript [8] version utilizes regular WebSockets [9] in arraybuffer mode. It was created because the requirements for fl0ws network protocol specification changed constantly and a dynamic approach to networking was required. Instead of one monolithic network protocol, undergr0und emphasizes small sub protocols. undergr0und manages itself through its own concepts by transmitting the initial handshake and metadata on a route.

1) undergr0und.js: undergr0und.js [10] is the feature complete client-side JavaScript port of undergr0und that was required for dashb0ard [11], the browser-based front-end for fl0w. It utilizes node-jspack [12] to unpack the binary messages it receives from the undergr0und server. It is based on the Python version but does not implement the server because there is currently no need for that functionality.
undergr0und.js can be used in Node.js [13] as well as browsers that support WebSockets. browserify [14] is used to create the browser version.

2) Exchange table: To reduce the required additional bandwidth per message that is introduced with variable character count routes, a numbered route lookup table is generated on startup by clients and the server. Before any communication takes place these lookup tables are exchanged. Route ID 0 is reserved for self management purposes.

```
>>> routes = {"echo": Echo(),
...     "help": Help()}
>>> create_exchange_map(routes)
{0: "meta", 1: "echo", 2: "help"}
```

Fig. 1. Creation of exchange maps used to lower bandwidth requirements

3) Connection constructs:

a) Route: A client → server / server → client construct. Invoked with the "send" call.

b) Pipe: A client → server → client construct. Invoked with the "pipe" call.

Clients can be targeted with unique IDs that are generated randomly for all connected peers by the server. There is no predefined mechanism in the network protocol that exposed these peer IDs, the application using undergr0und has to provide a way of making them available. This approach allows for more flexibility if additional peer-metadata has to be provided.

Pipe messages are packaged inside regular route messages with additional headers in the data segment. A server route called "pipe" unpacks the regular and the extended headers and forwards the message to the targeted peer. The original sender ID is always included for a possible response.

4) Data type preservation: The original type of the data segment is present inside the header (see Table I) to accurately reconstruct sent data on the other end. JSON [15] is used to transfer lists and dictionaries, and regular UTF-8 encoded strings are utilized to transfer integers, floats, null/none types and strings.

B. behem0th

behem0th [16] is a continuous network file synchronization protocol developed out of the need for an embeddable solution that would not require an additional background program to be present on the system running Sublime Text 3, instead utilizing its plugin environment.

It uses a client-server networking model without peer-to-peer capabilities to stay in line with undergr0und. It uses regular sockets instead of WebSockets because the in-browser components of fl0w never interacts with it. The file-system is monitored using the watchdog [17] library.

1) Synchronization: On startup, behem0th synchronizes files based on their last modification time and MD5 hash. After that, files get synchronized to other clients as soon as a file-system event happens. Synchronization conflicts are resolved on the server and it is possible to use behem0th with a theoretically indefinite amount of clients.

2) Transfer: behem0th neither sends nor receives files as a whole, instead it transmits the file as small blocks (4096 bytes), which are written to a temporary file on disk as soon as they are received. This is done to allow the transfer of large files even in very random-access-memory-constrained environments.

3) Protocol: behem0th is independent from undergr0und and defines its own network protocol. It is completely text-based, uses UTF-8 for character encoding and JSON as format encoding. behem0th is designed around 'requests' (see Figure 2) and 'routes'. Each request is a JSON-formatted string and separated by a newline.

```
{
    "route": "<route-name>",
    "data": "<route-specific data>"
}
```

Fig. 2. behem0th request encoding in JSON

A route can also send additional data if needed (a 'payload'). For binary data, base64 [18] encoding is used. After each payload, there is a newline to delimit the payload and the next request.

4) Security concerns: Although MD5 is deprecated and known to suffer from extensive vulnerabilities, behem0th is designed to only run in a local network, which is controlled by the users of fl0w This assumption also simplifies the implementation greatly.

C. dashb0ard

dashb0ard [11] is a single-page web application designed to manage the fl0w fleet.

It can be used to configure connected controllers and obtain sensor readouts as well as debug logs.

It utilizes undergr0und.js to retrieve data from fl0w, Vue.js [19] for its user interface, Bootstrap 3 [20] as a design baseline, Flot [21] to display graphs and jQuery [22] for additional DOM manipulation.

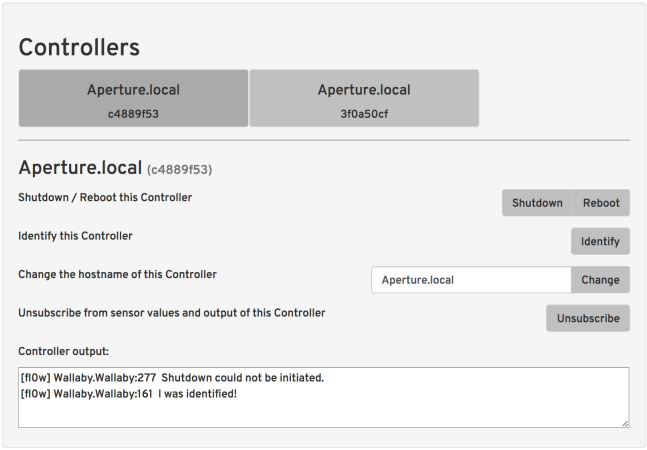

Fig. 3. The control and configuration page of dashb0ard

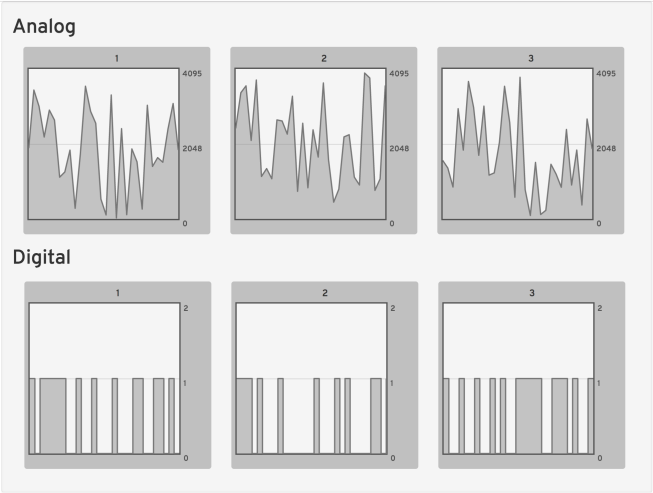

Fig. 4. The sensor readout page of dashb0ard

D. edit0r

edit0r [23] is the Sublime Text 3 plugin. It is modeled as a fl0w client and allows for remote robot management, source code synchronization as well as in-line sensor readouts.
It also provides quick access to dashb0ard, which is required because the IP addresses of controllers are not always static. behem0th is embedded into edit0r and enables the source code synchronization. In its current form only C programs are supported because the folder structure for Python programs in Harrogate is different.

1) Robot selection: fl0ws networking model allows for the management of multiple controllers without a direct connection to them. The controller hostname is utilized to identify the robots in the user-interface. Controllers can be selected to obtain additional functionality such as in-line sensor readouts and keyboard shortcuts.

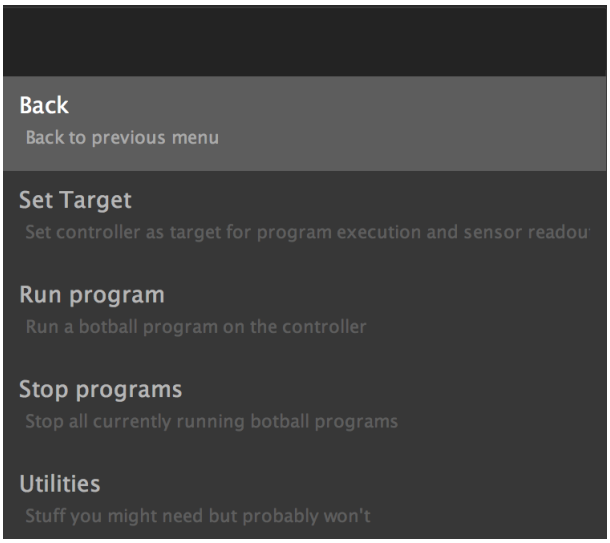

Fig. 5. edit0rs robot management interface in Sublime Text 3

2) Robot management: Botball programs can be started and/or stopped. Additionally, edit0r allows for modification of the controller hostname, playback of a sound for identification purposes, listing of running processes, and power management functionality (reboot, shutdown).

3) In-line sensor readouts: Invocations of the functions ”analog” and ”digital”, functions typically used to obtain sensor values, are located every time the content of a view changes and the parameters of the calls are grouped across views. edit0r subscribes to all required sensor ports by requesting them from the currently selected controller. The controller keeps track of all subscriptions to sensors and continuously transmits the appropriate values to the Sublime Text 3 plugin. Sublime Text 3 phantoms, which are in-line fields to display data within a text view, are created as soon as new sensor values are available. This approach has the disadvantage of higher than usual processor utilization by edit0r, because phantoms were primarily designed to display infrequently updating content. To partially circumvent this, sensor readouts can be disabled and are not enabled by default.
In-line sensor readouts can be toggled on a per view basis with a keyboard shortcut.

```
1  if (analog(3)) { 682
2      if (digital(5)) { 1
3          if (analog(1)) { 508
4              printf(digital(1)); 1
5          }
6      }
7  }
```

Fig. 6. edit0rs inline sensor-readouts in Sublime Text 3

E. r0adrunner

r0adrunner is the installer of fl0w.
It is built to mimic a Wallaby controller software update to remain compatible with the preexisting update functionality found in Harrogate. It is implemented in Python 2.7, which is already installed on all Wallaby controllers. A full-fledged precompiled version of Python 3.6.0 is installed by r0adrunner because fl0w utilizes features of Python 3.3.6.
It also changes the Wallaby hostname to their unique manufacturing ID because fl0w uses hostnames to represent controllers in its user facing components. r0adrunner is distributed as an archive that has to be copied to an USB storage medium.

F. disc0very

disc0very [24] is the LAN discovery protocol of fl0w. It relies on UDP [25] broadcasts to coordinate server and client assignment. Wallaby controller instances as well as the Sublime Text 3 plugin edit0r make use of it to determine if there is already an active fl0w server on the network upon startup without user interaction.

1) Discovery process: Upon launch disc0very listens for advertisements from an already running fl0w server. If an advertisement is received the client-mode is engaged and a connection to the server is established. If no advertisement has been received after 2s server-mode is engaged.

```
{
    "address": "<address>",
    "port": "<port>"
}
```

Fig. 7. disc0very advertisement encoding in JSON

2) Error recovery: disc0very actively listens for advertisements in server-mode to prevent more than one running server on the network. If a server receives an advertisement from another server both shut down, wait for a random amount of time (between 1 and 5 seconds), and return to discovery mode.

G. *fl0w*

fl0w itself is the combination of all other components split into Wallaby client and server.

1) Wallaby client: The Wallaby client serves as an information source for sensor values, program output, as well as currently running processes. Additionally it exposes its own standard output for monitoring purposes.

2) Server: The server keeps track of all clients and provides means to acquire their peer IDs, which are used by undergr0und to target specific clients. Additionally it serves dashb0ard and handles program compilation. It utilizes undergr0und for networking, behem0th for file-synchronization, and disc0very to determine if another server is already running in the network.
In that situation the start is interrupted, the Wallaby client is started and it connects to the available server instance, otherwise server and client are started.

IV. Conclusion

After prolonged testing it was deemed reasonable to claim that fl0w can improve the Botball development work-flow on Wallaby controllers. Experienced Botball teams that would rather program in native tools than the browser-based Harrogate and want to observe their sensor-readouts in realtime can benefit from fl0w.

Networking boilerplate code can be cut down drastically with a fitting networking solution already in place.

Security was never a design goal, instead relying on network interface level security was chosen to save time. This assumption will cause problems in densely populated wireless networks.

There is a high amount of complexity involved when performing continuous file-synchronization with an undetermined amount of clients across multiple operating-systems and file-systems.

Support for Python as a robot programming language as well as basic remote procedure call functionality is the next development goal.

Acknowledgment

The authors would like to thank Dr. Michael Stifter for making the existence of our robotics 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] Philip Trauner, Christoph Heiss, Sebastian Schaffler, *fl0w*, https://github.com/robot0nfire/fl0w, source code, accessed February 9th 2017
[2] KIPR, *Harrogate*, https://github.com/kipr/harrogate, source code, accessed February 9th 2017
[3] Sublime HQ, *Sublime Text 3*, https://www.sublimetext.com/3, product page, accessed February 9th 2017
[4] KIPR, *Wallaby Controller*, http://botballstore.org/product/wallaby-controller, store page, accessed February 9th 2017
[5] Python Foundation, *Python*, https://www.python.org, project page, accessed February 9th 2017
[6] Philip Trauner, *undergr0und*, https://github.com/robot0nfire/fl0w/wiki/undergr0und, implementation notes, accessed February 9th 2017
[7] Philip Trauner, *ws4py*, https://github.com/robot0nfire/ws4py, source code, accessed February 9th 2017
[8] Ecma International, *ECMAScript*, https://www.ecma-international.org/ecma-262/7.0/index.html, language specification, accessed February 12th 2017
[9] A. Melnikov, *The WebSocket Protocol*, https://tools.ietf.org/html/rfc6455, request for comment, accessed February 9th 2017
[10] Philip Trauner, *undergr0und*, https://github.com/robot0nfire/undergr0und.js, source code, accessed February 9th 2017
[11] Sebastian Schaffler, Philip Trauner, *dashb0ard*, https://github.com/robot0nfire/dashb0ard, source code, accessed March 16th 2017
[12] Peter Griess, *node-jspack*, https://github.com/pgriess/node-jspack, source code, accessed February 9th 2017
[13] Node.js Foundation, *Node.js*, https://nodejs.org, product page, accessed February 9th 2017
[14] James Halliday, *browserify*, http://browserify.org/, product page, accessed February 9th 2017
[15] T. Bray, Ed., *JSON*, https://tools.ietf.org/html/rfc7159, request for comment, accessed February 9th 2017
[16] Christoph Heiss, *behem0th*, https://github.com/robot0nfire/behem0th, source code, accessed February 9th 2017
[17] Yesudeep Mangalapilly, *watchdog*, https://github.com/gorakhargosh/watchdog, source code, accessed February 9th 2017
[18] S. Josefsson, *Base64*, https://tools.ietf.org/html/rfc4648, request for comment, accessed March 22nd 2017
[19] Evan You, *Vue.js*, https://vuejs.org/, product page, accessed March 16th 2017
[20] Twitter, Inc., *Bootstrap 3*, http://getbootstrap.com, product page, accessed March 16th 2017
[21] David Schnur, *Flot*, https://github.com/flot/flot, source code, accessed February 9th 2017
[22] jQuery Foundation, *jQuery*, https://jquery.com/, product page, accessed February 9th 2017
[23] Philip Trauner, *edit0r*, https://github.com/robot0nfire/fl0w/tree/master/Sublime/fl0w, source code, accessed February 9th 2017
[24] Christoph Heiss, *disc0very*, https://github.com/robot0nfire/fl0w/blob/master/Shared/Disc0very.py, source code, accessed March 30th 2017
[25] J. Postel, *UDP*, https://www.ietf.org/rfc/rfc768.txt, request for comment, accessed March 30th 2017