Final report of the development of **lupulo**

Alejandro López Espinosa

VIVES University

January 20, 2016

Table of contents

Useful info

lupulo is developed as a free software project under the GPL license.
lupulo's backend currently **is built with python2** because twisted is not written to be compatible with python3.
lupulo is currently at the 0.3.0 stable release.

You can find the source code in github.com/kudrom/lupulo
You can download lupulo with pip.
The docs are updated in ReadTheDocs.

Goals

What's lupulo

lupulo is a web framework to build realtime web pages that monitor and/or command a device.

Only two goals:

- Easiness of development
- Extensibility of complex behaviour

In order to be able to provide those two goals, the framework provides two sets of abstractions, the ones targeted at **beginner** users and the ones targeted at **experienced** users.

Main abstractions

- Data schema language
- Layout language
- Templates
- Widgets
- Accessors
- Listeners

All of this abstractions provide a smooth **workflow** that both type of users should follow to develop a web page.

Create a valid project

You can install the software with:
`pip install lupulo`
Once you have installed lupulo, you type `lupulo_create` in a directory to create a valid lupulo project.
It's going to create a bunch of directories and files that you're going to change later on.
One of those files is the **settings.py** file which you'll modify to configure the backend mostly.

Write the data schema

The data schema is made of **source events** that relate to every sensory information the device is sending.
Each source event is described by a set of obligatory parameters depending of its **type**.
There are two kind of types, the **primitive** and the **aggregated**.
The primitive type describes some raw data like **number**, **enum** or **date**.
The aggregated type is an aggregation of other types. There are two aggregated types: **dict** and **list**.
There is no limit to aggregation.

Write the layout

A widget is some **JS object** that renders information in the web page.
A layout is a **description** of a widget, it describes how the widget is going to behave in the web page.
In the layout the user **binds** a particular widget with a particular data source event.
A layout can **inherit** from another layout some or all of its parameters.
There is no limit to the levels of inheritance.
Multiple inheritance is not allowed.

Templates

You can modify the html pages with **jinja2** templates.
At the moment lupulo provides some base templates that you can extend.
You can design your own url sitemap and use RESTful principles for the command interface. You only need to write your own **urls.py** file in the same fashion that django does.
Each url must be handled by a **twisted Resource**.
The templates are rendered **asynchronously**.

Launch

Once you have written everything, you launch the server with `lupulo_start` in the 8080 port (by default).
You can edit the layout and data schema **without the need to restart the web server**.
You can use the **debug web page** to see how things are working in the frontend.
You can use a **standalone sse client** to see what the backend is sending through the sse data connection.

Write widget

If you want to visualize some information in a new way, you can write your own widget and use them later in your layout file.
You have to call the **Widget supertype** with the this object and the layout as parameters.
The supertype will add some attributes to the object that will allow you later on to modify the web page.
You can **aggregate** other types of widgets by constructing them in your constructor and calling them later to modify the web page.
The framework is going to call **paint** each second and **clear_framebuffers** when it needs to clear the visualization that the widget has created in the web page.
You might need to use the **accessors** abstraction.
You should provide **dynamic sizing**.

Accessors

The accessors abstraction allow a widget to access the data **without knowing its schema**.
The widget **delegates** the real access of the data to another object that is constructed with the data schema and that retrieves the data the widget needs.
The idea is to allow the programmer of the web page to describe in the layout the data it wants a widget to render **paying attention to the widget needs and the structure of the data**.
Once the accessors are described in a layout, the widget will construct them with the **get_accessors** function and will call them in the paint function to get the data it has to render.

Write listener

In order to connect the backend to the data source, the user can build its own listener that will retransmit to the backend the data it receives.
A listener is a **twisted service** that will be run when the server is started.
Once the listener is created by the backend, it will **publish** data it receives from the device to a twisted resource that will validate and push the information to the frontend.

Main components

There are two main components of the project, the **backend**, which is built with python2 and twisted and the **frontend** which is built in javascript.
These two main components communicate through an asynchronous data link provided by a HTML5 API called **Server Sent Events**, which provides a unidirectional stream to push information from the server to the browser.
The user doesn't see this asynchronous data link, it only configures what widgets listen to what data sources.

Diagrams

The future

For future releases it could be nice to:

- Expand the number of listeners
- Expand the number of widgets
- Review both high level languages
- More granularity in the paint loop

Demo

Thanks for your attention.