Report of the progress made in **lupulo**

Alejandro López Espinosa

VIVES University

November 11, 2015

Table of contents

Useful info

lupulo is developed as a free software project under the GPL license (for the moment).
lupulo's backend currently **is built with python2** because twisted is not written to be compatible with python3.
lupulo is currently at the 0.1.0 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

- Dynamic description of the data the device is sending
- Allow big modifications of the web page
- Multiple web pages for a single device
- Ease of development of the web pages
- Record the data for offline analysis
- Reuse of code
- Extensibility of the behaviour

What's lupulo

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

Main abstractions

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

Create a valid project

You **can** install the software with:
`pip install lupulo`
But you **should** build your own distribution from the source code and later install it from your filesystem with pip. You can use a Makefile that does just that and something more.

Once you have installed lupulo, you type `lupulo_create` in a directory to launch the server.
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**.

Launch

Once you have written everything, you launch the server with `lupulo_start`.
You can test that everything is working as expected with a standalone sse client called `lupulo_sse_client` or you can go to localhost:8080 and enjoy your first lupulo realtime web page.

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.

Backend

The backend is responsible of:

- Compile the data schema
- Compile the layout
- Implement a web server
- Push information to the frontend whenever it's available
- Provide the listeners abstraction
- Provide the templates abstraction
- Provide hot layout and data schema

Frontend

The frontend is responsible of:

- Initiate the connection towards the server
- Render the information
- Provide the widgets abstraction
- Provide the accessors abstraction

The future

I'm working currently on (for the 0.2.0 release):

- The templates abstraction
- Building a smart debug web page
- Building a smart web page for mongodb
- Expanding the number of default widgets
- Spotting and fixing bugs

I'd like to work on:

- Expanding the number of default listeners
- Test the framework with real devices
- Think about what more abstractions I can provide in the frontend and the embedded device

Demo

Any question?