**Lab Assignment 11 - Event oriented programming**

Luis Raul Villanueva Lopez ([raulvlopez@gmail.com](mailto:mraulvlopez@gmail.com))

Iram Castañeda ([blinketito@gmail.com](mailto:blinketito@gmail.com))

May the 26th, 2013

**Universidad Autónoma de Guadalajara**

Event oriented programming also known as event-driven programming (EPD) is a software paradigm where program flow is determined by events. In embedded systems, as opposed to standard user operating systems, developer has to manage I/O and hardware resources by using interrupts and here is where EPD substantially decrease complexity and result in more robust hardware.

On this paper, we present a design proposal for an Event Oriented Programming architecture cable of handling three kinds of events:

* Asynchronous events
* Periodical Events
* One-shot events

This architecture should let the user to enable or disable events at will, and should be easy to implement and port to different hardware above all.

**Introduction**

Event oriented programming can be split in at least 2 blocks:

* The Events Generator.
* The Events Dispatcher.

Usually, someone else has already implemented the basic set of events so that the developer can only focus on how to handles those events. Even though this may suit many applications, this is not always the case. Then, there should be a way to generate new events that better suit to the problem in question. This design attempts to handle the 2 before mentioned blocks so that it can fit into every application.

Also, there may be scenarios where there is no an event handler in the system, this design should be easy to implement on any architecture.

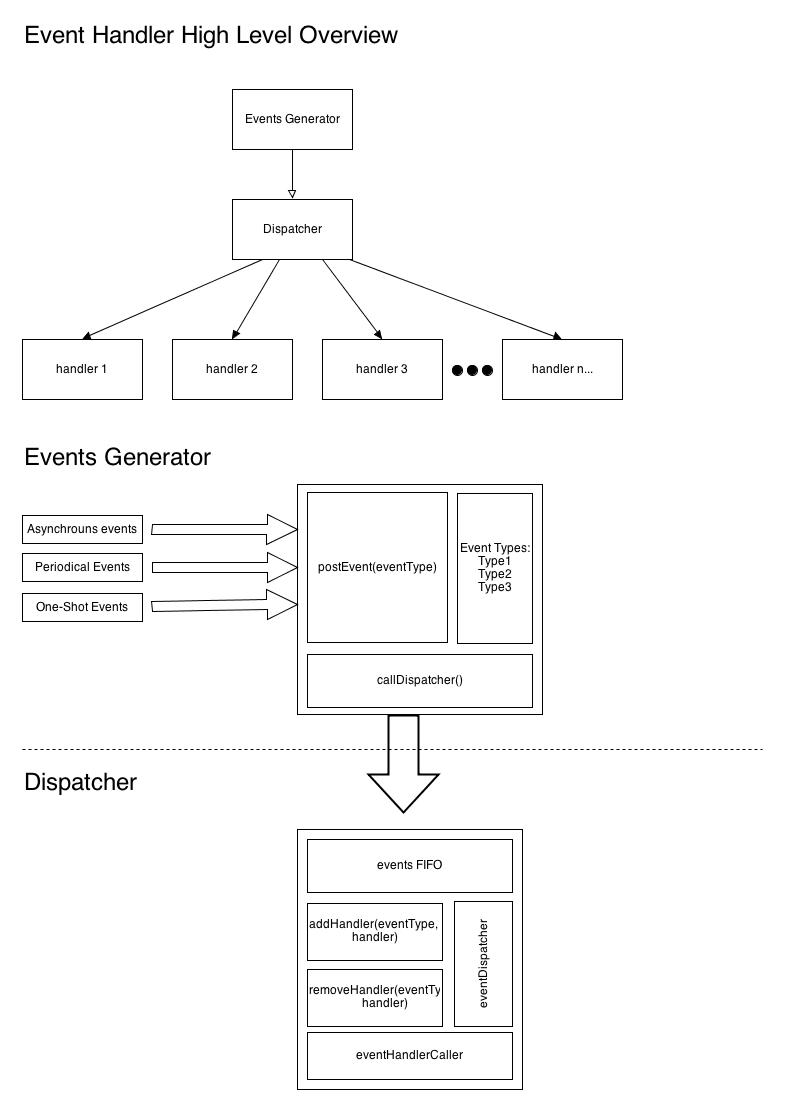
This is the high levels overview block diagram:



**Events Generator**

There should be an easy way to tell that an event has just occurred. Events generator is a module that provides function calls that post events to the system. Event types are defined by the final user and can be created as many as the system allows.

Also, as discussed previously the design shall be able to support the aforementioned kind of events. In this case, our design does not directly distinguish among the three types as they can be handled as an regular events, nothing more.



However, either asynchronous, synchronous or one-shot events has to be programmed by the user. This is, this design provides a way to post events to the system, but someone has to call the postEvent function. It is up to the developer to implement calls to this function in a way that the three kinds of events are supported.

**Events Dispatcher**

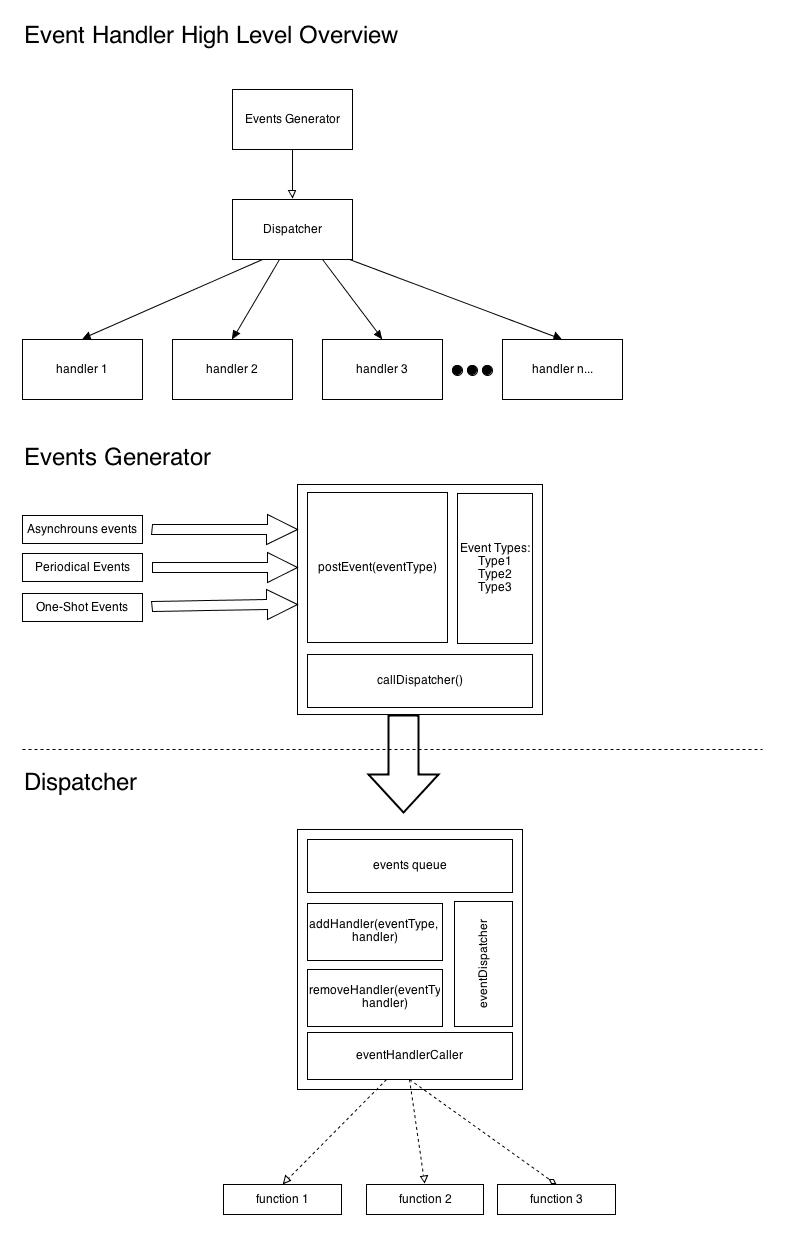
Dispatcher basically contains a table with a list of events associated to function calls. This is, for each event registered in the dispatcher, there is a function that will be called when the event happens in the system. It is up to the implementation to subscribe every specific event to a specific function call. If an event is desired to be ignored, the user should not associate a call for that event.

This design provides a mechanism to subscribe handler to any type of events. The dispatcher is in charge of processing all events in the queue and calls their respective function. There is also a way to unsubscribe and event in case user don’t want to be listening a particular event during a time frame.

Three are the internal modules within the dispatcher:

* Events Subscribe agent
* Events queue handler
* And the events dispatcher

All are shown in the next block diagram:



**Conclusion:**

Event oriented programming is well known as a programming paradigm where I/O concurrency is desired as it simplifies the way the programmer needs to handle the application. We are currently accustomed to program applications (either embedded or software) using a linear approach checking for changes in flags and acting upon them. However, this approach is not always the better for all applications and it’s there where we have to change to a paradigm that better suit the problem to attack.