# Domotica Packages
# Codename: Pear

Albert-Jan Nijburg

9 december 2010

## 1 Start

The idea is that it will be posibile to easily talk to different homegrown domotica systems. Like:

- Control Anything
- X10/S10
- PCDimmer
- Other homebrew domotica (send me ideas)

## 2 Basics

Just basics that we'll need. To talk to these things.

### 2.1 Input

Something that takes input. Like a button or a heat sensor or a camera looking for smoke signals over a hill in the distance.

### 2.2 Endpoints

A lamp. A Pump. A Lock. Lamps could probably have an on/off setting or a min max and a value property to specify the brightness, or the speed of a motor.

### 2.3 Connection

Connecting an (Input)Button to a (Endpoint)Lamp. So to turn on a lamp we'd need an Input with a Connection to an Endpoint. But you're getting the feeling that this is a bit too straight foreward. We need a MoronInterfaceParser the MIP.

### 2.4 MIP (MoronInterfaceParser)

A few things need to be clear. We want to more with a button than to turn on a light. We want to turn more lights on. Maybe schedule lights to go off. Maybe if we press it twice we'd want all the lights to go off.

So now we have an Input with a Connection to the MIP which has a Connection to an Endpoint. Now we need a way to tell the MIP what we want to do when the Input receives input. To do this the MIP will have modules small (SMALL!) dynamic scripts that that tell the MIP what to do.

Examples pseudocode (the idea is to use the dlr) these are just syntactical blups to give an idea of how to do this. But these are the kinds of things you'd want to be able to do.

Example 1 turn lamp on on button touch:

```
when().input("ButtonInLivingRoomNearFrontDoor").receives(Touch)
    .set().endPoint("LightUnderTheFloor")
    .at(1)
```

Example dim lamp down on on button hold:

```
while().input("ButtonInLivingRoomNearFrontDoor").receives(Hold)
    .set().endPoint("LightUnderTheFloor")
    .decrease(1)
```

Example dim lamp up on on button touch then hold:

```
when().input("ButtonInLivingRoomNearFrontDoor").receives(Touch)
     .thenWhile().input("ButtonInLivingRoomNearFrontDoor")
     .receives(Hold)
     .increase(1)
```

### 2.5 Scheduling/Timer

Set an endpoint to receive a signal at a specific time. Maybe be able to record a month of input and replay it when your on vacation.

### 2.6 MIPRecorder

Like I said before. To record on not to record that'll be the question with this one. This will enable you to record what was on when it was on and when it went of. You'll be able to replay it. guess the amout of power consumed. Prove your aliby to the police. And all the unholy things the cia'd be able to do with it.

## 3 HAL (Hardware AccessLayer)

Hardware needs to talk to the MIP. Endpoint hardware needs to talk to IEndPoint implementations in the MIP. There could in general be multiple endpoint implementation for one hardware endpoint, E.g. you could have a pulse switch and a flip switch, both could very likely work the same way as far as the hardware in concerned, but in software you'd need to handle these things completely different.

## 3.1 General outline (restrictions)

What are the messages that would be sent between the two systems. It shouldn't really matter how I see some change in the state of the hardware. We can safely say every piece hardware endpoint has an ID, and a capability.

A contact closure endpoint always needs to be mapped to `IInputEndpoint` and a Relay always needs to be mapped to a `IOuputEndPoint`.

Every piece of hardware must identify wether it has input and/or output capabilities.

## 3.2 NonBlockingIO and Asynchronous State

Output behavoir should always be implemented in a NonBlocking way we wouldn't want the whole house to wait when there's a slow reacting piece of hardware.

Input behaviour should always be implemented using Observer pattern. i.e. implementing the `IObservable<T>` interface. `http://msdn.microsoft.com/en-us/library/dd783449.aspx`. Maybe there could be one Observer observing many Inputs implementing `IObservable<T>`.