FlowerPower Project

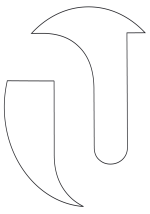
External Integration Component

Abstract

This document describes the External Integration component in the FlowerPower project, its provided and required interfaces, and its dependencies and implementation constraints.

Version History

Date	Version	Author	Description
18/05/18	1	Max Körlinge	First complete version.



IVAR JACOBSON CONSULTING

An Essential Unified Process Document

Table of Contents

1 Introduction

1.1 Document Purpose

The purpose of this document is to describe the behavior and interfaces of the External Integration component in order to:

- Establish the dependencies that exist with other components
- Describe the dependencies that exist with external hardware
- Support further development of this component
- Describe how the component can be tested

1.2 Document Scope

The scope of this document is limited to:

- Specifying the required behaviors for this component
- The specification of the interfaces that this component must implement
- Listing the required interfaces that this component depends upon
- Listing the dependencies that must be in place for the component to function
- Describe the internal structure of the component using UML

This scope of this document does not include consideration of:

- Specifications of the unit tests that are implemented to test the component
- The specification or implementation of any other components that depend upon this component or that this component depends upon

1.3 Document Overview

his document contains the following sections:

- **Required Behavior** – behavior that is required from the component.
- **Implementation Constraints and Dependencies** – constraints like implementation environment, implementation language, architectural patterns, code guidelines and standards, dependencies on external hardware, and code dependencies.
- **Provided Interfaces** – the interfaces that the component must provide.
- **Required Interfaces** – the interfaces that the component depends upon
- **Internal Structure** – describes the internal structure of the component using UML.
- **References**

2 Required Behavior

The External Integration component of the FlowerPower system handles the Plant class abstraction and all its communication with external hardware, as well as sending notifications via email. This means that for each plant connected to the system, you are able to instantiate a new Plant object, which runs in its own thread, rely on it to handle internal logic concerning care of this particular plant, and you call upon the object to interact manually with the hardware connected to the plant.

The externals which the component is responsible for handling are:

- Moist readings from the Arduino component.
- Watering (turning the pump on and off) via the Tellstick Duo hardware component.
- Sending notifications on plant status via email.

The component is required to enable:

- Getting updated moist readings from the plant at reasonable intervals
- Turning on the watering pump
- Turning off the watering pump
- Sending an email notification when the plant is drying up
- Setting what email to send notifications to
- Setting at which level of dryness the notification will be sent

Non-functional requirements on the component:

- Being able to run the component in a separate thread, to enable a parallell system handling several plants in different threads at the same time.

3 Implementation Constraints

The component is implemented in Python and must be able to run on a Raspberry Pi 3B using the Raspbian OS. Since the component is responsible for handling external communication and integration, it has dependencies on external hardware:

- A Tellstick Duo and a compatible electric switch, with the telldus-core Linux package installed on the Raspberry Pi [1].
- An electric water pump connected to the Telldus switch.
- An email account to receive notifications from.
- The Arduino component with the moist reading sensor.

The component also has several dependencies on Python libraries, listed here as Class Name : Dependency name:

- Plant: threading, time, requests
- TellstickHandler: tellcore-py
- ArduinoConnection: requests

- NotificationSender: SMTPLib

It is recommended to handle the project's dependencies using the Python tool PipEnv. Guidelines on how to use this tool is provided in Attachment 1: Virtualenv Plan at the end of this document, a guide written for the team by the author of this document. In short, the tool provides easy command line operations which handle Python's virtual environments and code dependencies.

The component is written in Python following the code standards stated in the Design plan, which can be found in the Github repository or in the final report of the project.

The component is architecturally placed within a MVC layered design pattern, where its classes belong to the Model and Integration layers, more info can be found in the Architecture description, to be found at the github repository or the final report of the project. As such, all communication has to be made by the Plant class, which belongs to the Model layer, calling on the other classes, which all belong in the Integration layer, and no method calls are allowed the other way around.

4 Provided Interfaces

This section specifies the interfaces that the component must provide in terms of their signatures and constraints.

- start() method to start the component in new thread.
- getMoistness() method to get the current moist value from plant.
- waterPlant() method, to water the plant.
- abortWatering() method, to stop ongoing watering.
- setDrynessTrigger((int) value) method, to set at which moistness value the component will send a notification.
- setEmail((string) userEmail) method, to set which email to send a notification to.

5 Required Interfaces

The component requires an IP address to request the current moistness value from the Arduino component. It must receive this value in an object from the module *requests*, and the value must be a *string*.

The component has no other interfaces it depends on.

6 Testing

The component is tested using unit tests and the PyTest framework. Unit tests are fully implemented for all methods and must continue to be so if development would continue. See the project's Test Plan provided at the github repository or in the final report of the project for more information.

7 Internal Structure

This section describes the internal structure of the component using a class diagram (Figure 1).

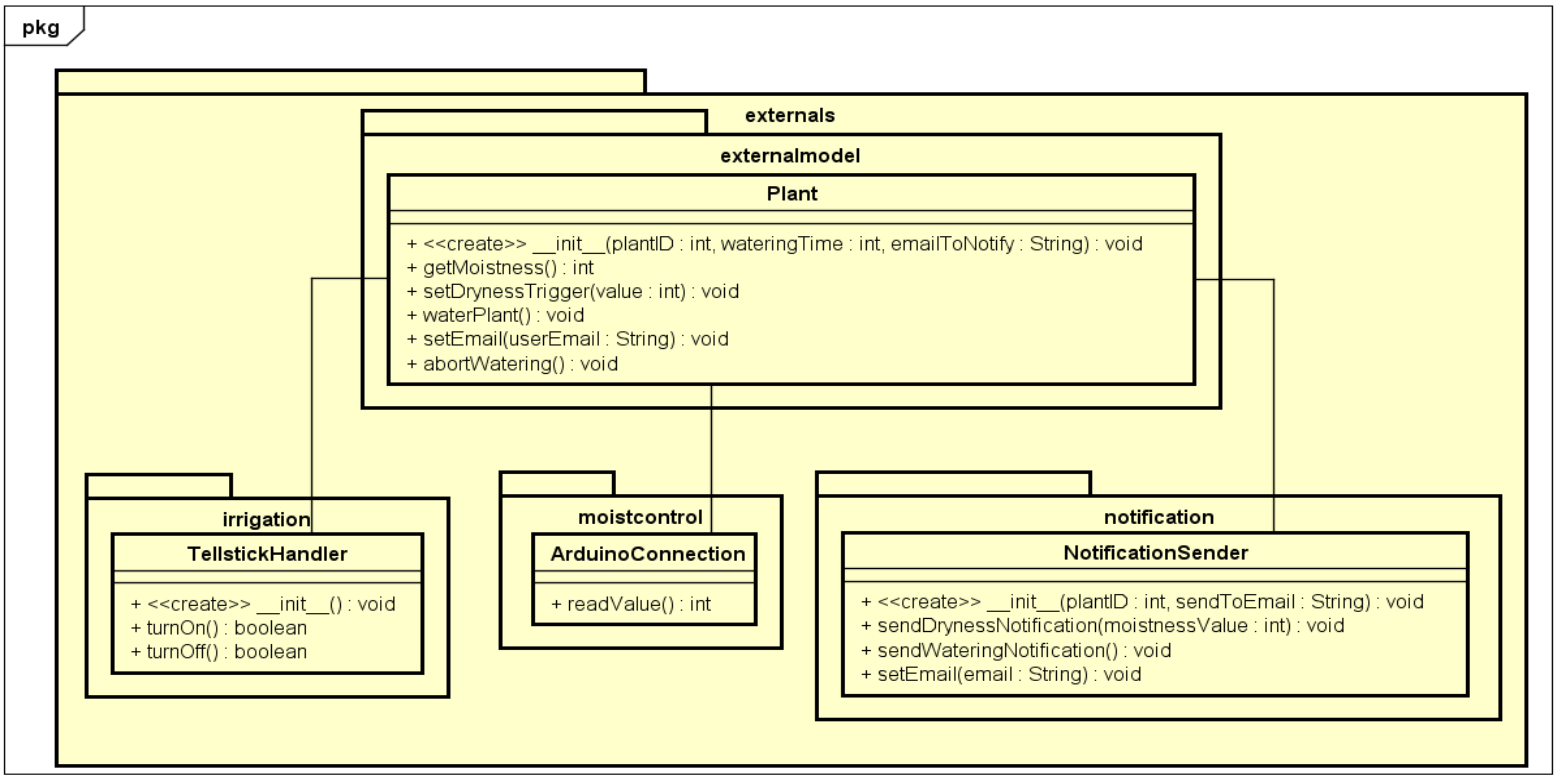


Figure 1. A class diagram describing the internal structure of the External Integration component.

Appendix A - References

[1] Telldus Developer Wiki. (2018). *Tellstick Installation Ubuntu*. http://developer.telldus.com/wiki/TellStickInstallationUbuntu (hämtad 2018-05-18).

Virtualenv-plan

Syfte

Detta dokument ska ge en gemensam plan för hur vi hanterar pythons virtual environments i projektet FlowerPower.

Introduktion

PipEnv är en python-modul som du installerar i din globala pythoninstallation för att hantera virtual environments. En virtual environment är en lokal pythoninstallation som har sina egna moduler (bibliotek). Det används till exempel när man ska utveckla ett program och vill installera bibliotek och moduler för att använda i det programmet, men man vill inte att alla dessa moduler och bibliotek ska installeras lokalt på ens “globala” pythoninstallation. Om man installerar fel grejer globalt kan det leda till problem med operativsystemet. Man kan även använda dessa virtualenvironments senare för att någon annan person enkelt ska kunna installera samma moduler och bibliotek för att kunna köra din kod på en annan dator.

Installation

Installation kräver Python3.x och pip(3), som är pythons pakethanterare. Se http://docs.python-guide.org/en/latest/starting/installation/ för installering av dessa.

Öppna kommandotolken och skriv **pip install --user pipenv**. Om du kör Linux eller Mac och har en Python2 installation på datorn (samtidigt som din Python3 installation) kan du behöva använda **pip3** som kommando i stället för pip, alltså skriva pip3 install --user pipenv.

Om du stöter på problem här, se http://docs.python-guide.org/en/latest/dev/virtualenvs/

Användning

- Navigera till roten av vår Gitmapp och pulla senaste versionen, eller klona den till din dator om du inte har den än. I mappen finns det en fil som heter *Pipfile*, vilket är den filen som PipEnv använder för att hantera moduler och bibliotek som projektet ska använda. Just nu (i skrivande stund) finns enbart modulen py.test där, men när systemet fungerar och ska köras på Raspberry Pin så ska alla moduler och bibliotek som används finnas med här, så att det går lätt att installera på Raspberryn (och andra platser man vill). Du ska inte behöva bry dig om den här filen så mycket, den ska kunna sköta sig själv.

- Kör kommandet **pipenv install** i kommandotolken. Detta skapar ett virtualenv för detta projekt på din dator.

- För att installera nya paket, skriv **pipenv install PAKETNAMN**, t.ex *pipenv install flask*. Detta installerar det nya paketet i din virtualenv, och lägger till det i Pipfilen.

- Paketet (modulen) går nu att använda i din kod. T.ex ska du nu i dina pythonfiler kunna skriva “import flask” och använda modulen.

- När du ska köra din kod måste du se till att du kör din i din “virtual environment”. Det kan du göra på två sätt:

1. Skriv **pipenv run FILENAMN.py**. Då körs programmet med hjälp av din virtualenvironment, där alla dina moduler ligger och kan användas.

2. Skriv först **pipenv shell** för att “logga in” på ditt virtual environment. Detta **måste** göras i rotmappen, där din Pipfile ligger. I ditt shell bör det nu indikeras att du har loggat in på ett virtualenv. För mig går jag från att se **| max@SUPERLAPTOP ~/ProjectGaming/gitmap $ |** där jag skriver, till att se **| (gitmap-OWbcew-k) max@SUPERLAPTOP ~/ProjectGaming/gitmap $ |**. Nu kan du navigera runt i filsystemet och köra filer med det vanliga kommandet, dvs *python SCRIPTNAMN.py* eller *python3 SCRIPTNAMN.py*, och den kommer alltid att köra scripten med modulerna i ditt virtualenv. För att logga ut ur ditt virtualenv, skriv **exit**.

- För att avinstallera ett paket, skriv **pipenv uninstall PAKETNAMN**.

För mer info om pipenv se https://docs.pipenv.org/basics/.

Commit-regler

När du har jobbat med din kod och vill commita något, **comitta inte pipfile.lock**, comitta **enbart Pipfile**. När man sedan ska merga olika Pipfiles till masterbranchen kan den ansvarige för detta behöva gå in manuellt och redigera Pipfile så att allas moduler kommer med.