FSE 2019: Demonstrator project Basura

Basura is a smart trash system which reports how full each trash can in a network of trash cans is. The system's goal is primarily to practice creating a system which spans across multiple devices, thus providing students with experience in project planning, execution, and verification. The idea was produced by students at the FSE 2017 Accra workshop, organized by the Accra Institute of Technology's Robotics Club.

Basura is designed to make it easy to know whether a trash can needs to be emptied by monitoring the trash can's contents and reporting using a variety of mechanisms, making it easy for users to tell whether a trash can needs to be emptied.

The name "Basura" was chosen because this system is designed to demonstrate how to help users manage a large amount of trash, and the word "Basura" is Spanish for "trash".

All Basura components can be pulled and deployed together by cloning the Basura metarepository and its submodules.

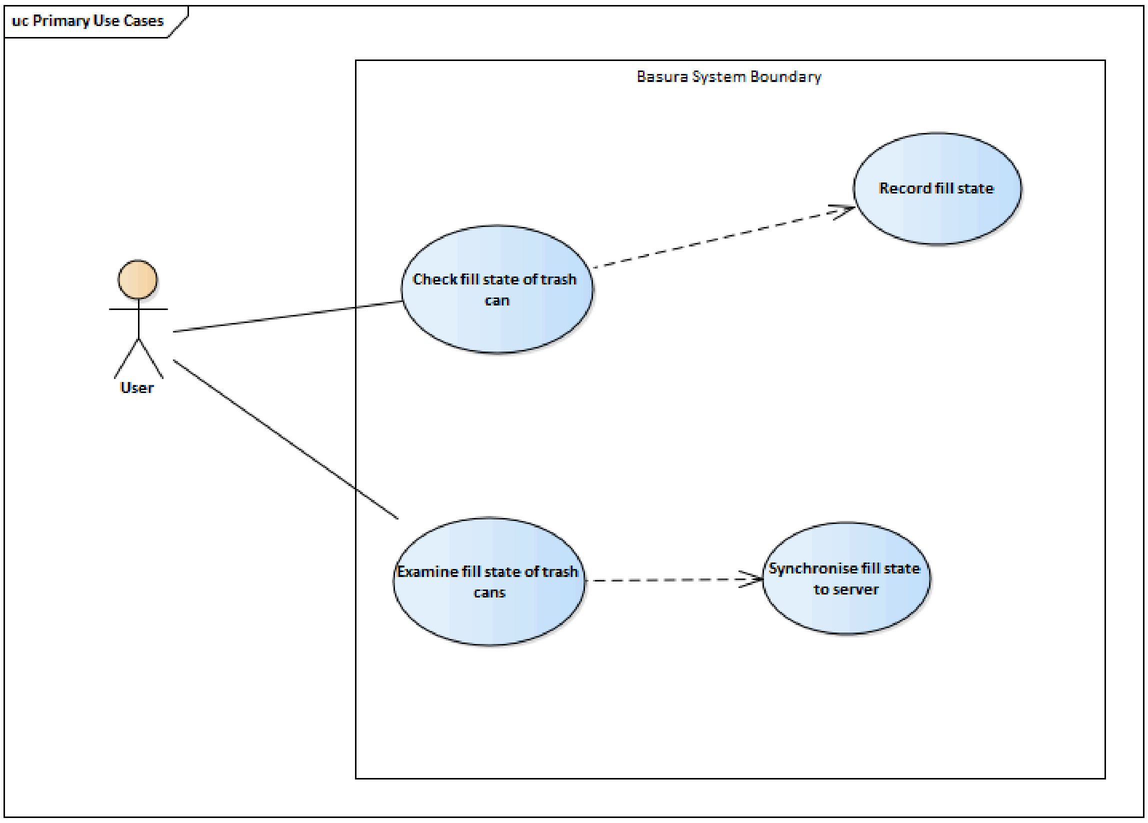
Table of contents

Views and more

This section presents several architectural views of the Basura Trash System, as well as ancillary text supporting the understanding of the view. Where possible, a view is explained via a single figure. If necessary, additional text is provided below the figure in order to clarify items which are not clearly explained by the UML.

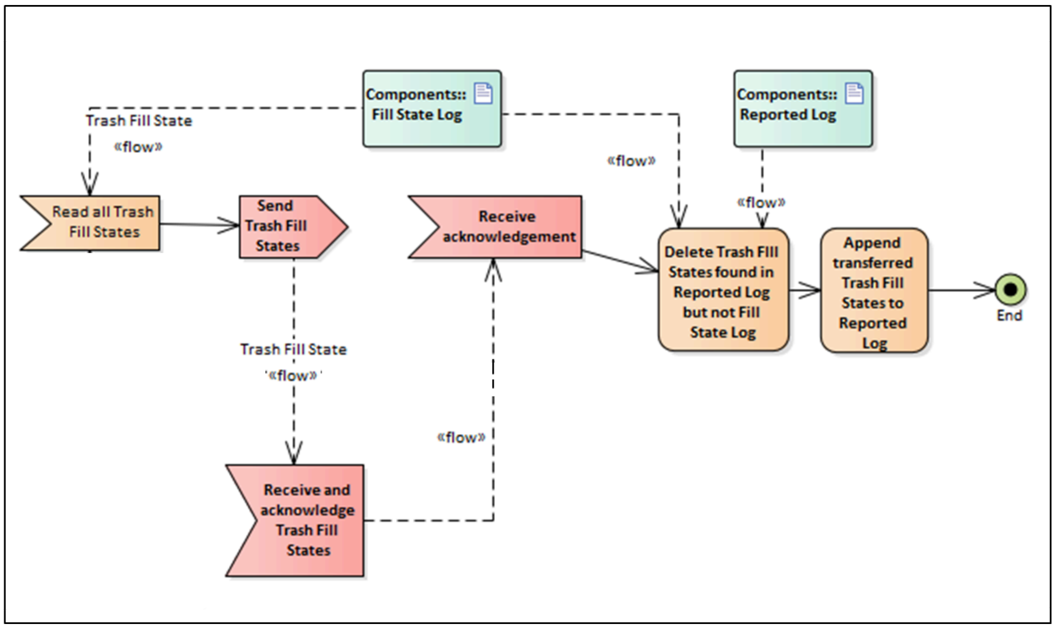
Some elements appear in more than one view. In this case, the element names are kept consistent across all views; if you encounter multiple elements with the same name, they are indeed the same element.

Use Cases

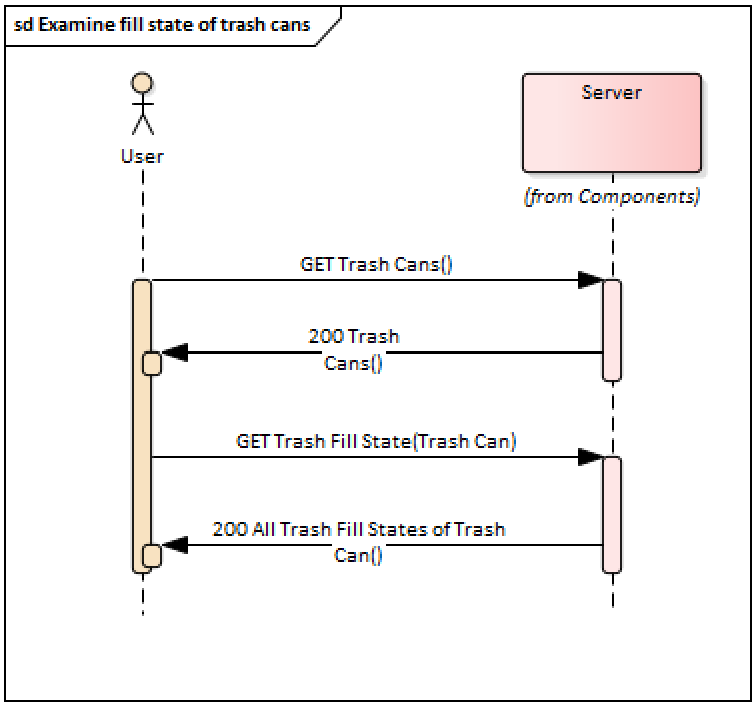

uc Primary Use Cases
Basura System Boundary
User
Check fill state of trash can
Record fill state
Examine fill state of trash cans
Synchronise fill state to server

Behavioural views

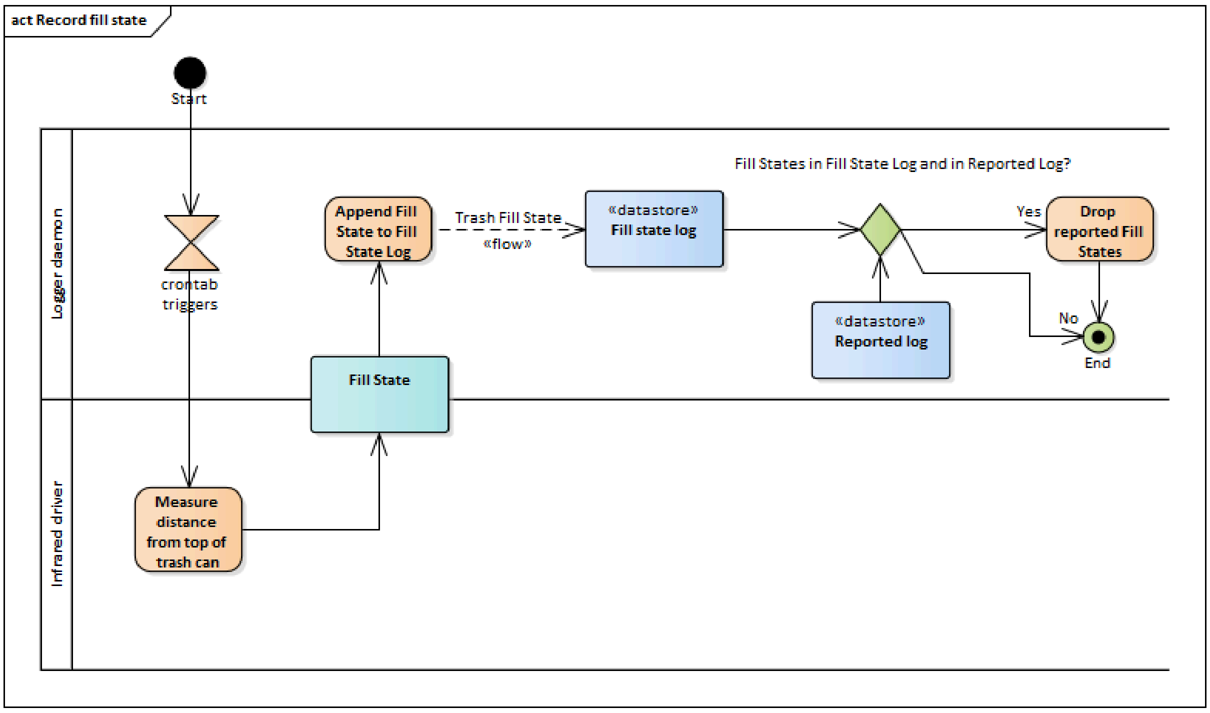
Check Fill State of Trash Can


Components:: Fill State Log
Components:: Reported Log
Trash Fill State
«flow»
Read all Trash Fill States
Send Trash Fill States
Receive acknowledgement
«flow»
«flow»
Delete Trash Fill States found in Reported Log but not Fill State Log
Append transferred Trash Fill States to Reported Log
End
Trash Fill State
«flow»
«flow»
Receive and acknowledge Trash Fill States

Request Trash Fill States


sd Examine fill state of trash cans
User
Server
(from Components)
GET Trash Cans()
200 Trash Cans()
GET Trash Fill State(Trash Can)
200 All Trash Fill States of Trash Can()

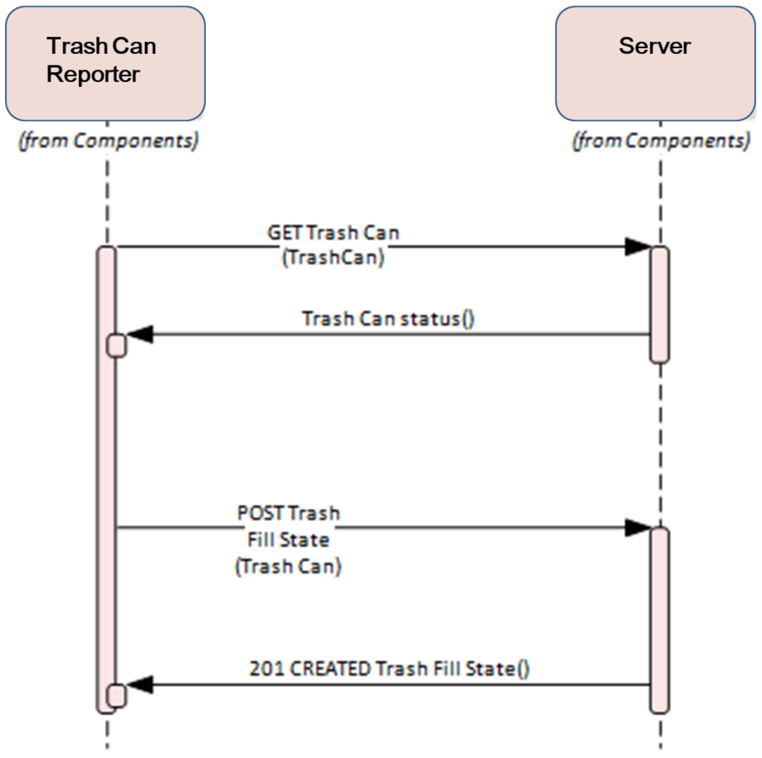
Record fill state


act Record fill state
Start
Logger daemon
Infrared driver
crontab triggers
Measure distance from top of trash can
Fill State
Append Fill State to Fill State Log
Trash Fill State
«flow»
«datastore» Fill state log
Fill States in Fill State Log and in Reported Log?
«datastore» Reported log
Yes
Drop reported Fill States
No
End

If a Fill State is found in both the Fill State Log and the Reported Log, it is deleted from both logs.

Fill States can be identified by their UUID field.

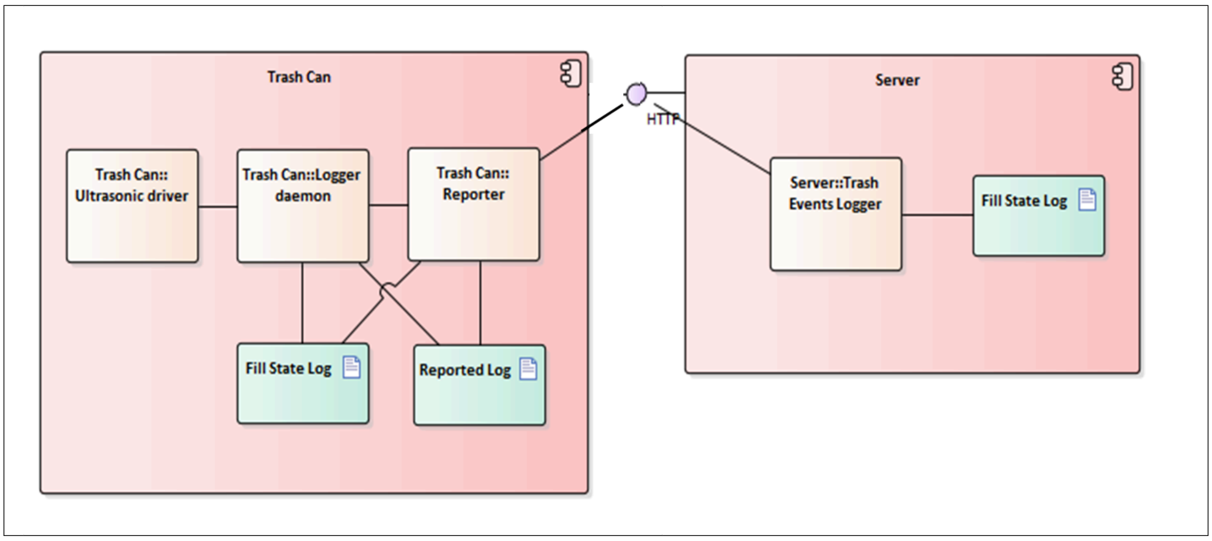
Synchronise fill state to server



INTERACTION MESSAGES
1.0 'GET Trash Can' from 'Trash Can Reporter' sent to 'Server'.
1.2 'POST Trash Fill State' from 'Trash Can Reporter' sent to 'Server'. If Trash Can does not exist, it is created.
1.1 'Trash Can status' from 'Server' sent to 'Trash Can Reporter'. If the Trash Can is already registered with the Server the next call POSTs the Trash Fill State. Otherwise the Trash Can is POSTed first.
1.3 '201 CREATED Trash Fill State' from 'Server' sent to 'Trash Can Reporter'. At this point the Trash Fill State is deleted from Server.

If 201 is not received, transaction is considered not to have taken place and it is repeated at next opportunity.

Components



All components have a Fill State Log. These contain Trash Fill States which have not successfully been reported to the next component down the line, i.e. Cell Phone or the Server, yet.

Trash Can

Reported Log

Contains all newly reported Trash Fill States.

Logger daemon

Calls the Ultrasonic Driver at regular intervals and stores the result with a UUID in a text file. At each update of the text file, it checks if a TrashEvent UUID is recorded in the file of deletables; if this is the case, that UUID is removed from the file.

Owns Fill State Log.

Reporter

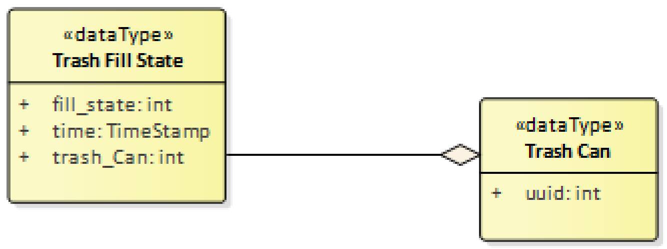
Reports all logged Trash Events. Records UUIDs of reported Trash Events in a file. Upon updating that file, deletes any entries which have UUIDs which can no longer be found in logger's file. The Trash Can's Reporter subcomponent communicates directly with the Server using the same HTTP REST interface.

Owns Reported Log.

Server

The Server receives Trash Fill States via REST. It is proposed to use a Django web app which represents Trash Fill States and Trash Cans using the Django native ORM. It is proposed to use the Django REST Framework for processing the objects which are passed to the Server, and for serializing those objects that the Server owns when requested.

Data



The Basura system serves primarily to move data from where it is observed to a sink where it can be called up. The primary data type in question is the Trash Fill State. This is an observation made by a smart Trash Can. The Trash Can entity is also represented and exchanged within the Basura system.

Trash Can

Represents a Trash Can deployment (with sensor, etc.).

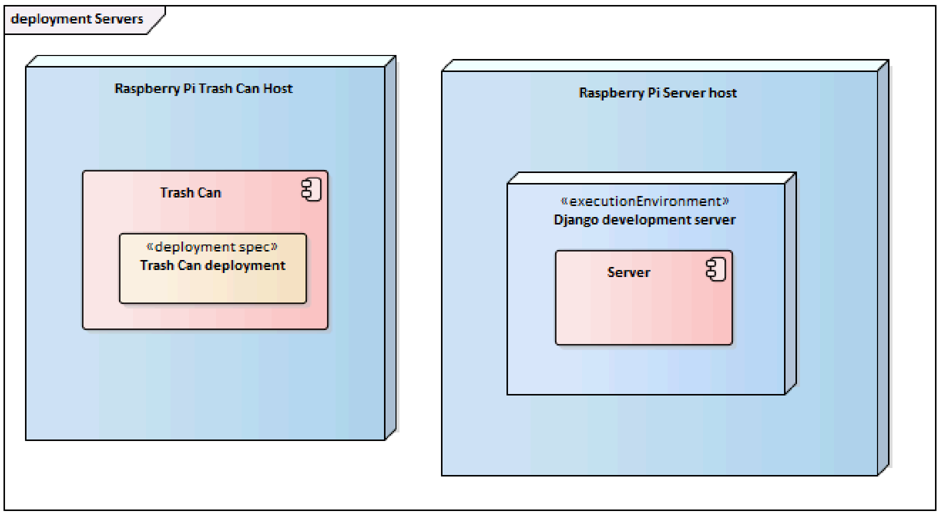
At install, each Trash Can deployment must generate and store its own Trash Can ID.

UUID is used rather than running number so that Trash Cans can be created without communication with Server. UUID is generated at install time.

Trash Fill State

fill_state : cm measured from top of trash can

trash_Can: Foreign Key to Trash Can. Each Trash Fill State is associated with 1 Trash CanDeployment



A given Basura system has 1 Server and m Trash Cans.

Raspberry Pi

One Raspberry Pi is used for the deployment of the Trash Can and another is used to deploy the server. This is not technically necessary, but it makes it clearer that we are building a distributed architecture.

Trash Can deployment

The ultrasonic driver is from the RpiAutonomousCar repository and is packaged with the Basura Trash Can.

The entire package is implemented in Python 3 and is self-contained and installed as described in its README.md file.

calibration.py generates the UUID for the specific Trash Can.

Logging intervals and the storage location of the logs is configurable, as described in the package's README. The reference deployment of Basura writes its logs to /var/log/basura/, e.g. /var/log/basura/fill-state.log.

Django development server

Normally, the Django development server would not be used; instead, the Server would be deployed e.g. to a Tomcat. However, in this case, the Django development is used for the sake of simplicity for the students.

The ORM communicates with the SQLite DB backend, also for the reason of simplicity. In a future workshop we should link to a container instead, which can also be deployed very easily, but for this workshop there are enough new concepts being introduced.

Verification

At least the following tests were planned to verify this version of Basura:

Test	**Verification method**
Trash Can can log Trash Fill States successfully	Tested manually.
Trash Can deletes reported Trash Fill States	Tested manually.
Trash Can clears Reported Log	Tested manually.
Server reports Trash Fill States from two separate Trash Cans via REST	Tested manually.
End-to-end server entity creation and retrieval tests	Implemented in the Server unit tests.