Group5 Report

ANDREW SOWINSKI,
HAOCHENG SONG,
QIHAO WANG,
ZIWEI ZHOU

This report introduces the architecture, how it works, and used open source of the IDE.

1 ARCHITECTURE

Each thing has Atlas which keeps sending tweets. Since all things with their services are connected to VSS. A program connected to VSS can listen to those tweets.
The IDE consists of front-end, back-end, and the local working directory. The front-end provides the UI and send json to back-end endpoints to use the functionalities. The backend end keeps listen to tweets and store all information from tweets in memory for front-end to present and other future uses. When running an app, the backend sends json files to corresponding Things and call services in specific order. Backend also handles interactions with file system such as saving files to and reading files from local.

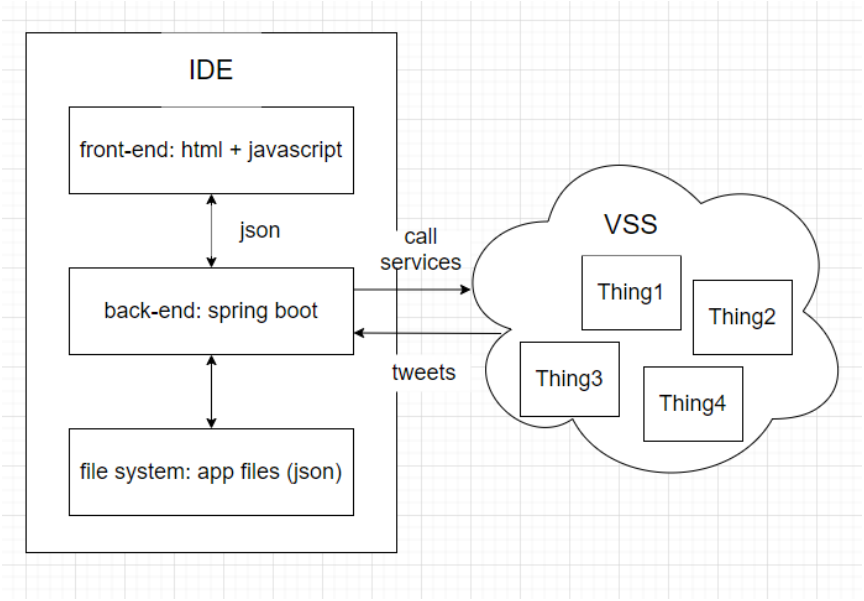

Fig. 1. Architecture of the IDE.

2 SCREENSHOTS OF TABS AND FUNCTIONALITIES

2.0 Navigator

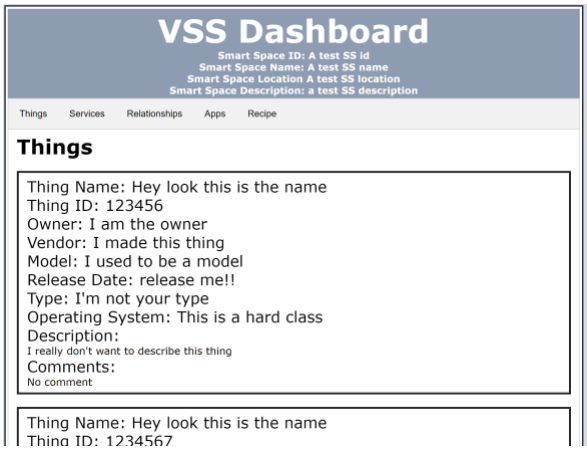

Fig. 2. navigator

2.1 Thing Tab

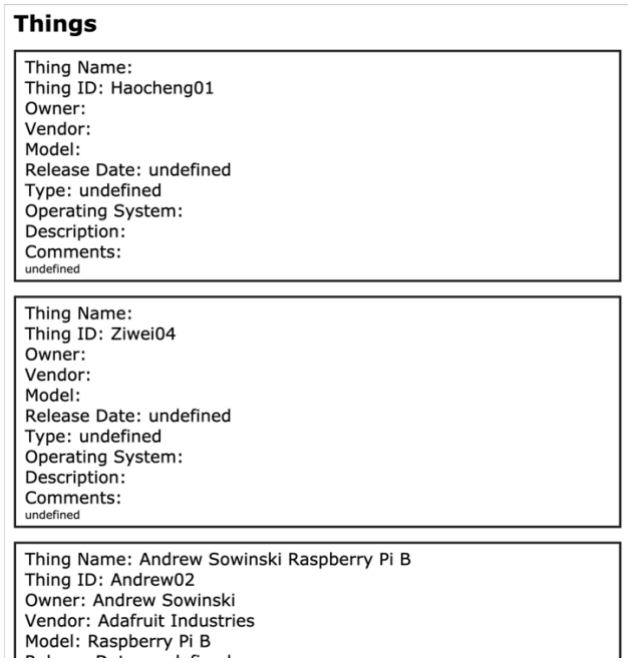

Fig. 3. thing tab

2.2 Service Tab

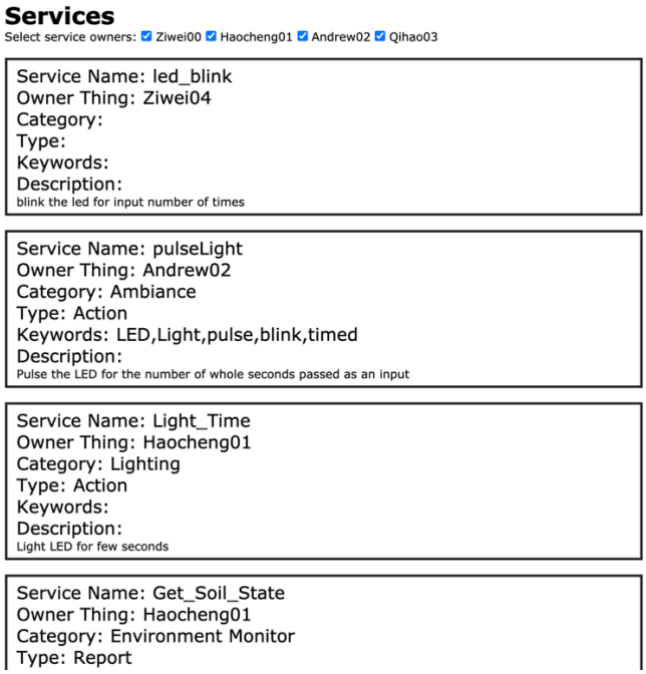

Fig. 4. service tab: showing available services can be grabbed to recipe tab

2.3 Relationship Tab

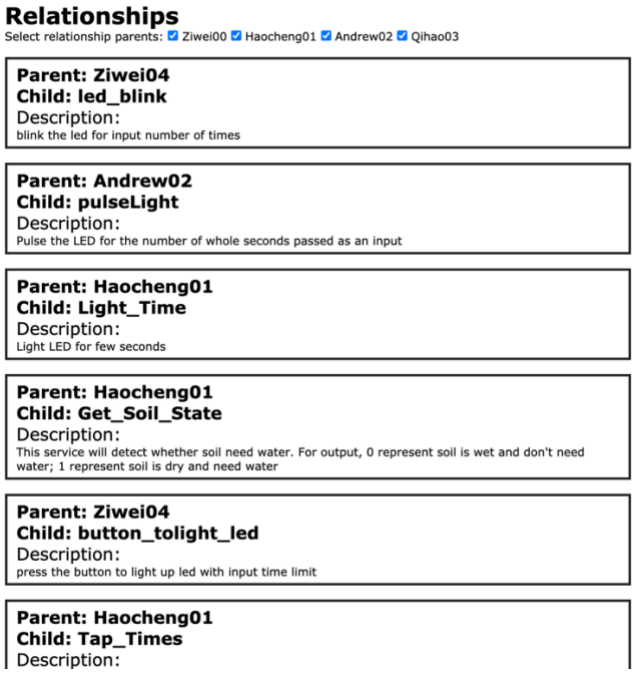

Fig. 5. relationship tab

2.4 Recipe Tab

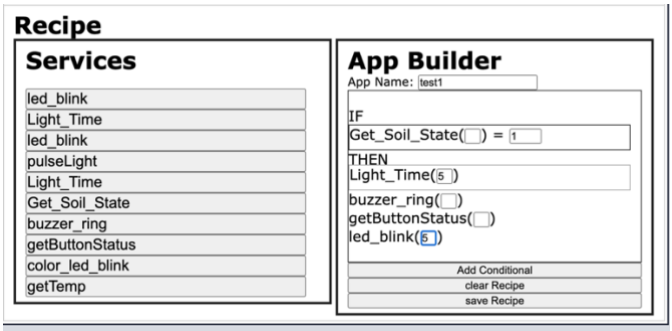

Fig. 6 recipe tab: grab services to build app

2.5 App Tab

Fig. 7 app tab: press available app to get into manage app

2.6 Application manager

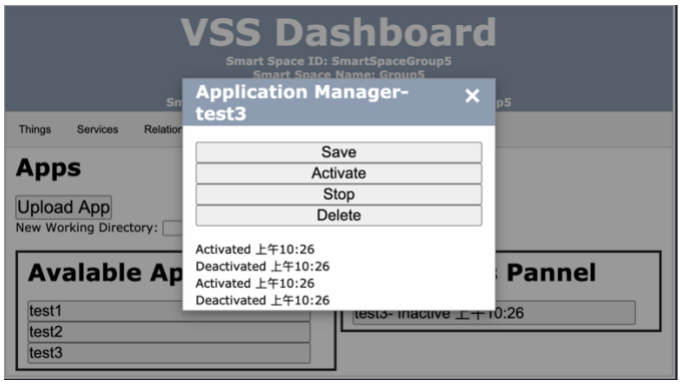

Fig.8. application manager: save to local; activate service; interrupt services which haven't been executed in a sequence of app; delete app from available apps

3 DOCUMENTATION

3.1 Front-end

Below are descriptions of each front-end function and its parameters. Some are visual in nature, some are used for front-end to front-end communication, and some are used for front-end to back-end communication.

function addThing(ID, name, owner, vendor, model, release, type, OS, description, comments): Used to add a Thing to the "Things" tab of the UI.

function addService(name, category, type, keywords, description, owner): Used to add a Service to the "Service" tab of the UI.

function addRelationship(parent, child, description): Used to add a Relationship to the "Relationships" tab of the UI.

function addAvailApp(name, data): Adds an app unit to the available apps menu.

function addStatusApp(name, data): Adds an app to the status panel as "active".

function activateApp(): Activates an inactive app.

function stopApp(): Deactivates an active app.

function stopAppFromBackend(appname): Allows the back end to update the UI to reflect that an app is no longer running.

function updateHistory(historyJson): Updates the app history shown on the application manager.

function saveApp(): Saves an app to the working directory as a json through communication with the back end.

function deleteApp(): Deletes an app from the UI and from the working directory (using the back end).

function filterRel(): Filters relationships based on selected parents.

function filterServ(): Filters services based on selected owner device.

function onLoadFunc(): Does UI setup on load.

function setSmartSpaceInfo(ssID, ssName, ssLoc, ssDescrip): Updates the smart space information in the header.

function newWDSubmit(): Sends the new working directory to the back end.

function allowDrop(ev): Drag and drop support function.

function drag(ev): Drag and drop support function.

function drop(ev): Drag and drop support function for Recipe tab.

function drop2(ev): Drag and drop support function for Services tab.

function addConditional(): Adds a conditional block to the Recipe builder.

function clearRecipe(): Clears the recipe builder.

function saveRecipe(): Saves a recipe as an app to the Apps tab.

function makeModal(caller): Displays the modal pop-up window.

function removeOldApps(): Removes apps from the App status window that have been inactive for 5 or more minutes

function oneSecondCycle(): Used to check for updates once per second

function doClickAction(tempThis): Click and double-click support function for Apps tab.

function doDoubleClickAction(tempThis): Click and double-click support function for Apps tab.

function handleClick(tempThis): Click and double-click support function for Apps tab.

function handleDoubleClick(tempThis): Click and double-click support function for Apps tab.

function openTab(evt, tabName): Opens a specific tab based on the parameters.

3.2 Back-end

function getTweetsFromPi(): Implemented udp socket to listen to multicast address, filter out irrelevant tweets by space id, the filter maintained thinglist to monitor every thing's tweets and if received duplicated tweet, the status to receive tweet of this thing will be set to "stop". Then parse our filtered tweets to corresponding "thing", "service", "relationship" objects and save them into hashmaps.

Entity class:

Service: It's basically mapping to Service from our Pi, it include all metadata of service in pi, include name, thing_id, entity_id, space_id, vendor, api, type, appcategory, description, keywords and ip address of its running Pi.

Relationship: It's basically the relationship of Pi, contains include parent, child, description about service .

Thing: This class represent Things of Our Pi, it include thing_id, space_id, name, model, vendor, description, owner, OS.

App: This class represent App, which is the combination of services. It contains App's filename, appName, workingDirectory, units(Which is an array of another class) and originJson(Which is the originally json of this app). There is a inner class Unit represent every single unit of App.

AppForWeb: It's a App class for frontend to show, only include appName and originJson.

Controller:

public Thing[] initiationOfThings(): Listen on GET method on uri "/getThing", will initiate all things in to frontend.

public Service[] initiationOfServices(): Listen on GET method on uri "/getService", will initiate all services in to frontend.

public AppForWeb[] initiationOfApps(): Listen on GET method on uri "/getApps", will initiate apps from local file.

public void activateApp(@RequestBody String request): Listen on POST method on uri "/activateApp", will activate a specific app.

public void saveApp(@RequestBody String request): Listen on POST method on uri "/saveApp", will save a specific app.

public void deleteApp(@RequestBody String request): Listen on POST method on uri "/deleteApp", will delete a app from local file if possible.

public void changeDirectory (@RequestBody String request): Listen on POST method on uri "/changeDirectory", will change default saving and working directory for apps.

Util method:

public static void getTweetsFromPi(): Get tweets with select space_id from Pis, and loaded their metadate into memory.

public static void scan(): Scan working directory to load possible apps

4 OPEN SOURCE USED

Spring Boot.
It is one of the most popular open source projects in the cs domain. It is mainly for development of Java backend of web application with support of many modules such as spring data.
The usage for our project is to build endpoints for front-end so that front-end can send requests to run real functionalities.