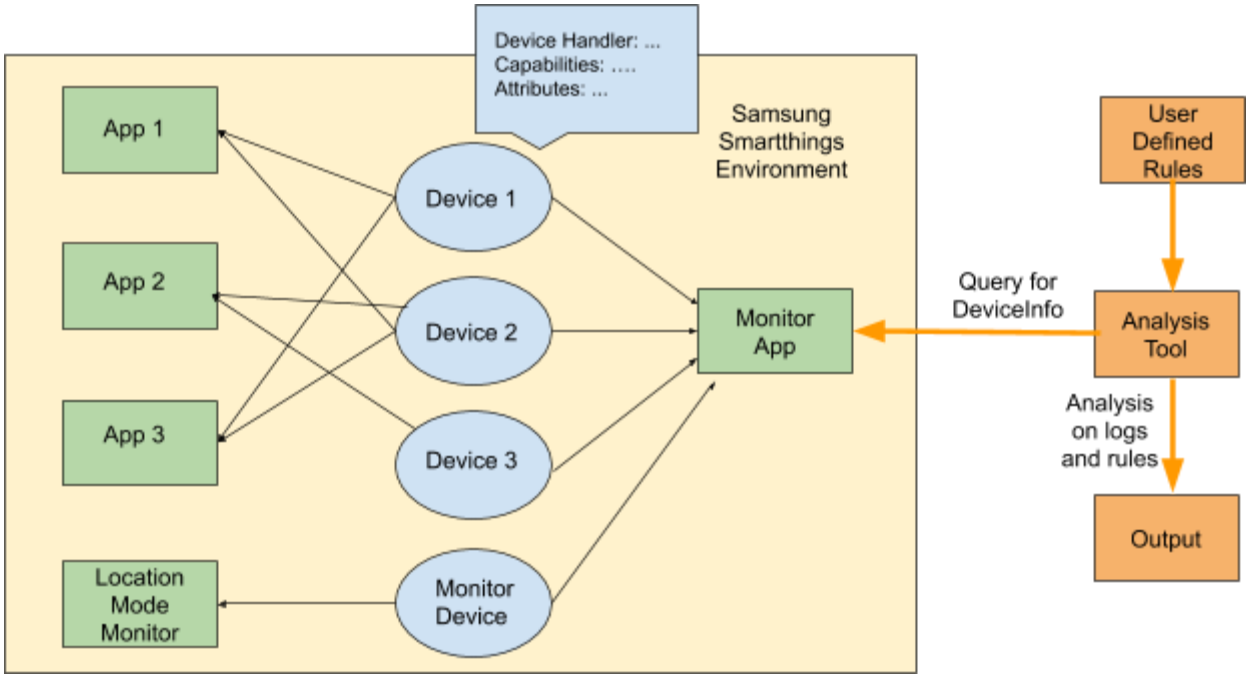
System Architecture:

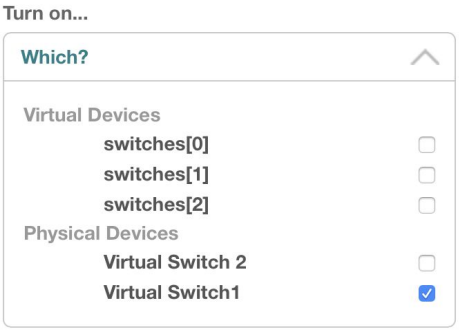


System On Samsung Smartthings:

On the Samsung Smartthings system, we are able to create a virtual environment that imitates the actual interactions of the devices in real life. In the system, first we have multiple devices each have their own corresponding device handlers to handle the method call on the device. Since we are simulating on the virtual devices, the device handler for each is used with the prefix "Virtual". Each device handler defines the capabilities of the device, and the capabilities of the device have attributes that can be changed by other apps. For example, a switch in our environment would have a handler called "Virtual Switch" and the handler would define the capability of switch, which has an attribute called switch that can be "on" or "off".

Each device can be added to apps that ask input for the device with capabilities it has, as shown in the image(left) where it is asking for a device with switch capability to be added to the app. We can input devices to use the app through this preference section when we run simulate (right).

```
preferences {
    section("Turn on...") {
        input "switches",
        "capability.switch",
        multiple: true,
        required: true
    }
```



Our system monitors the interaction between these apps and analyzes for abnormal behavior from the logs it generates. It achieves this through an app form on the Samsung Smartthings system to ask for input with the corresponding capabilities as the devices we want to monitor in the system. When a device is added to the app, the app automatically has access to all the events that have happened on the device, and all the attribute states the event has since a certain time. Such information for all the devices is stored on the Samsung cloud, and the rest of our monitor system requests this information through our Smartthings monitor app as a medium to perform analysis.

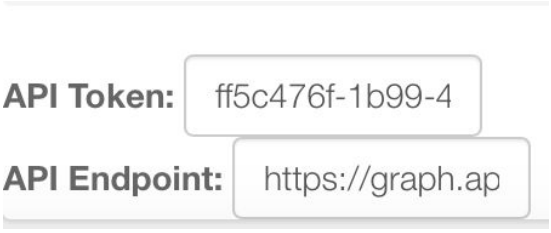
Since it is not possible to obtain the mode for current location directly from our monitor app, we can create a separate app and a device to monitor the location mode changes in the system. When the location mode of a system is changed, the app would send to the device the information regarding the new mode that the location is changed to, which we store in the device's state. We then add the device to our monitor system to obtain the state information of the mode changes. If no such location mode changes happened in the system, our monitor is able to provide the current location mode for the system to be used for analysis.

System Locally:

Finding Conflicts:

Locally, our analysis tool is separated into two parts: obtaining the information from the cloud and performing analysis. Samsung Smartthing apps supports Oauth requests locally and can be connected through the app's API-Key and API-endpoint, which is obtained when running the app under simulation on the IDE (left). We send GET requests for device information to our

monitor app, and our monitor app is able to handle these requests through the “mapping” feature (right).




```
mappings {
    path("/endpoint") {
        action: [
            GET: "handlerURL"
        ]
    }
}
```

We first request the monitor app to get all the devices in our system and their corresponding ID’s. With their ID we can query from Samsung Smartthings about events and states of the device. All the device information the Smartthings cloud responded are stored in Json files as shown below:

```
Device:
{"id": "9793402f-fcb7-42af-8461-da541b539f01", "name": "Monitor Device",
"label": null, "capabilities": [{"name": "Execute", "attributes": ["data"],
"commands": ["execute"]}]}

Event:
{"device": {"id": "abeafef6-7372-4347-bab6-4f485b8fb2d7", "name": "Virtual
Switch 2", "label": null}, "source": {"enumType":
"smartthings.message.event.EventSource", "name": "DEVICE"}, "date":
"2020-04-02T04:28:38Z", "deviceId": "abeafef6-7372-4347-bab6-4f485b8fb2d7",
"name": "switch", "stringValue": "on", "desc": "Virtual Switch 2 switch is on"}

State:
{"state": "switch", "date": "2020-04-02T00:30:15Z", "value": "on"}
```

Utilizing the information in these JSON files, we check abnormal behaviors among our virtual system in three ways: direct conflicts between apps, abnormal state changes between apps, and places where important devices get modified. We observe that a normal app behavior should have an “APP_Command” event followed by a “DEVICE” event, where the former being described as “App x is called on y with z” and the latter as “State of y is changed to z”. If the two events are not consistent (they do not have the same z) or not immediately following each other, we know that we have an abnormal state change. We also check if the app is immediately called from a previous state change to change the device to a different state, which would show the app has a direct conflict with a device state change. Finally, it is still possible for the apps to have direct conflicts also behave normally, thus we scan through all the app events to make sure none of the two are happening immediately following each other.

Date	Source	Displayed Text
2020-04-05 10:52:35.573 PM UTC *5 days ago*	DEVICE	Virtual Switch 2 switch is off
2020-04-05 10:52:35.365 PM UTC *5 days ago*	APP_COMMAND	Big Turn ON sent off command to Virtual Switch 2

We combine all the events from each device in the system together, and sort them based on the time of event being performed. During the sorting, we also add all the logs of location mode changes from our previous location mode monitor device to the list. We search for the events where the important devices have a state change, and logs the last five events that happened before the state change to see what leads to that behavior. We generate all of our results as an output log in a file specified when running the program.

Analysis on User Defined Rules:

After obtaining all the events in the system as well as a log of all location mode changes, we can analyze all of these events according to simple rules provided by the user. Currently, the rules supported are only DO rules and DONT rules, the syntax of such rules are shown below:

```
DO/DONT $deviceMethod THE $device WHEN $attribute OF $devicename IS $value AND $attri.....
DO/DONT $deviceMethod ....  WHEN LOCATION MODE IS $mode
DO/DONT SET LOCATION MODE TO $mode WHEN ...
```

For the rules, the portion before WHEN specifies the behavior under the conditions defined by the portion after WHEN. For DONT rules, the behavior should not happen when all the conditions are satisfied, while for DO rules the behavior should happen. The key words are used in **capital** to serve as tokens for our parser. For example, under the rule “DONT unlock THE Door WHEN LOCATION MODE IS Away AND alarm OF Smoke Alarm IS siren”, the device “Door” should not call “unlock” method under the condition of Location Mode : Away and smoke alarm’s alarm attribute is siren. We achieve this through keeping a state of all the devices of the system while iterating through all the events and location mode changes provided from analysis before, we check whether any rule is violated along the way. For the DO rules, we say that the rule is violated if the asked behavior did not happen within 2 seconds of the last event that satisfies all the conditions. For all the rules, if there is no log for the current state of the specified devices (ex: last modified is too long ago), we assume they are not violated.

We have also added the feature for rules regarding time, for DO rules, it is possible to add AFTER $time $timeunit to specify the action to be done after some time when all the conditions are being met. For DONT rules, it is possible to add FOR $time $timeunit to restrict the action to

not be done when all the conditions are being met. For all the conditions, we can also specify with the FOR syntax, so that the conditions are only being met after the state value matching for the certain time. An Example is shown below:

```
DONT unlock THE Door FOR 3 SECONDS WHEN LOCATION MODE IS Away
AND alarm OF Smoke Alarm IS siren
DO SET LOCATION MODE TO Home AFTER 1 MINUTES WHEN
lock OF Door IS unlocked FOR 3 SECONDS AND LOCATION MODE IS Away
```

We added the time feature by keeping the last time the state is modified in our state of all devices. When checking our rules, when checking each condition with “FOR” a duration of time for a certain state, we first check if the values for the states match the condition. If it matches, check if the last time state modified to the matching value is before the event time specified by the date minus the duration. When doing DONT do something for a duration, we check the event to see if it is done within the duration after the time for the last satisfied condition. When doing DO something after a duration, we add the duration as an offset and check if the event happens after the time with added offset. For durations, the only supported units are SECONDS/MINUTES/HOURS, as there are often rarely any rules should be applied for a longer duration.

We also added OR rules to specify conditions like A OR B, with which we only need one of them to be satisfied, and we would always report the first satisfied condition if the rule is violated. When there is a combination of OR and AND’s, AND would have a higher precedence. If it is intended for OR to have a precedence, we would need to break the rule into multiple smaller rules. An example of separation for or rules and other uses is shown below.

Ex:

DONT C WHEN A OR (B AND D) => DONT C WHEN A, DONT C WHEN B AND D
DONT C OR D WHEN A => DONT C WHEN A, DONT D WHEN A
DONT C AND D WHEN A => DONT C WHEN A AND D, DONT D WHEN A AND C
DONT A AFTER t WHEN B FOR x => DONT A WHEN B FOR x+t
DO A FOR t WHEN B FOR x => DO A WHEN B, DONT A WHEN B FOR x+t

As a final note, our parser is fairly simple and fragile. Currently, we disallow extra white space characters and line breaks in rule files. We also disallow individual conditions appear more than once in the rule, such as DONT A WHEN B AND B.

Example Interactions:

We created a simple system based on the potential dangerous interaction where a hacking of the oven may cause the smoke alarm to rang and open the household door for intruders. We have created two apps that do the minimal interaction of recreating the scenario, and ran the simulation. We specified the door as the important device in the household, with which our analysis tool is able to generate the log that demonstrates the potential dangerous interaction between the oven and the door.

```
All the indirect conflicts:
All the important device change log:
    Object:Door
        Change Number: 0
            Device: Door's lock state changes to locked
            App: Oven and Smokealarm changes Oven with setMachineStatecommand
            App: Oven and Smokealarm changes Oven with setMachineStatecommand
            App: Oven and Smokealarm changes Smoke Alarm with sirencommand
            Device: Smoke Alarm's alarm state changes to siren
            App: Opens door when smoke detected changes Door with unlockcommand
```

We have also inserted simple rules for analysis in the system, which we have our desired log shown below:

Rules:

```
DONT unlock THE Door WHEN LOCATION MODE IS Away AND alarm OF Smoke Alarm IS siren
DO SET LOCATION MODE TO Home WHEN lock OF Door IS unlocked AND LOCATION MODE IS Away
```

Result:

```
All the DONT rules that are violated:
    1. App: Opens door when smoke detected calls unlock method on Door
    Under the condition of:
        The location mode is: Away
        The attribute alarm for Smoke Alarm is siren
All the DO rules that are violated:
    1. Not Changing location mode to: Home
    Under the condition of:
        The attribute lock for Door is unlocked
        The location mode is: Away
```

Rules: (There should not be line breaks within 1 rule)

```
DONT unlock THE Door FOR 3 SECONDS WHEN LOCATION MODE IS Away
AND alarm OF Smoke Alarm IS siren
DO SET LOCATION MODE TO Home AFTER 1 MINUTES WHEN
lock OF Door IS unlocked FOR 3 SECONDS AND LOCATION MODE IS Away
```

Result:

```
All the DONT rules that are violated:
    1. App: Opens door when smoke detected calls unlock method on Door FOR 3 SECONDS
    Under the condition of:
        The location mode is: Away
        The attribute alarm for Smoke Alarm is siren
All the DO rules that are violated:
    1. Not Changing location mode to: Home AFTER 1 MINUTES
    Under the condition of:
        The attribute lock for Door is unlocked for 3 SECONDS
        The location mode is: Away
```

To show the example of a system with direct conflict, we have also created a system where we have two apps trying to turn a switch on and off at the same time. Our analysis successfully captured the conflicting interaction as shown in the log below:

```
All direct conflicts from monitor:
    Component0
        App:Leave Door Light On Device: Virtual Switch 2
        App:Turn off all lights Device: Virtual Switch 2
```

Future Work:

While user provided rules can help testing for the majority of possible indirect conflicts in the system such as two apps may have conflicting interests such as one turn on AC while another turn on heater at the same time, we are still trying to find a more automated analysis scheme for these conflicts without the need of user rule input. Currently, we only support simple DO and DONT rules for analysis, and we would like to expand the possible rules we can have in the system even more.

Furthermore,we would like to explore more abnormal interactions and possibly build a virtual monitor device that is able to interrupt or handle abnormal behavior when it occurs, the immediate approach is to have a device sending SMS messages to the users when any rule is violated or any abnormal interaction is encountered in the system.

Code On Github:

System on Samsung Smartthings:

Monitor App: MonitorApp.groovy

Location Monitor App: LocMonitorApp.groovy

Location Monitor Device: DeviceHandler.py

Local System:

Communicate with Samsung Cloud: getDeviceInfo.py

Analysis Infrastructure: Analysis.py

Parser On Rules: Parse.py

Analysis on User Defined Rules: Rules.py

Running Analysis: getlog.py

Getting Device Logs only: test.py

Sample Systems:

Smoke Alarm: Interactions/Smokealarm/*

Direct conflict between switches: Interaction/Switches/*