SRI-Delhi-IIT-Delhi

Smart Home with connected devices

[*Scene-based device management*]

Internet of Things (ELV780)

Project Name	***Smart Home with connected devices***	
Date	***2018-04-18***	
Author	**Harshit Gupta- 2017EET2303** **Amritanjan Kumar - 2017JOP2313** **Rahul Panwar- 2014MT10601**	Course Coordinator: Dinesh Kumar
Institute	**IIT-Delhi *(Electrical Engineering)***	

Contents

1 SW Development Plan

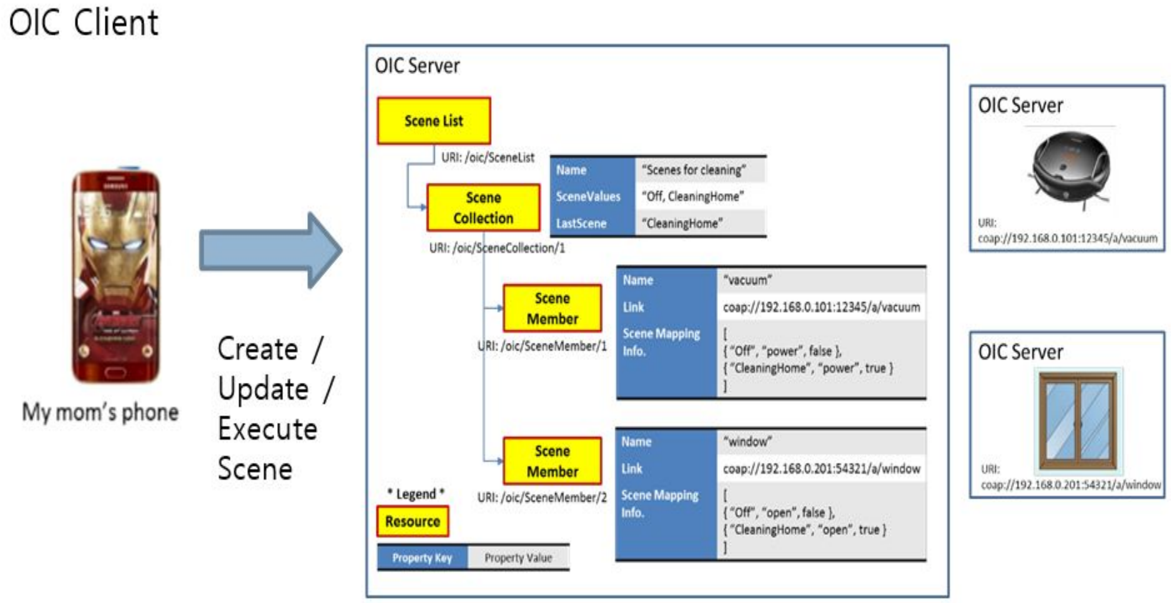
1.1 Project Overview

Objective and Project Scope

The objective of this project is to create an interaction between various smart connected devices whose states can be changed based on different situations/scenes. Situation/Scenes can be like going office, TV mode, watching movie or it can be selected manually.

The applications of this project are very wide. In an iotivity based smart home, it is always desirable to control IoT devices and their states based on certain conditions. **Scene manager** which is available in IoTivity comes into picture to shape that task into a code and execute.

Below is the simple overview of the project. Here in this figure vacuum and window is used as resource servers.



Major review items
Discovery Manager
Resource encapsulation
Scene Manager: 1. Scene server 2. Scene client
Resource servers 1. Light 2. Fan

1.2 Assumptions, Dependencies and Constraints

Item	Assumptions, Dependencies and Constraints	Remarks
1.	Header files and libraries	Necessary header files needed to compile .cpp modules
2.	Scene Collection	Limited to Living Room only
3.	Scenes	Limited to three use-case: 1. Going office 2. TV Mode 3. Manual
4.	Resource servers	Limited to two resources: 1. Light 2. Fan
5.	SceneActions	Limited to two: 1. Light on/off 2. Fan speed [0-100]
6.	Iotivity stack	Machine running is a IoTivity node

1.3 Roles and Responsibilities

Institution		Roles and Responsibilities	Person in Charge	Department
IIT-Delhi	Software Requirements Analysis	**Developer** **Software Requirements Analysis** Verifying requirements and performing analysis on requirements;	Amritanjan Kumar - 2017JOP2313	EE
	Software Architecture	**Developer** **Software Architecture -**Mapping the requirements into Architecture	Rahul Panwar -2014MT10601	MT
	Software Design	**Developer** **Software Design** Mapping of SW Architecture into Design	Harshit Gupta - 2017EET2303	EE
	Software Development	**Developer** **sceneclient.cpp** Modification in client configuration and development	Harshit Gupta - 2017EET2303	EE
		Developer **sceneserver.cpp** Creating a iotivity server providing scene manager services	Harshit Gupta - 2017EET2303	
		Developer **fanServer.cpp, lightServer.cpp** Iotivity resource severs	Harshit Gupta - 2017EET2303	

1.4 Development Plan

1.5 Development Schedule

Estimated Project Period	2018.03.14 ~ 2018.04.18
Project Team Size	03
Estimated Man Months	03*1 = 3 months

1.6 Development Environment

Item	Development Environment	Remarks
Program Languages	C++ with scons	Main modules are in Cpp and IoTivity API are in C
Compiler, Build	GCC 4.3.1	gcc available on ubuntu 16.04 is used to compile and run
Target Kernel	Linux (Ubuntu 16.04)	Most used.
Word Processor for Document Creation	MS Word, Libreoffice	
Configuration Management	Using Ubuntu inbuilt functionality	

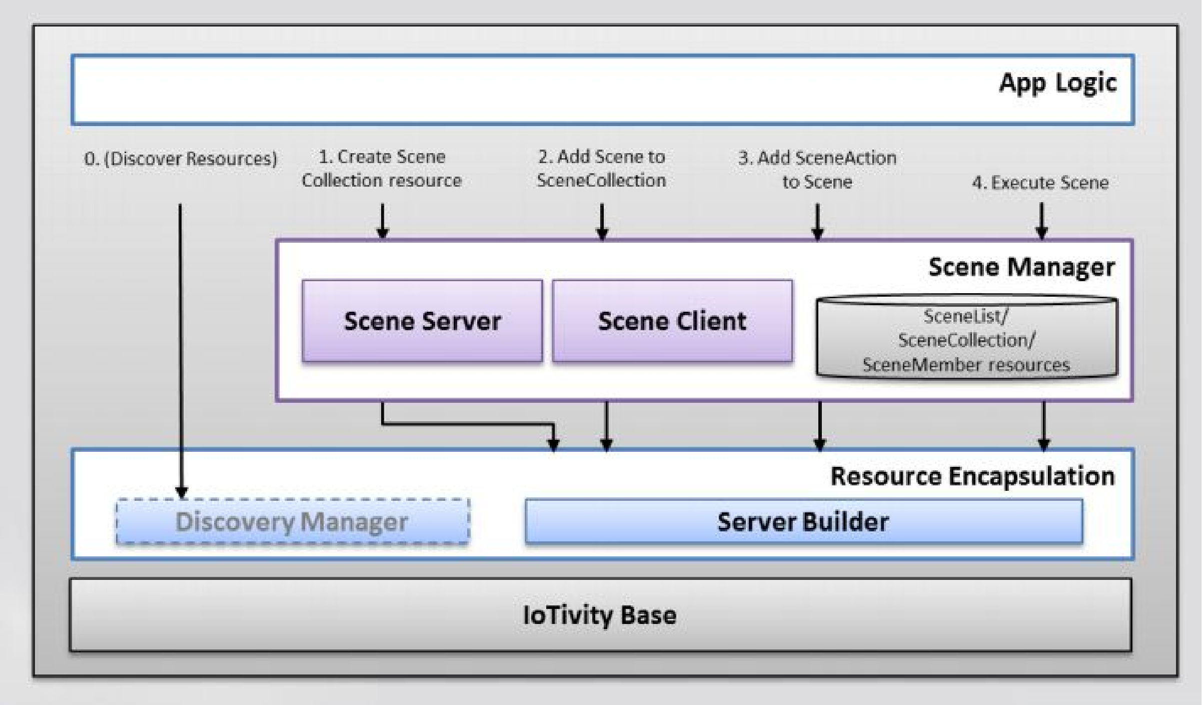
2 SW Requirements Specification

2.1 Major Functional Requirements

No	Requirement Id	Function Requirement Name	Description
1	discoverResource()	Discover Resource servers	The SW must be able to search for available server.
2	Configure	Configure Platform	The SW must comply with platform configuration.
3	registerResource	Register available resources	The SW must be able to connect and register available resource servers.
4	createScene()	Create a scene	The SW must be able to create a scene.
5	addScene()	Add scene to scene collection	The SW must be able to add created scene to scene collections in scene manager.
6	addSceneAcio()	Add actions to scene	The SW must be able to add some actions to created scene.
7	executeScene()	Execute the scene actions	The SW must be able to execute the scene and perform its actions.
8	exit()	Exit the program	The SW must be able to release resources and exit the main program gently.

3 SW High & Detailed Level Design

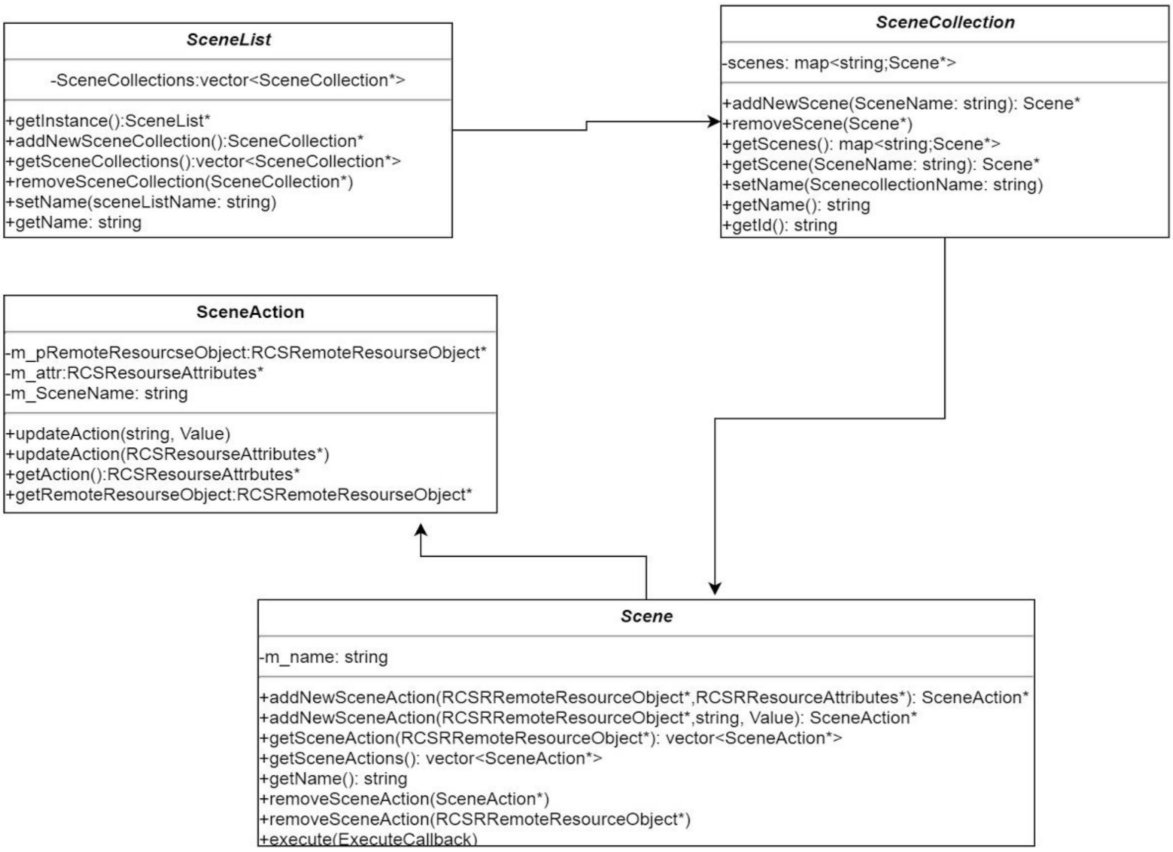
3.1 Overall Architecture



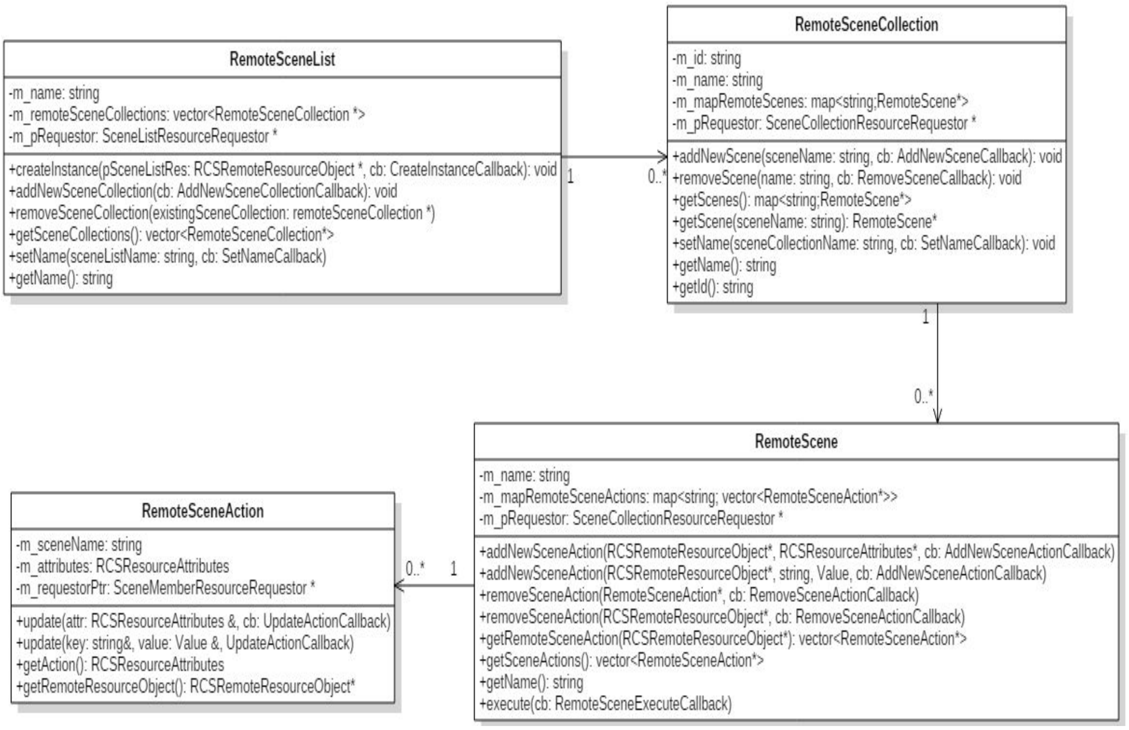
3.2 SW System Operation Design

3.3 {DesignID} Structure Diagram

The class-diagram of different module interaction is presented below (**Drawn using draw.io->UML->class_diag.xml**) :


SceneList
-SceneCollections:vector<SceneCollection*>
+getInstance():SceneList*
+addNewSceneCollection():SceneCollection*
+getSceneCollections():vector<SceneCollection*>
+removeSceneCollection(SceneCollection*)
+setName(sceneListName: string)
+getName: string
SceneCollection
-scenes: map<string;Scene*>
+addNewScene(SceneName: string): Scene*
+removeScene(Scene*)
+getScenes(): map<string;Scene*>
+getScene(SceneName: string): Scene*
+setName(ScenecollectionName: string)
+getName(): string
+getId(): string
SceneAction
-m_pRemoteResourcseObject:RCSRemoteResourseObject*
-m_attr:RCSResourseAttributes*
-m_SceneName: string
+updateAction(string, Value)
+updateAction(RCSResourseAttributes*)
+getAction():RCSResourseAttrbutes*
+getRemoteResourseObject:RCSRemoteResourseObject*
Scene
-m_name: string
+addNewSceneAction(RCSRRemoteResourceObject*,RCSRResourceAttributes*): SceneAction*
+addNewSceneAction(RCSRRemoteResourceObject*,string, Value): SceneAction*
+getSceneAction(RCSRRemoteResourceObject*): vector<SceneAction*>
+getSceneActions(): vector<SceneAction*>
+getName(): string
+removeSceneAction(SceneAction*)
+removeSceneAction(RCSRRemoteResourceObject*)
+execute(ExecuteCallback)

Class diagram for simple client applications is as below:


RemoteSceneList
-m_name: string
-m_remoteSceneCollections: vector<RemoteSceneCollection *>
-m_pRequestor: SceneListResourceRequestor *
+createInstance(pSceneListRes: RCSRemoteResourceObject *, cb: CreateInstanceCallback): void
+addNewSceneCollection(cb: AddNewSceneCollectionCallback): void
+removeSceneCollection(existingSceneCollection: remoteSceneCollection *)
+getSceneCollections(): vector<RemoteSceneCollection*>
+setName(sceneListName: string, cb: SetNameCallback)
+getName(): string
1
0..*
RemoteSceneCollection
-m_id: string
-m_name: string
-m_mapRemoteScenes: map<string;RemoteScene*>
-m_pRequestor: SceneCollectionResourceRequestor *
+addNewScene(sceneName: string, cb: AddNewSceneCallback): void
+removeScene(name: string, cb: RemoveSceneCallback): void
+getScenes(): map<string;RemoteScene*>
+getScene(sceneName: string): RemoteScene*
+setName(sceneCollectionName: string, cb: SetNameCallback): void
+getName(): string
+getId(): string
1
0..*
RemoteScene
-m_name: string
-m_mapRemoteSceneActions: map<string; vector<RemoteSceneAction*>>
-m_pRequestor: SceneCollectionResourceRequestor *
+addNewSceneAction(RCSRemoteResourceObject*, RCSResourceAttributes*, cb: AddNewSceneActionCallback)
+addNewSceneAction(RCSRemoteResourceObject*, string, Value, cb: AddNewSceneActionCallback)
+removeSceneAction(RemoteSceneAction*, cb: RemoveSceneActionCallback)
+removeSceneAction(RCSRemoteResourceObject*, cb: RemoveSceneActionCallback)
+getRemoteSceneAction(RCSRemoteResourceObject*): vector<RemoteSceneAction*>
+getSceneActions(): vector<RemoteSceneAction*>
+getName(): string
+execute(cb: RemoteSceneExecuteCallback)
0..*
1
RemoteSceneAction
-m_sceneName: string
-m_attributes: RCSResourceAttributes
-m_requestorPtr: SceneMemberResourceRequestor *
+update(attr: RCSResourceAttributes &, cb: UpdateActionCallback)
+update(key: string&, value: Value &, UpdateActionCallback)
+getAction(): RCSResourceAttributes
+getRemoteResourceObject(): RCSRemoteResourceObject*

3.4 {Module 'n'} Component Design

3.5 Module Description

Component	Module	Description
Fan	fanserver.cpp	This module is resource server of iotivity representing a Fan whose state can be change locally or remotely by controller.
Light	lightserver.cpp	This module is also resource server representation of light controlled by controller.
Server application	sceneserver.cpp	This is simple server which has a role to 1. Discover resources 2. Create a scene 3. add new scene to scene collection 4. add actions to scene 5. execute the scene
Client application	sceneclient.cpp	It also have feature similar to local controller. the difference is it can be done remotely using simple client.

3.6 Interfaces

Component providing interface	Related Module	Interface ID	Description
core.light, core.fan	server,fan light, client	oic.if.baseline	This is default interface by OIC. Includes all information about the resource.
core.sceneserver, core.sceneclient	server,fan, client, light	oic.if.ll	includes only the collection information. Default type.

3.7 Sequence Diagram

The execution of different modules is explained using sequence diagram. There are three interaction described below:

1. Check support for a particular scene
2. Create Scene
3. Scene interaction

Fig: Check Scene Support

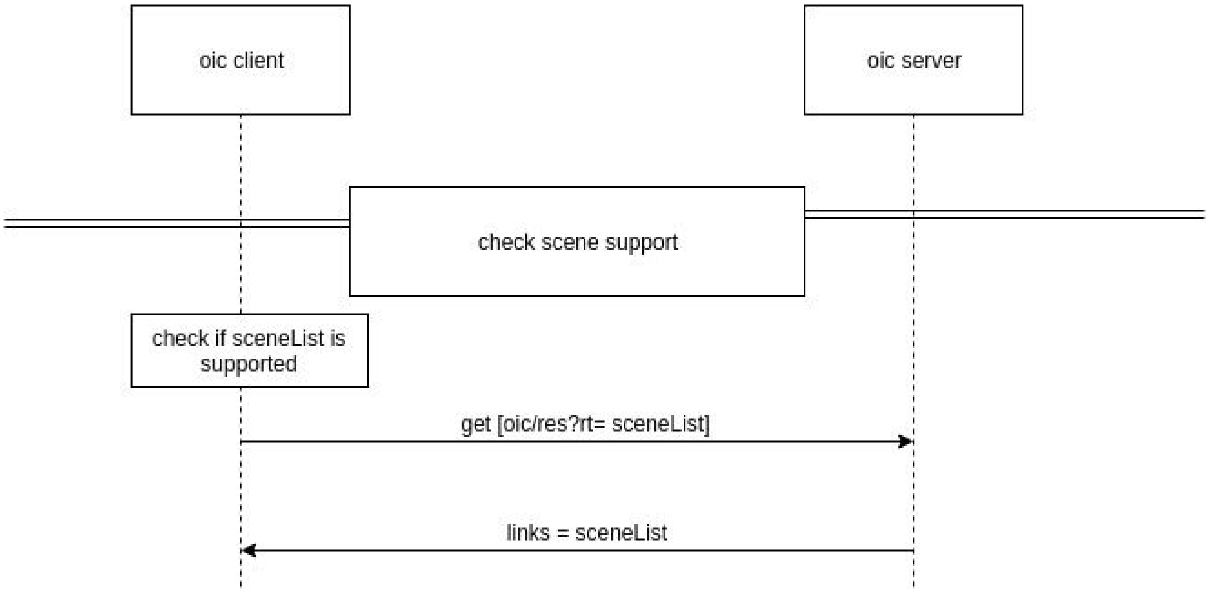


Fig: Create Scene

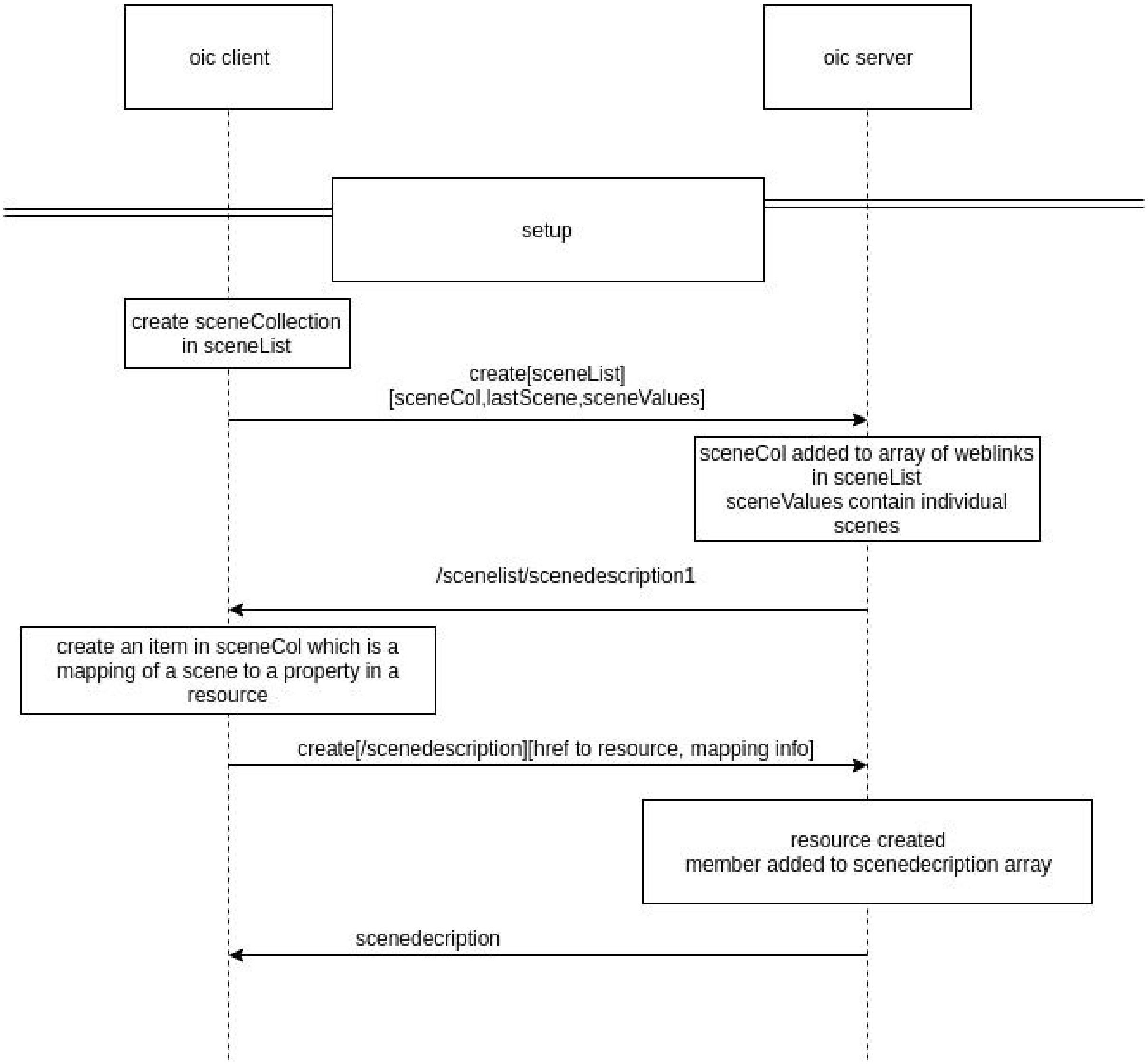
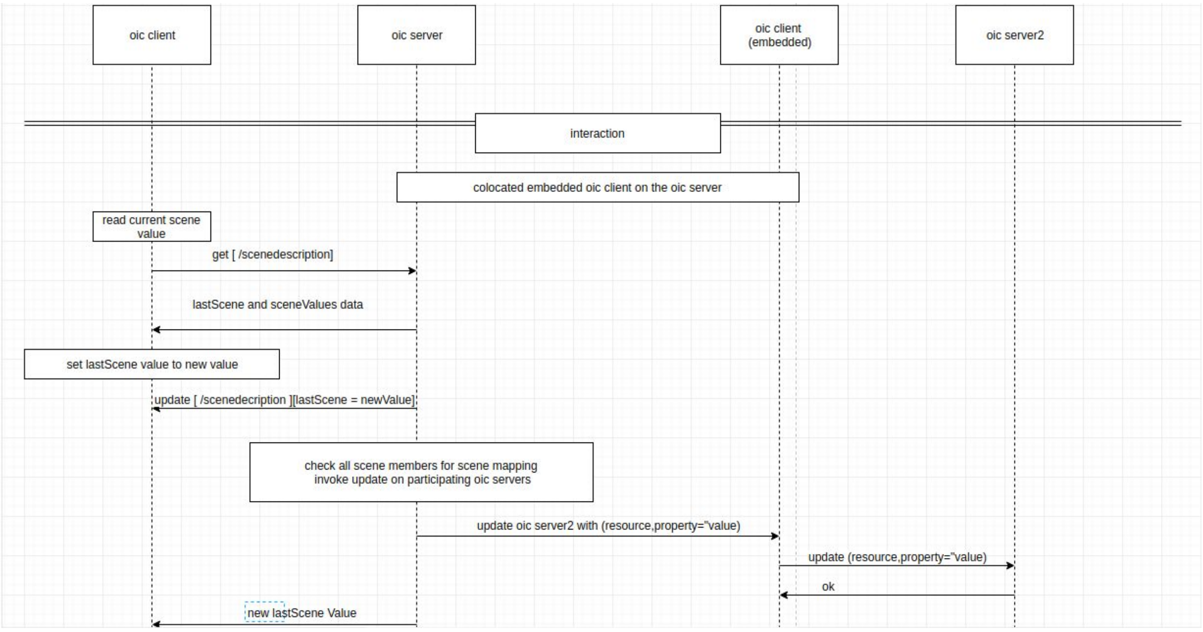

oic client
oic server
setup
create sceneCollection in sceneList
create[sceneList]
[sceneCol,lastScene,sceneValues]
sceneCol added to array of weblinks in sceneList
sceneValues contain individual scenes
/scenelist/scenedescription1
create an item in sceneCol which is a mapping of a scene to a property in a resource
create[/scenedescription][href to resource, mapping info]
resource created
member added to scenedecription array
scenedecription

Fig: Scene Interaction


oic client
oic server
oic client (embedded)
oic server2
interaction
colocated embedded oic client on the oic server
read current scene value
get [/scenedescription]
lastScene and sceneValues data
set lastScene value to new value
update [/scenedecription][lastScene = newValue]
check all scene members for scene mapping
invoke update on participating oic servers
update oic server2 with (resource,property="value)
update (resource,property="value)
ok
new lastScene Value

3.8 SW Code Structure

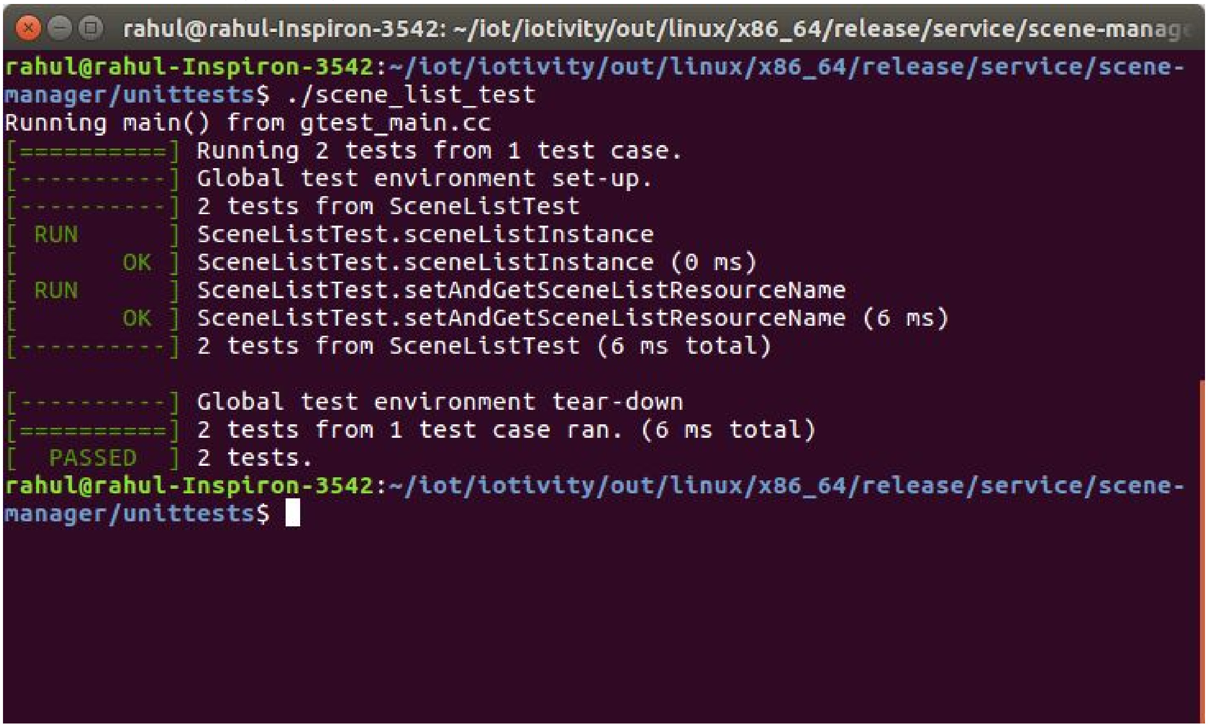
Mapping list of modules and files (or folders)

Module name	File name (or folder name)
sceneserver	sceneserver.cpp
	IoT_submit/cpp_files
	executable file name: sceneserver
	folder: IoT_submit/output_files
fanserver	fanserver.cpp
	IoT_submit/cpp_files
	executable file name: fanserver
	folder: IoT_submit/output_files
lightserver	lightserver.cpp
	IoT_submit/cpp_files
	executable file name: lightserver
	folder: IoT_submit/output_files
sceneclient	sceneclient.cpp
	IoT_submit/cpp_files
	executable file name: sceneclient
	folder: IoT_submit/output_files

3.9 SW Unit Test Report

The following test are performed on different units and results are attached as figures.

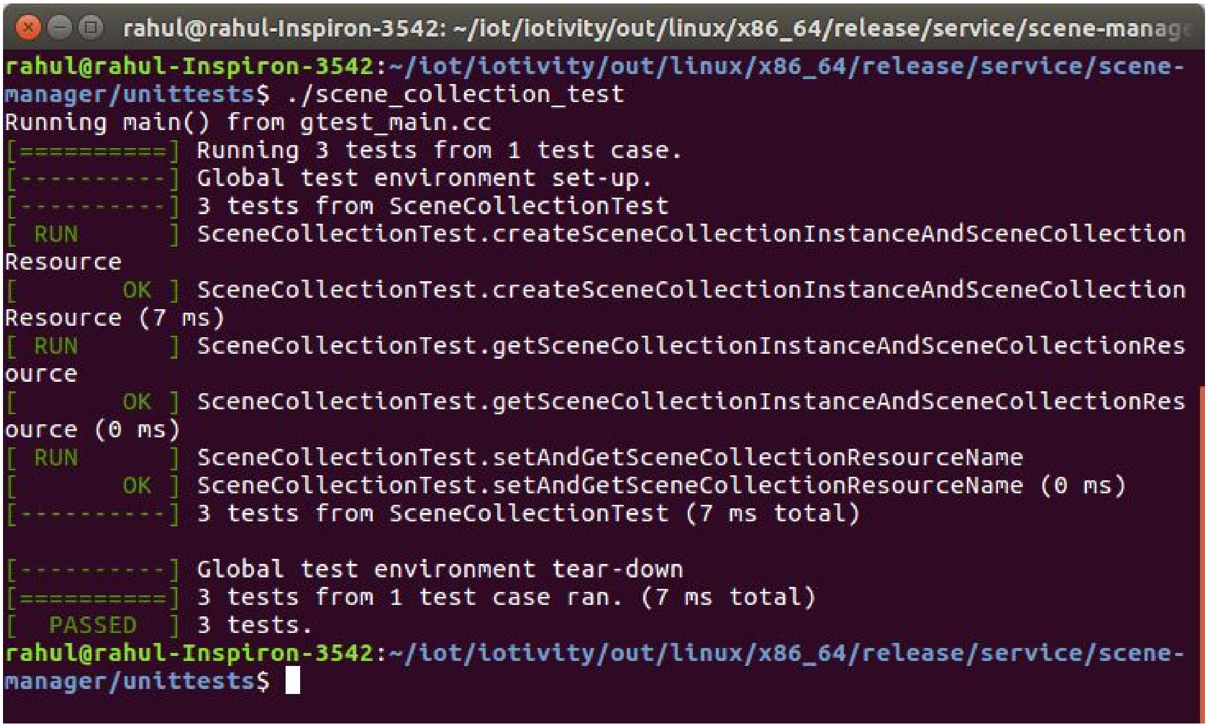
1. **scene_list_test (Passed)**


rahul@rahul-Inspiron-3542: ~/iot/iotivity/out/linux/x86_64/release/service/scene-manag

```
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ ./scene_list_test
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from SceneListTest
[ RUN      ] SceneListTest.sceneListInstance
[       OK ] SceneListTest.sceneListInstance (0 ms)
[ RUN      ] SceneListTest.setAndGetSceneListResourceName
[       OK ] SceneListTest.setAndGetSceneListResourceName (6 ms)
[----------] 2 tests from SceneListTest (6 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (6 ms total)
[  PASSED  ] 2 tests.
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ 
```

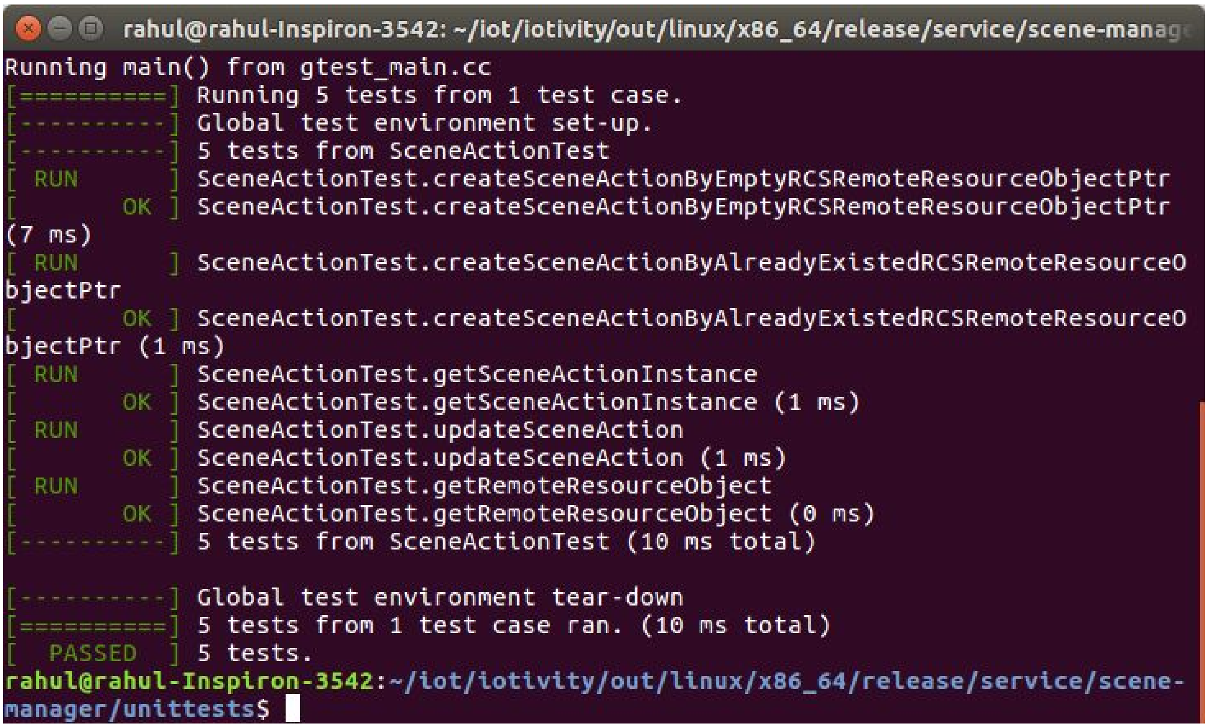
2. **scene_collection_test (Passed)**


rahul@rahul-Inspiron-3542: ~/iot/iotivity/out/linux/x86_64/release/service/scene-manag

```
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ ./scene_collection_test
Running main() from gtest_main.cc
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from SceneCollectionTest
[ RUN      ] SceneCollectionTest.createSceneCollectionInstanceAndSceneCollection
Resource
[       OK ] SceneCollectionTest.createSceneCollectionInstanceAndSceneCollection
Resource (7 ms)
[ RUN      ] SceneCollectionTest.getSceneCollectionInstanceAndSceneCollectionRes
ource
[       OK ] SceneCollectionTest.getSceneCollectionInstanceAndSceneCollectionRes
ource (0 ms)
[ RUN      ] SceneCollectionTest.setAndGetSceneCollectionResourceName
[       OK ] SceneCollectionTest.setAndGetSceneCollectionResourceName (0 ms)
[----------] 3 tests from SceneCollectionTest (7 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (7 ms total)
[  PASSED  ] 3 tests.
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ 
```

3. **scene_action_test (Passed)**

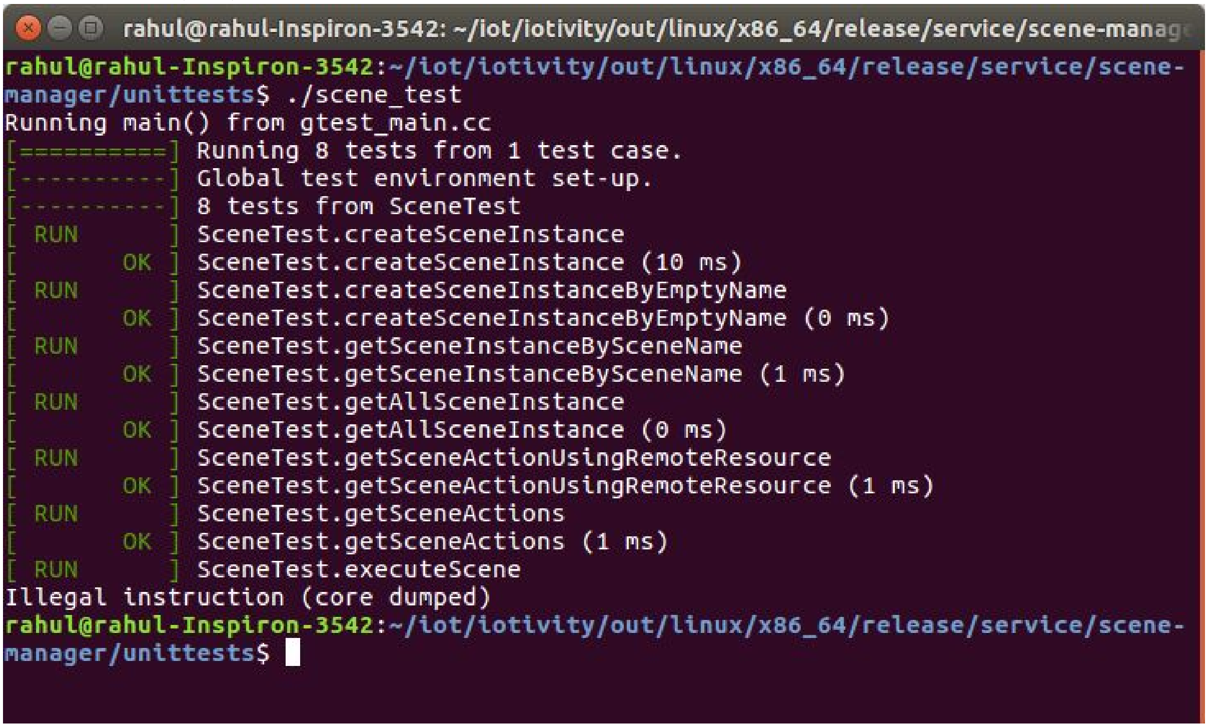


rahul@rahul-Inspiron-3542: ~/iot/iotivity/out/linux/x86_64/release/service/scene-manag

```
Running main() from gtest_main.cc
[==========] Running 5 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 5 tests from SceneActionTest
[ RUN      ] SceneActionTest.createSceneActionByEmptyRCSRemoteResourceObjectPtr
[       OK ] SceneActionTest.createSceneActionByEmptyRCSRemoteResourceObjectPtr
(7 ms)
[ RUN      ] SceneActionTest.createSceneActionByAlreadyExistedRCSRemoteResourceO
bjectPtr
[       OK ] SceneActionTest.createSceneActionByAlreadyExistedRCSRemoteResourceO
bjectPtr (1 ms)
[ RUN      ] SceneActionTest.getSceneActionInstance
[       OK ] SceneActionTest.getSceneActionInstance (1 ms)
[ RUN      ] SceneActionTest.updateSceneAction
[       OK ] SceneActionTest.updateSceneAction (1 ms)
[ RUN      ] SceneActionTest.getRemoteResourceObject
[       OK ] SceneActionTest.getRemoteResourceObject (0 ms)
[----------] 5 tests from SceneActionTest (10 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test case ran. (10 ms total)
[  PASSED  ] 5 tests.
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ 
```

4. scene_test (bug)



rahul@rahul-Inspiron-3542: ~/iot/iotivity/out/linux/x86_64/release/service/scene-manag

```
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ ./scene_test
Running main() from gtest_main.cc
[==========] Running 8 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 8 tests from SceneTest
[ RUN      ] SceneTest.createSceneInstance
[       OK ] SceneTest.createSceneInstance (10 ms)
[ RUN      ] SceneTest.createSceneInstanceByEmptyName
[       OK ] SceneTest.createSceneInstanceByEmptyName (0 ms)
[ RUN      ] SceneTest.getSceneInstanceBySceneName
[       OK ] SceneTest.getSceneInstanceBySceneName (1 ms)
[ RUN      ] SceneTest.getAllSceneInstance
[       OK ] SceneTest.getAllSceneInstance (0 ms)
[ RUN      ] SceneTest.getSceneActionUsingRemoteResource
[       OK ] SceneTest.getSceneActionUsingRemoteResource (1 ms)
[ RUN      ] SceneTest.getSceneActions
[       OK ] SceneTest.getSceneActions (1 ms)
[ RUN      ] SceneTest.executeScene
Illegal instruction (core dumped)
rahul@rahul-Inspiron-3542:~/iot/iotivity/out/linux/x86_64/release/service/scene-
manager/unittests$ 
```

3.10 Bugs known at submission date

DATE: 2018.18.04

S.No	Bug List	Description
1.	scene_test	core dumped when executing a NULL scene

4 SW Development Completion Report

4.1 Project Result Analysis

4.2 Development Work Promotion Results

This project helped us learn various technologies such as
(i) iotivity stack
(ii) server-client interaction using iotivity stack in a linux environment using loopback server
(iii) CoAP over UDP
(iv) using raspberry pi as a resource server for iotivity stack

Item	Result
Raspberry Pi	iotivity node
smart connected device	interaction using scene manager
discovery manager	how to discover and register resource in iotivity

4.3 Development Results and Utilization

This development can be used in smart home with smart connected devices, smart appliances, smart wearables and sensors whose state can be changed based on some scene or when some situation occurs.
This iotivity based project can be utilized in any smart home having client-server type functionality.

4.4 Deliverables List

S.No	Executable Name	Description
1.	sceneserver	This executable is executed with at least one resource server (fan/light) running. This is used to create scene, add a scene to scene collection and execute a scene from added scene.
2.	sceneclient	This is used to remotely create a scene, add or update already existing scene and execute a scene remotely.
3.	fanserver	This is resource server representation of IoT device for Fan.
4.	lightserver	This is the resource server representation of IoT device for Light.

References

[1] https://wiki.iotivity.org/
[2] https://jira.iotivity.org/browse/IOT-934
[3] https://wiki.tizen.org/images/1/1a/09-IoTivity_Scene_Manager.pdf
[4] https://openconnectivity.org/wp-content/uploads/2016/01/Ashok-Subash.pdf
[5] https://wiki.iotivity.org/scene_manager