TALLINN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF SOFTWARE SCIENCE

Assignment in subject "Software Assurance" (ITI8610)

Subsystem to be analyzed: smart home climate control (scenario 8 - smart home system)

Course Work on Contracts

Authors, group ID
Aleksei Gvozdev
Aleksei Netšunajev
Anna Krajuškina
Alex Neil
Natalya Berezovski
Viktoria Tisler

Group ID IAPM
Date: 07.01.2018

TALLINN 2017

1. Architectural Entities

For the project the smart home system is selected. The system may get very sophisticated and for that reason we focus on a very small part of the smart home. We describe light, humidity and temperature subsystems in the project shortly .

The following architectural entities:

1. Controllers: receive signals from sensors and environment and adjust behaviour of the devices
    a. humidity controller
    b. light controller
    c. temperature controller
2. Sensor: elements for retrieving and interpreting environmental data
    a. humidity sensor
    b. light sensor
    c. temperature sensor
3. Devices: apply changes to environment
    a. air conditioner
    b. dehumidifier
    c. heater
    d. humidifier
    e. lamp
4. Environment that surrounds the smart home
5. User

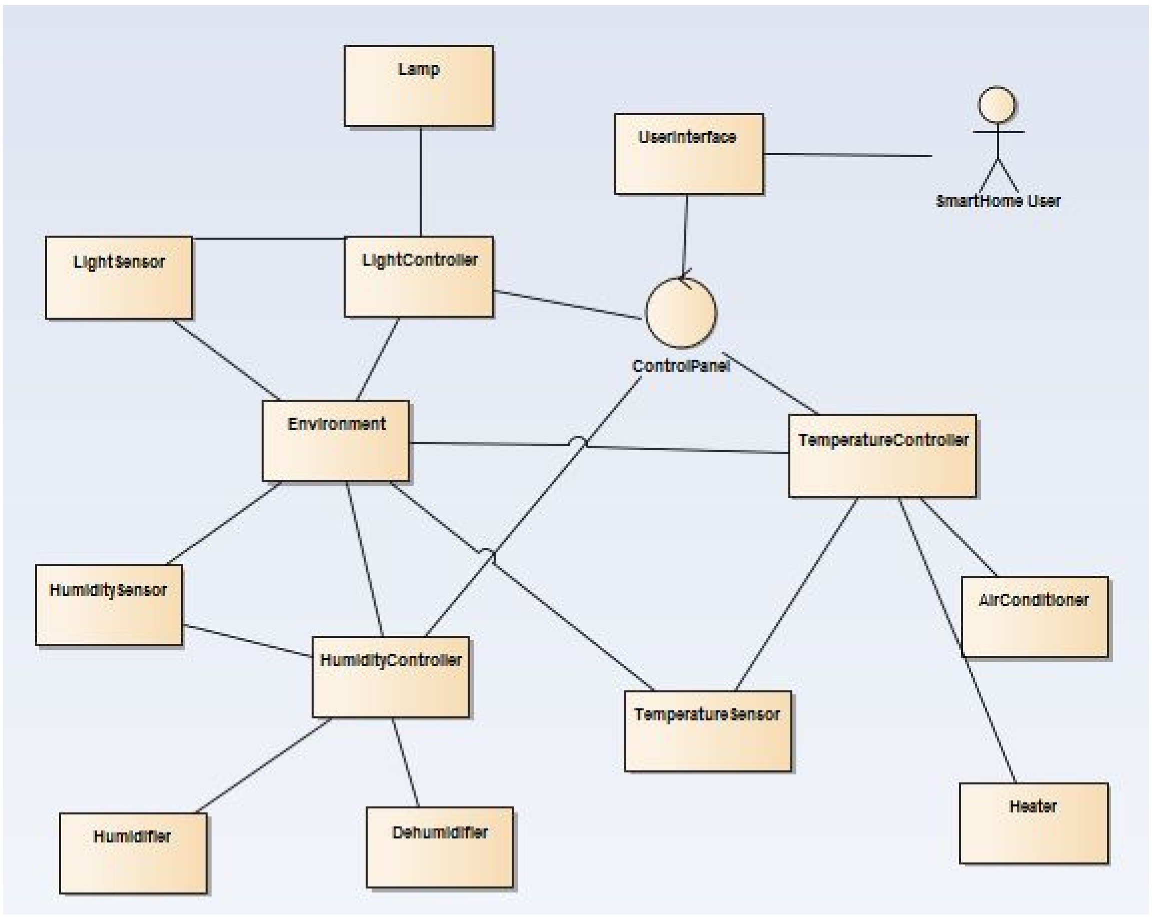


Diagram 1. Architectural diagram of the whole system with all entities present

2. Communication Diagrams

Below are provided communication diagrams that describe 3 processes:

- humidity control;
- temperature control;
- light control.

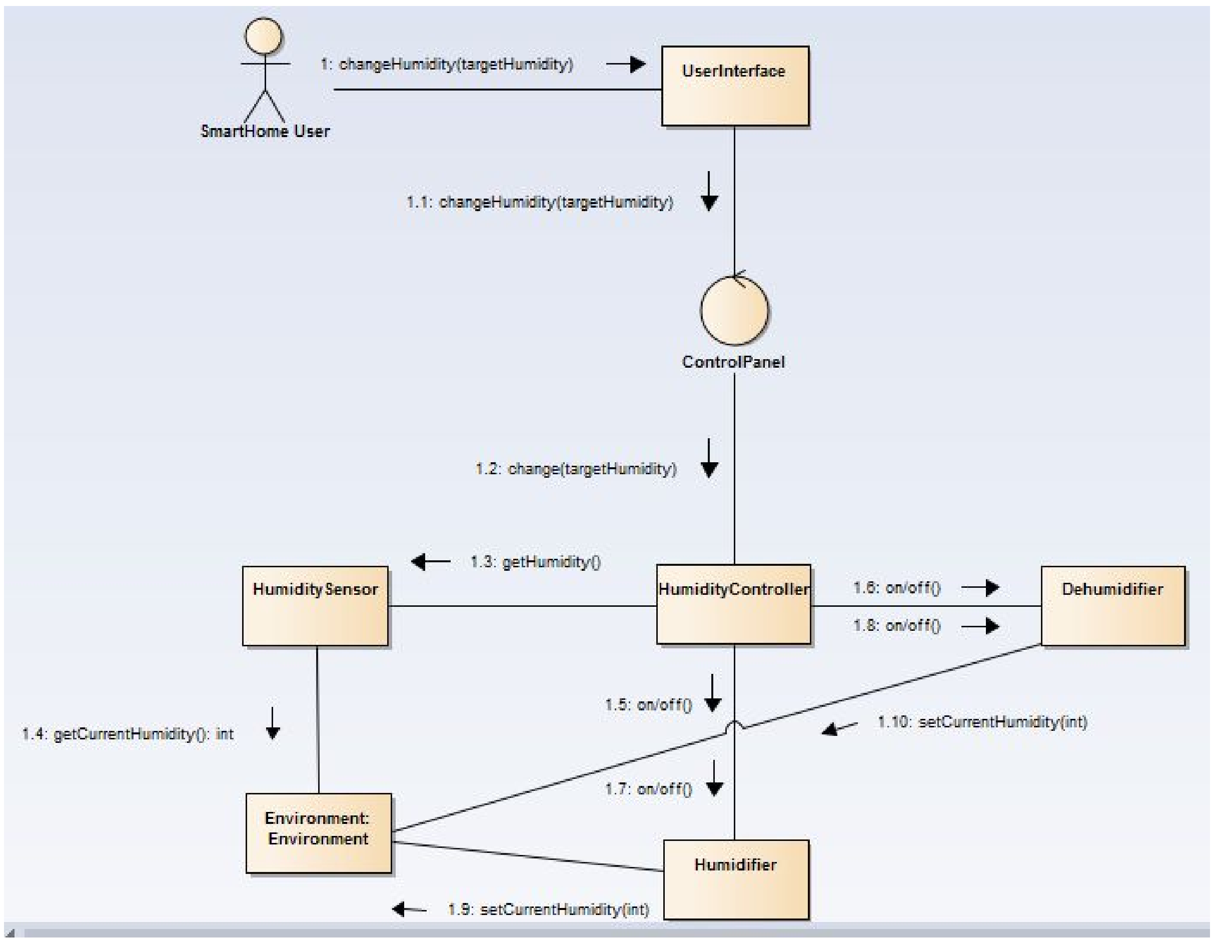


Diagram 2. Communication diagram of the humidity controlling part

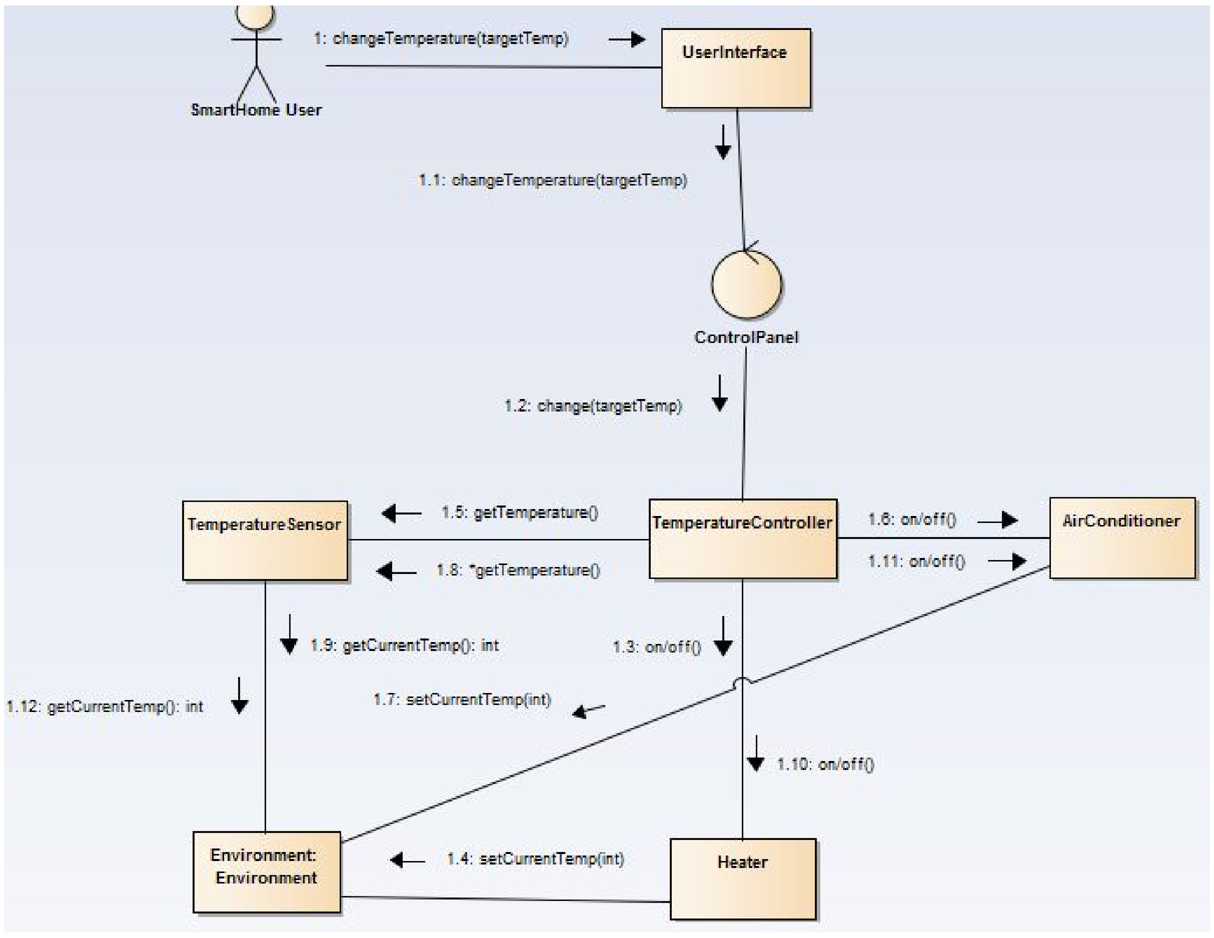
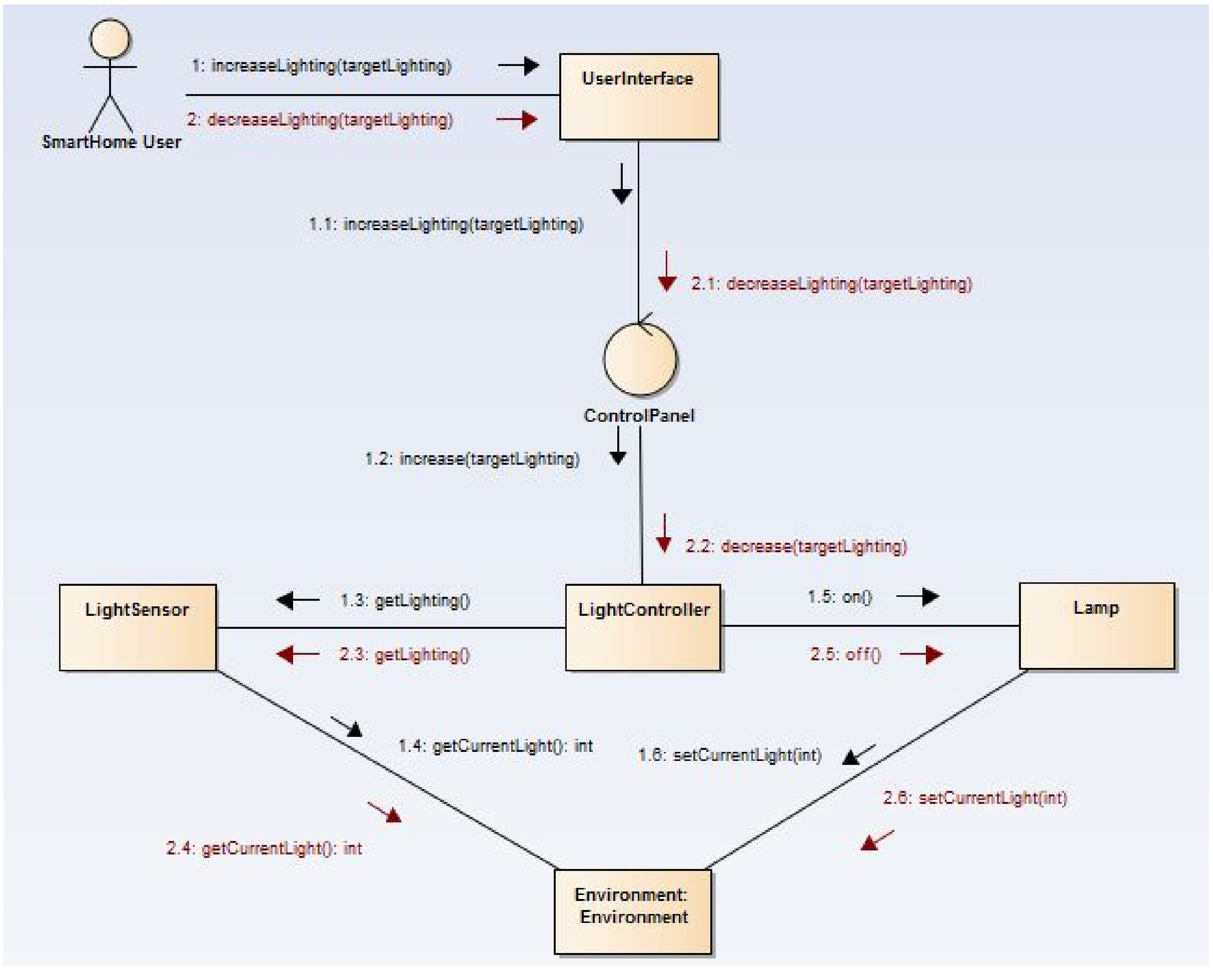


Diagram 3. Communication diagram of the temperature controlling part



SmartHome User
1: increaseLighting(targetLighting)
2: decreaseLighting(targetLighting)
UserInterface
1.1: increaseLighting(targetLighting)
2.1: decreaseLighting(targetLighting)
ControlPanel
1.2: increase(targetLighting)
2.2: decrease(targetLighting)
LightSensor
1.3: getLighting()
2.3: getLighting()
LightController
1.5: on()
2.5: off()
Lamp
1.4: getCurrentLight(): int
2.4: getCurrentLight(): int
1.6: setCurrentLight(int)
2.6: setCurrentLight(int)
Environment: Environment

Diagram 4. Communication diagram of the light controlling part

3. System quality attributes

We consider following quality attributes that were defined for the system:

a) Security - prevents environmental parameters from taking extreme values. For instance an option of increasing temperature over a certain threshold should be eliminated.
b) Functional correctness - ensures that code works as expected.

Below is provided an example of a JML contracts for security quality attribute.

```
//@ assignable lowerValue;
//@ requires goalValue >= 20 && goalValue <= 80;
//@ ensures lowerValue == goalValue - 1;
void setLowerValue() {
   lowerValue = goalValue - 1;
}
```

Functional correctness assertions:

```
  //@ assignable currentLight;
  //@ requires currentLight - increment >= MIN_POWER;
  //@ ensures (currentLight - increment >= MIN_POWER) && currentLight ==
\old(currentLight) - increment;
  public void decreaseLight() {
     if (currentLight - increment >= MIN_POWER){
        currentLight = currentLight - increment;
        Environment.getInstance().setCurrentLight(currentLight);
     }
  }
```

4. JML Contract Assertions and their Consistency

We chose to implement the JML contracts using OpenJML and OpenJML plugin for Eclipse. SMT solver used to verify the contract assertions is the Key Tool plugin for Eclipse IDE. The reason why this tool was chosen lies in the technical issues that our team members were facing while setting up the infrastructure for the project. This combination worked out for all team members while people were having troubles setting up Z3 or some other solvers on various operating systems. Solvers were performing differently during setup on various operating systems and that caused troubles in verifying assertions. Some proofs were also checked using Z3 solver on a Windows machine.

Below are some examples from the code, where JML was used. It is important to point out that not all Java features may be used by Key Software for verification, thus our team has decided to simplify code in order to avoid verification failure. OpenJML and Key solver does not support multi-threading, library methods, Java 8 and many other features. These features of Java were abandoned for the sake of the ability of verifying contracts formally.

LightController.java:

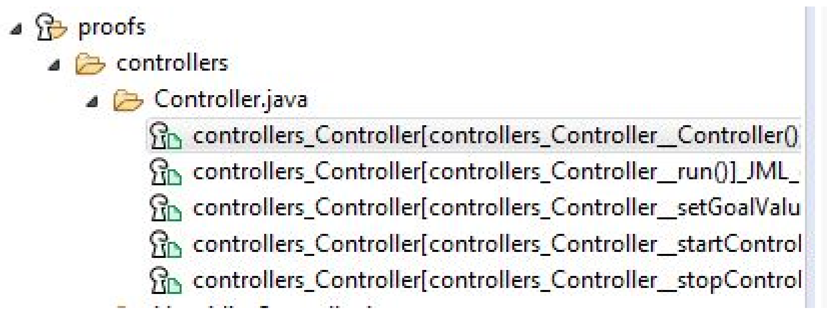
```
  //@ requires lightSensor != null;
  //@ ensures \result >= 20 && \result <= 100;
  int getEnvironmentalData() {
     return lightSensor.getEnvironmentalData();
  }
```

HumiditySensor.java:

```
  //@ assignable currentValue;
  //@ ensures \result >= 20 & \result <= 80;
  public int getEnvironmentalData() {
     currentValue = Environment.getInstance().getCurrentHumidity();
     return currentValue;
  }
```
Controller.java:
```
protected /*@ spec_public @*/ boolean runThread;
```

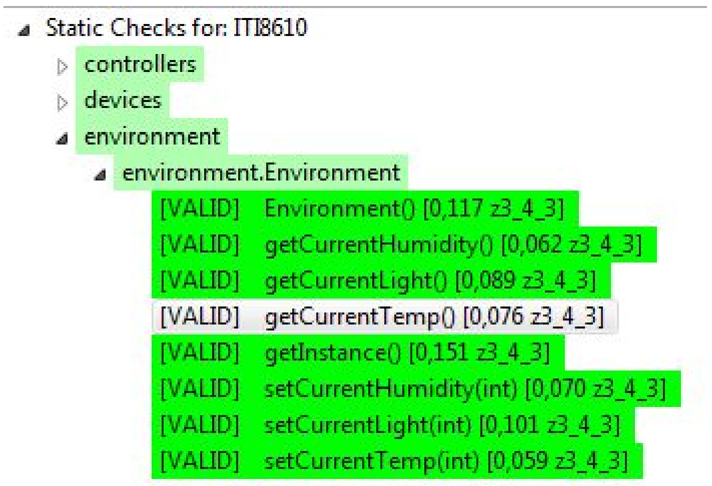
HumidityController.java:
```
  //@ diverges true;
  //@ assignable \nothing;
  void stopDevices() {
      stopHumidifier();
      stopDehumidifier();
  }
```

To complete proofs the project was converted to Key project in Eclipse. This results in automatic check of JML assertions by Key tool. The project structure is amended by *proofs* section. Figure 1 shows an example from the project structure obtained with Key tool. Figure 2 shows an example of a proof completion using Z3 solver.



```
proofs
  controllers
    Controller.java
      controllers_Controller[controllers_Controller__Controller()
      controllers_Controller[controllers_Controller__run()]_JML_
      controllers_Controller[controllers_Controller__setGoalValu
      controllers_Controller[controllers_Controller__startControl
      controllers_Controller[controllers_Controller__stopControl
```

Figure 1. Example of a proof completion from the project structure.



```
Static Checks for: ITI8610
  controllers
  devices
  environment
    environment.Environment
      [VALID]   Environment() [0,117 z3_4_3]
      [VALID]   getCurrentHumidity() [0,062 z3_4_3]
      [VALID]   getCurrentLight() [0,089 z3_4_3]
      [VALID]   getCurrentTemp() [0,076 z3_4_3]
      [VALID]   getInstance() [0,151 z3_4_3]
      [VALID]   setCurrentHumidity(int) [0,070 z3_4_3]
      [VALID]   setCurrentLight(int) [0,101 z3_4_3]
      [VALID]   setCurrentTemp(int) [0,059 z3_4_3]
```

Figure 2. Example of a proof completion using Z3 solver.

5. Multi-view contracts

Following contract is an example of a multi-view contract where 2 quality attributes are combined - first is security and the second is functional correctness. Our system is extremely simplified due to JML supported features limitations of Key Tool and for this reason we present only a single example.

```
//@ assignable lowerValue;
//@ requires goalValue >= 20 && goalValue <= 80;
//@ also
//@ ensures lowerValue == goalValue - 1;
void setLowerValue() {
   lowerValue = goalValue - 1;
}
```

References

Key Tool: https://www.key-project.org/
OpenJML: http://openjml.org/