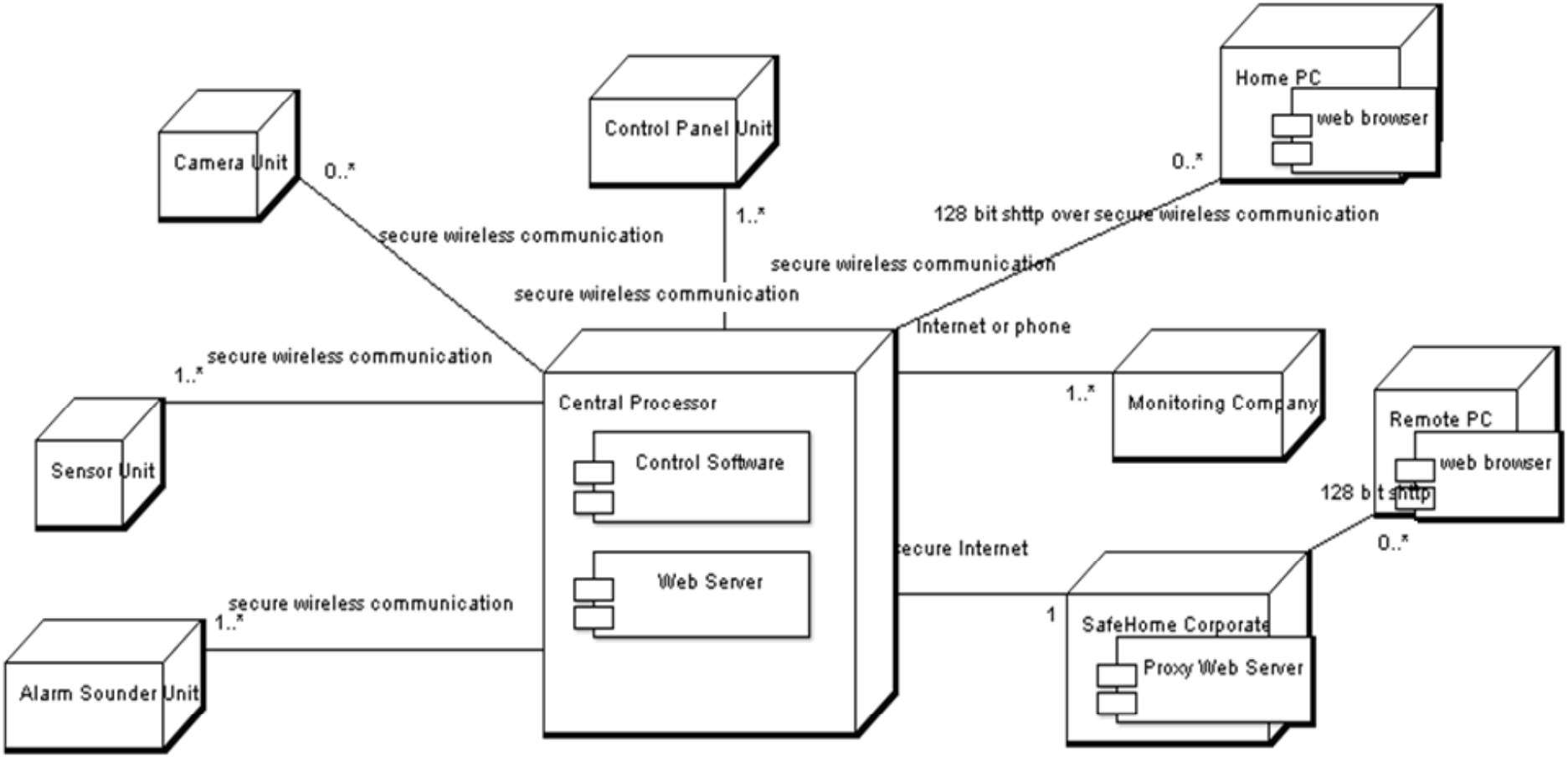
Guidelines for Safehome Implementation

Moonzoo Kim
CS Division of EECS Dept.
KAIST
moonzoo@cs.kaist.ac.kr
http://pswlab.kaist.ac.kr/courses/CS350-07

Overview of the Guidelines

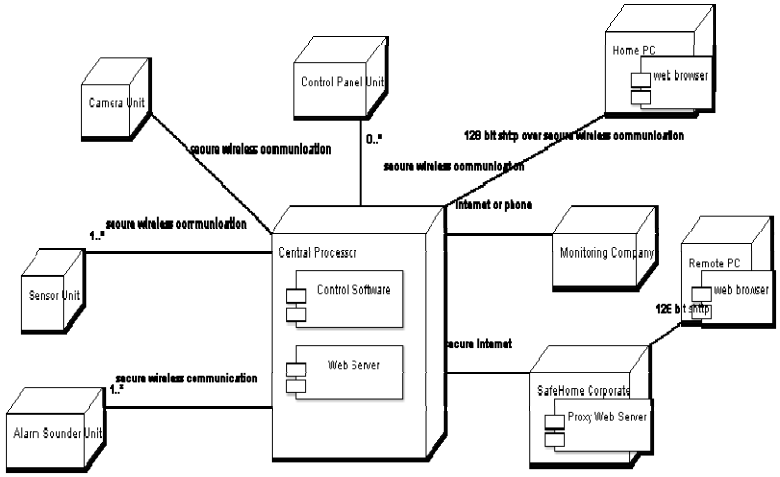
- Deployment diagrams
  - Original one v.s. simplified one
  - Alternative implementation solutions
- Virtual devices
  - Screen shots
  - APIs
- Javadoc
- Junit
- To-do-list

Original Deployment Diagram (1/2)



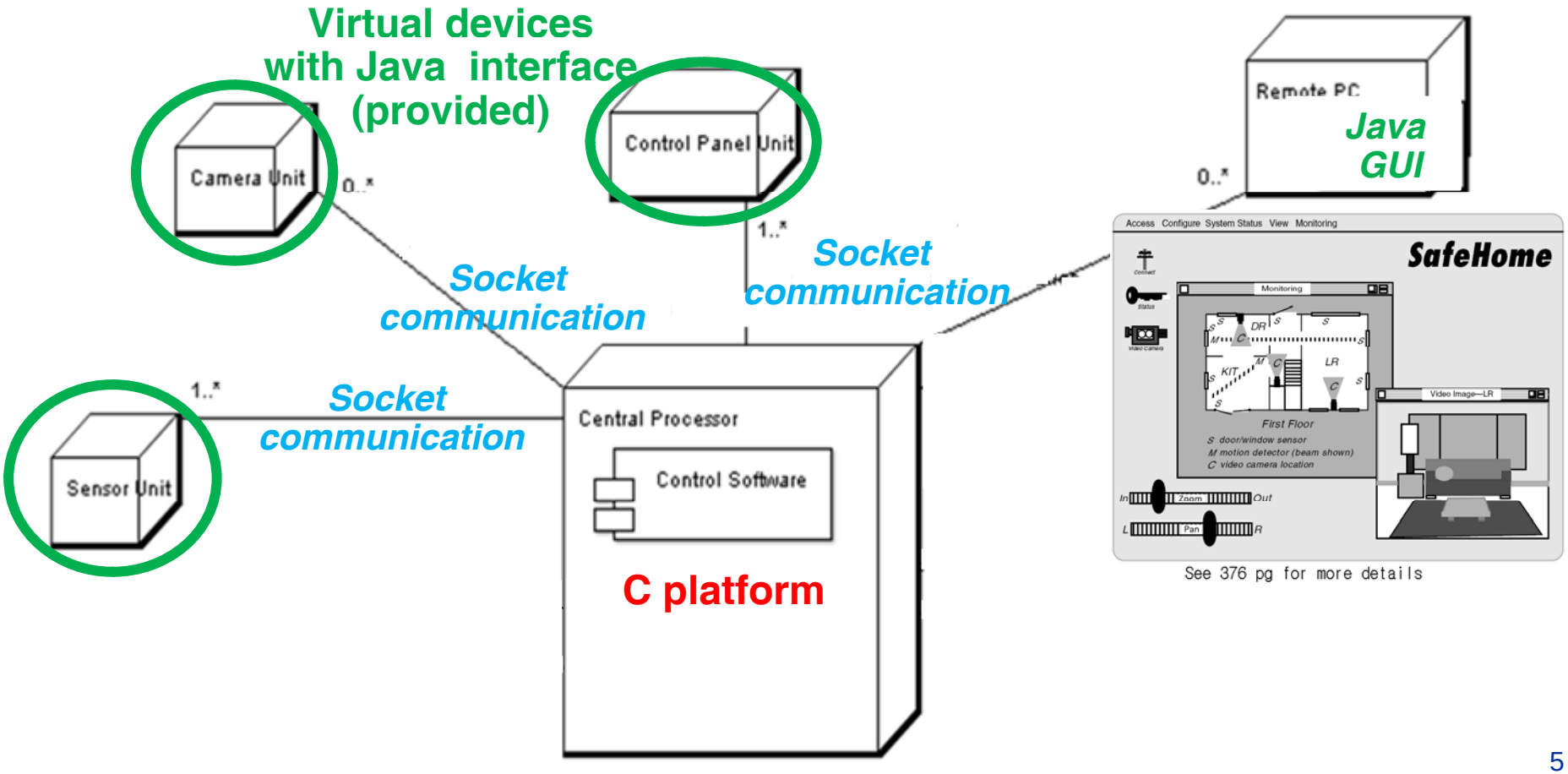
Original Deployment Diagram (2/2)

- Safehome is a safety critical embedded system
  - The central processor (CP) should be hosted by a dedicated reliable HW/SW platform for non-stop operations
  - Java is not suitable for this purpose
    - Java is heavy compared to C/C++
    - JVM is unnecessarily complex platform for Safehome
      - Core functions of Safehome are simple



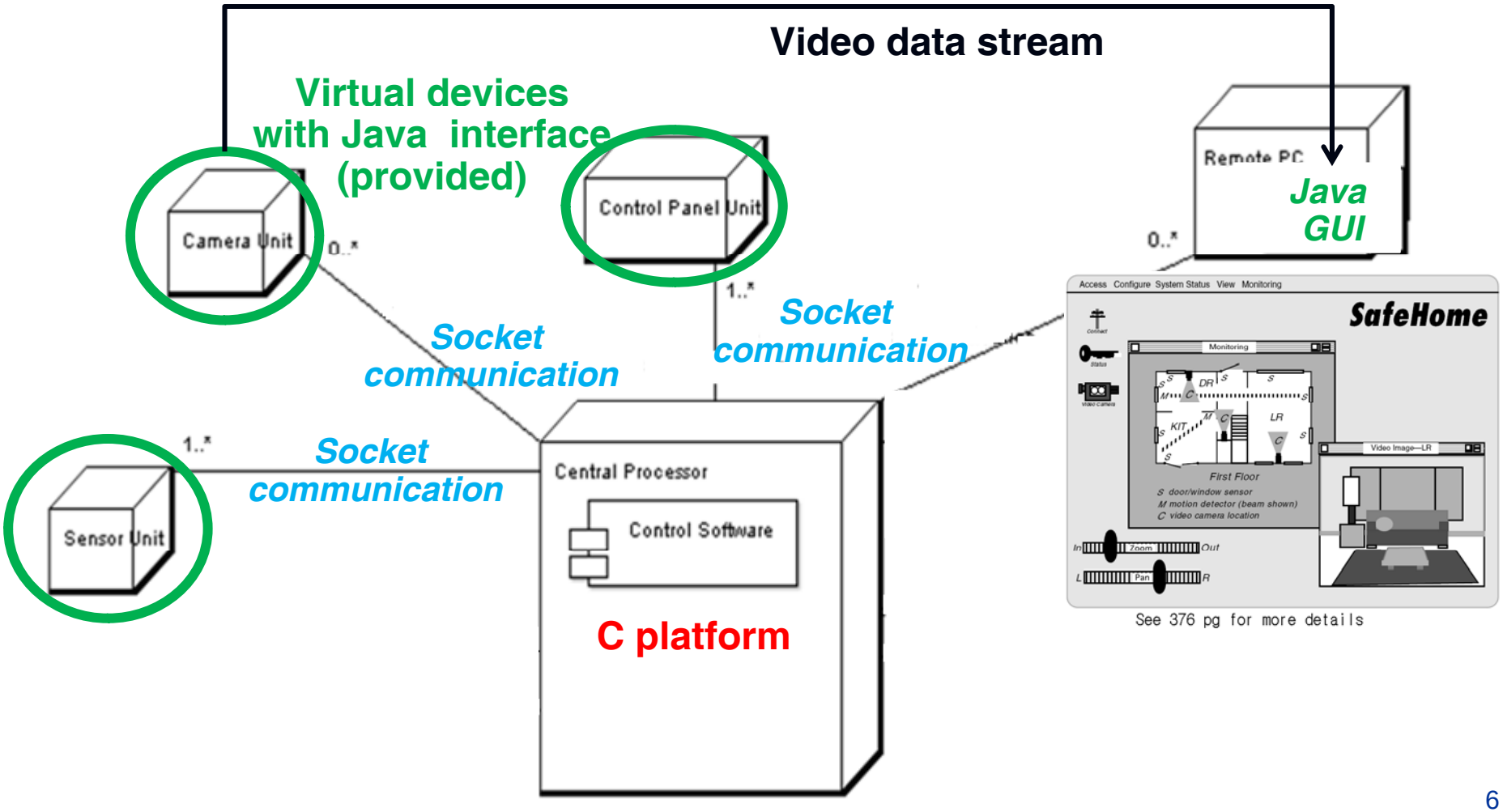
- Mind the heterogeneous platforms
  - Safehome central processor communicates with diverse devices through wireless network connections
  - Flexible communication among the components is crucial

Simplified Deployment Diagram

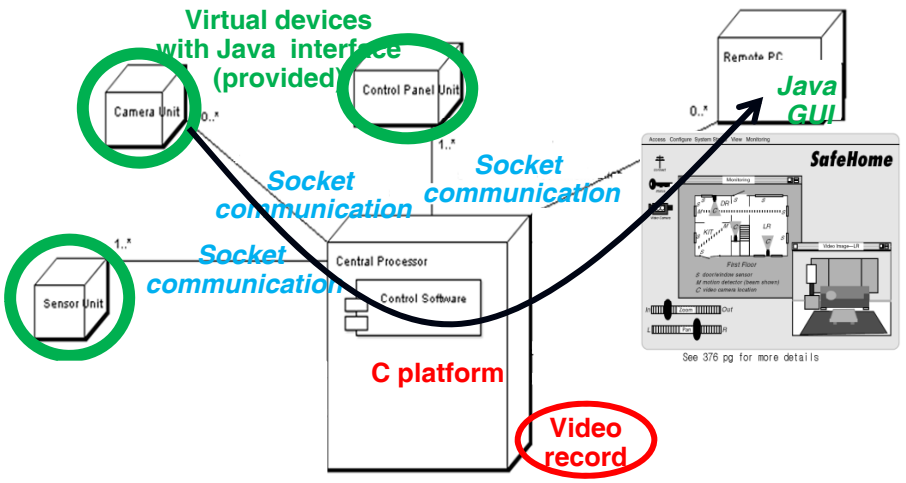


See 376 pg for more details

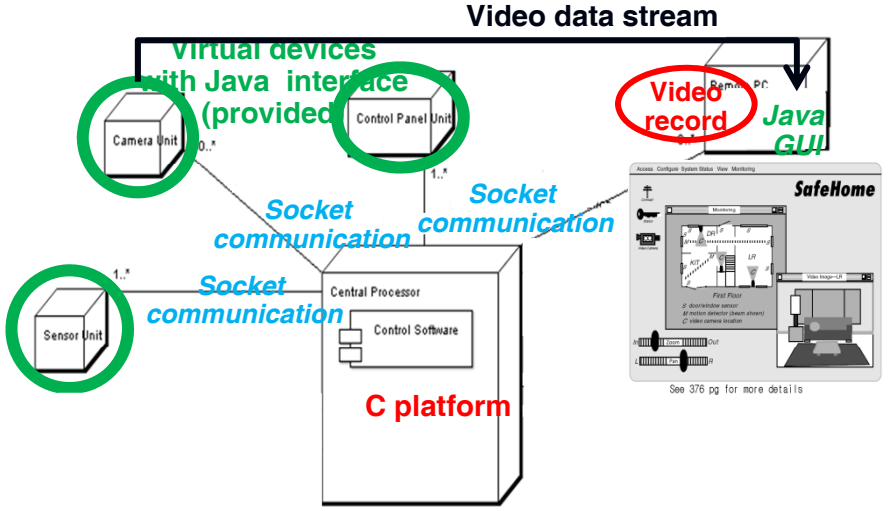
Alternative Implementation Choice



Comparison

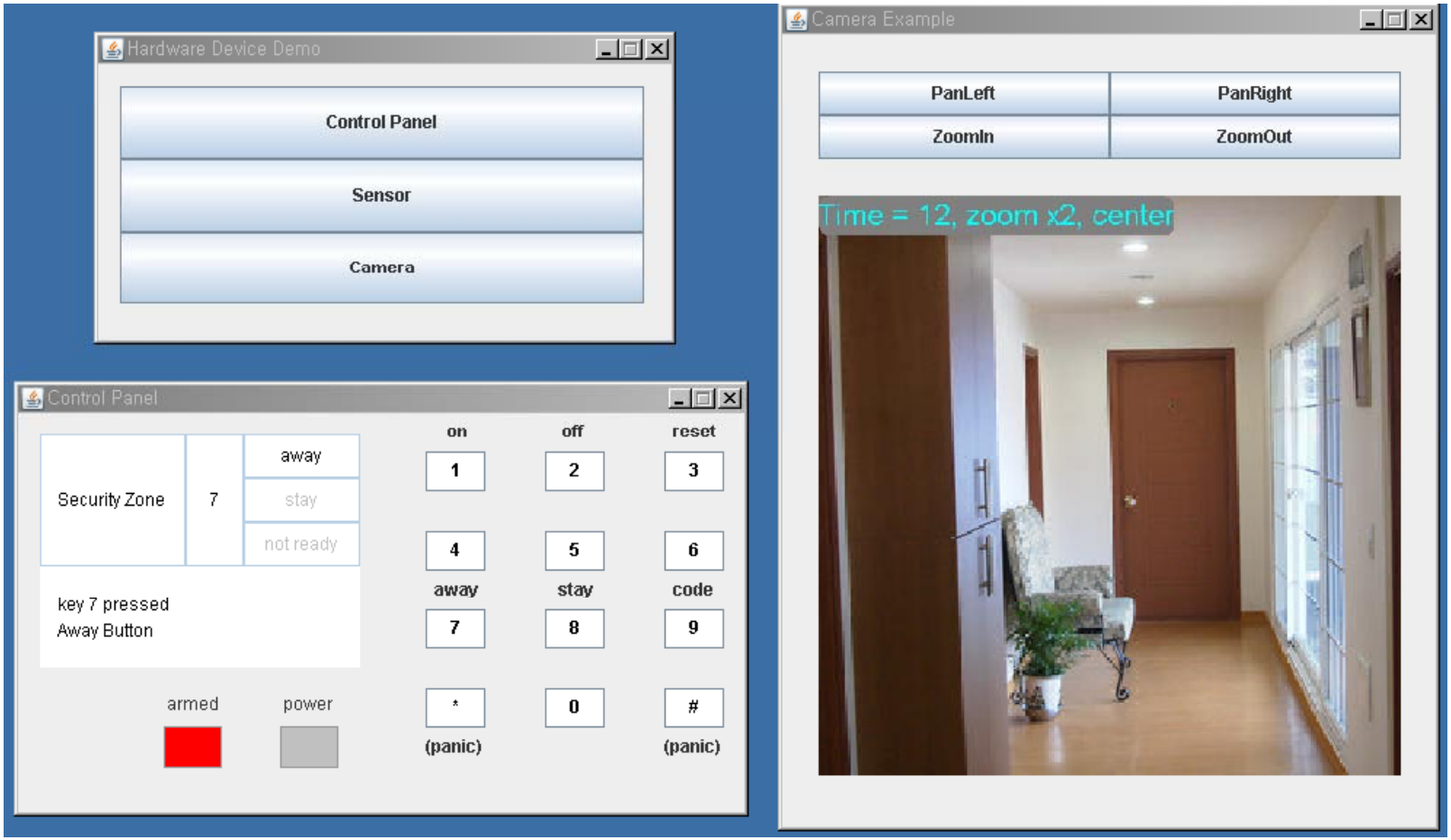


- If video recording is considered crucial, video stream should be handled by CP
  - Pros: reliable video recording
  - Cons: unnecessary computational resource of CP consumed for transferring video stream to the GUI

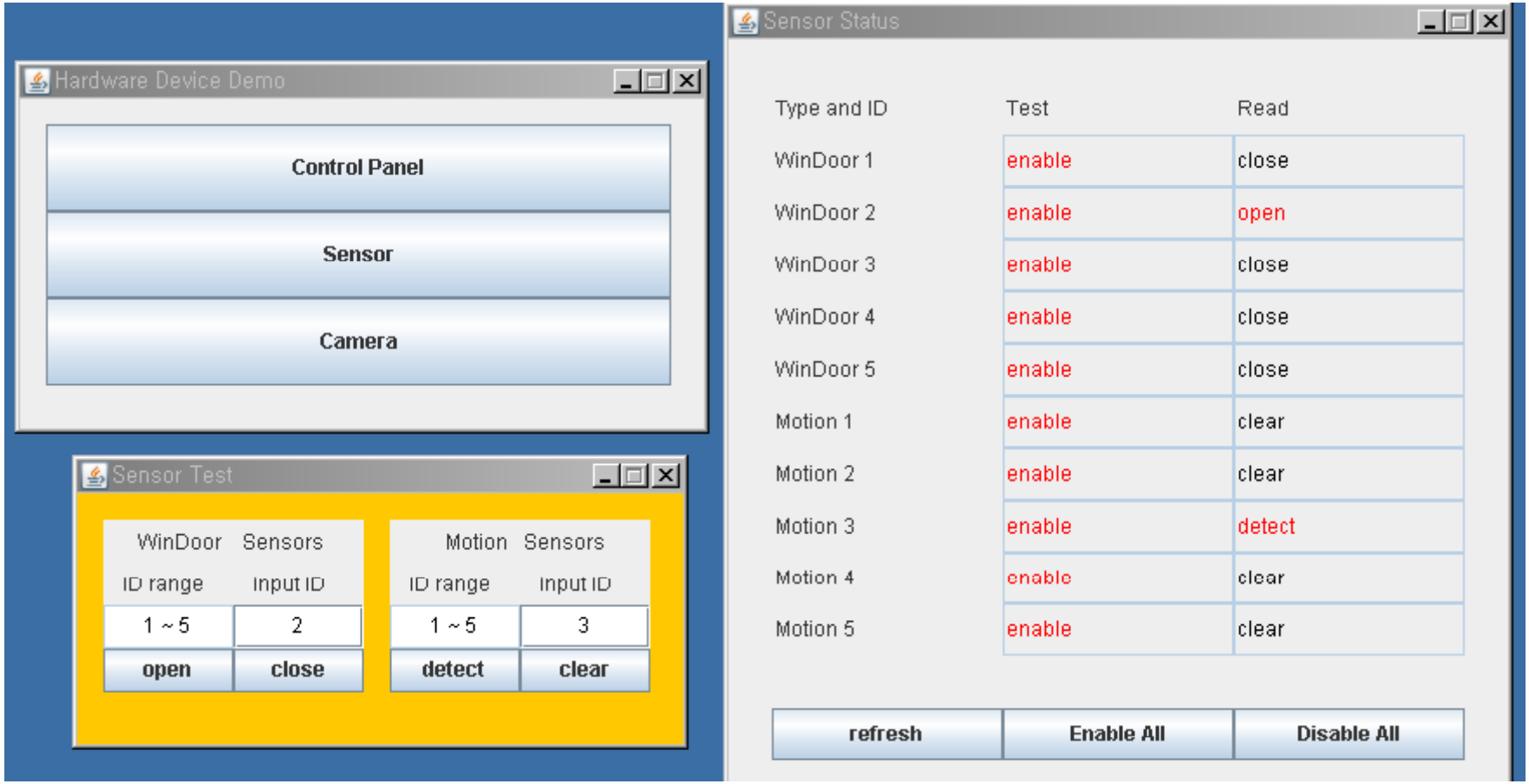


- If video recording is considered optional, video stream can bypass to the home PC without going through the central processor
  - Pros: saved network bandwidth and processing power of the CP
  - Cons: additional network connections and irreliable video recording

Screenshot of Virtual Devices (1/2)



Screenshot of Virtual Devices (2/2)



Virtual Device APIs

- `safehome` package contains `MainDemo.java` which is a main file for the demo application
- `safehome.device` contains virtual devices and its interfaces. You may want to inherit those device classes for your purpose
- You should read carefully the following three java files
  - `safehome.MainDemo.java`
  - `safehome.device.interfaceCamera.java`
  - `safehome.device.interfaceSensor.java`
- `All other java files should not be modified unless youare for your`

Virtual Device APIs for Control Panel

- Callback functions of the control panel's buttons
  - A call back function is invoked when the corresponding button of the control panel GUI is pressed.
  - You can inherit the super class `safehome.device.DeviceControlPanelAbstract`
- When you push a button <n> of the control panel, the corresponding call-back function `abstract public void button<n>()` is invoked.
  - You can inherit that method as shown in `MainDemo.java`
- Output to the control panel
  - `public void setDisplayAway(boolean on)`
  - `public void setDisplayStay(boolean on)`
  - `public void setDisplayNotReady(boolean on)`
  - `public void setDisplayShortMessage(String message)`
  - `public void setLedArmed(boolean on)`

Virtual Device APIs for Camera

- Note that we do not implement video recording function for saving your effort ☺
- There are four functions to control a camera
  - `public boolean panRight();`
  - `public boolean panLeft();`
  - `public boolean zoomIn();`
  - `public boolean zoomOut();`
- Image acquisition is performed by
  - `public BufferedImage getView();`
  - You may serialize the obtained `BufferedImage` object and transfer the serialized image to the central processor. Then the central processor simply deliver the serialized image to the GUI

Virtual Device APIs for Sensors

- `public int getID`
- `public boolean read()`
- `public void enable()`
- `public void disable()`

Javadoc Example

All Classes

Packages
safehome
safehome.device

All Classes
CameraTest
CameraView
ControlPanel
DeviceCamera
DeviceControlPanelA
DeviceMotionDetect
DeviceSensorTester
DeviceWinDoorSens
interfaceCamera
interfaceSensor
MainDemo
SafeHomeSensorTe
SensorTest

Overview Package **Class** Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

safehome

Class ControlPanel

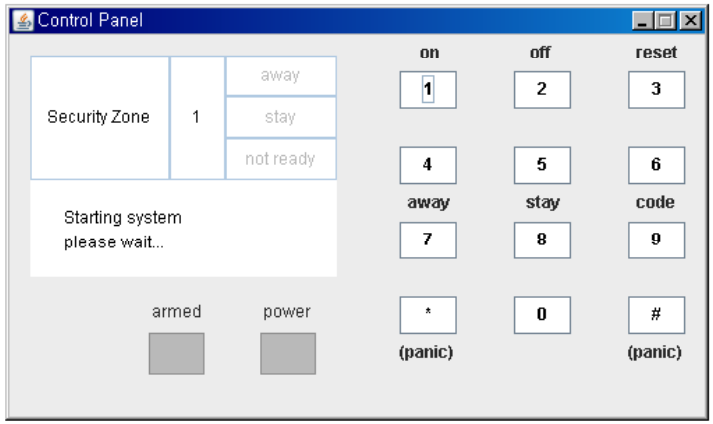
java.lang.Object
└ java.awt.Component
└ java.awt.Container
└ java.awt.Window
└ java.awt.Frame
└ javax.swing.JFrame
└ safehome.device.DeviceControlPanelAbstract
└ **safehome.ControlPanel**

All Implemented Interfaces:
ActionListener, ImageObserver, MenuContainer, Serializable, EventListener, Accessible, RootPaneContainer, WindowConstants

class **ControlPanel**
extends DeviceControlPanelAbstract

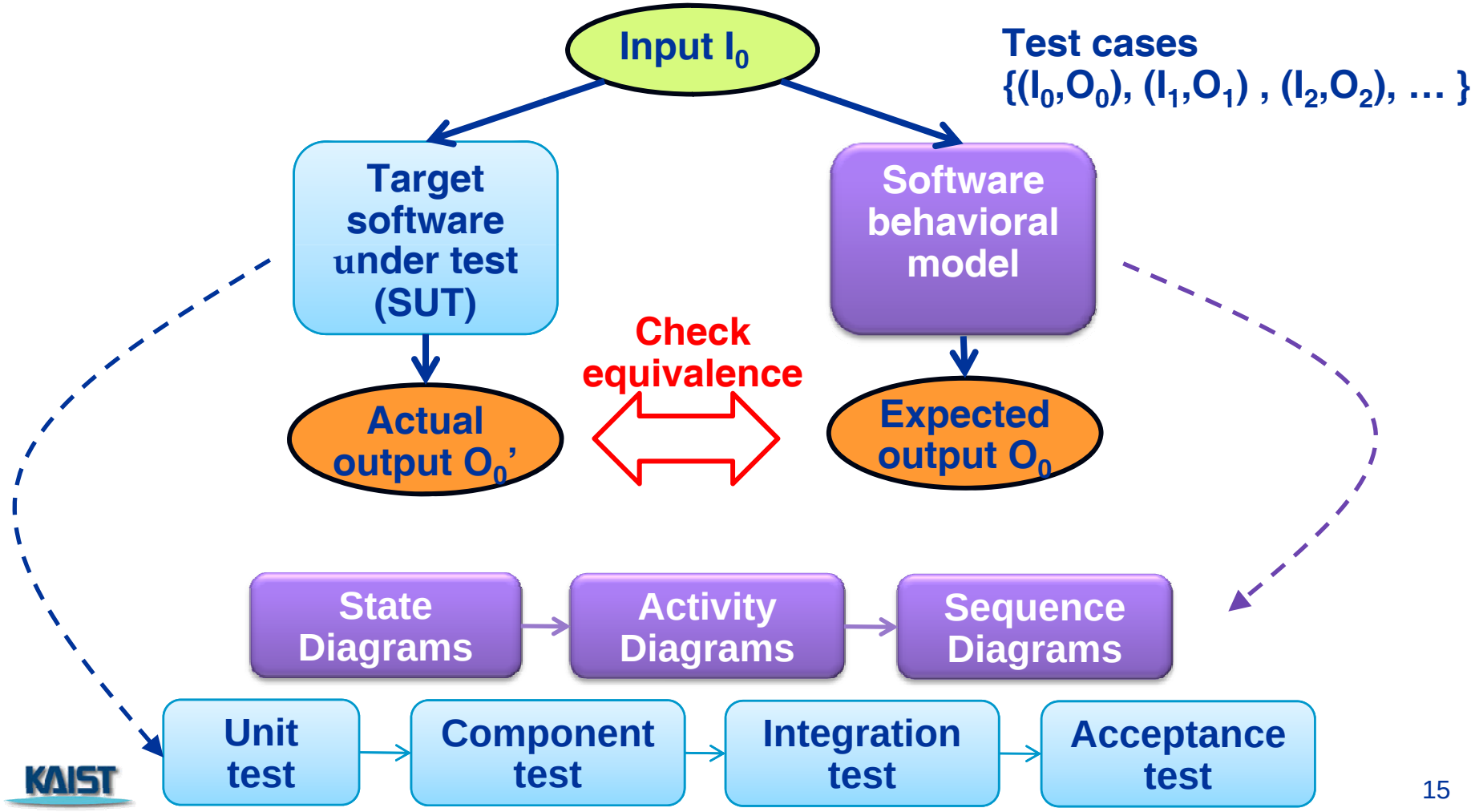
This class is a simple implementation of the abstract class DeviceControlPanelAbstract. All abstract methods are implemented simply to show that each button is working well.

The control panel consists of three parts. It looks like this

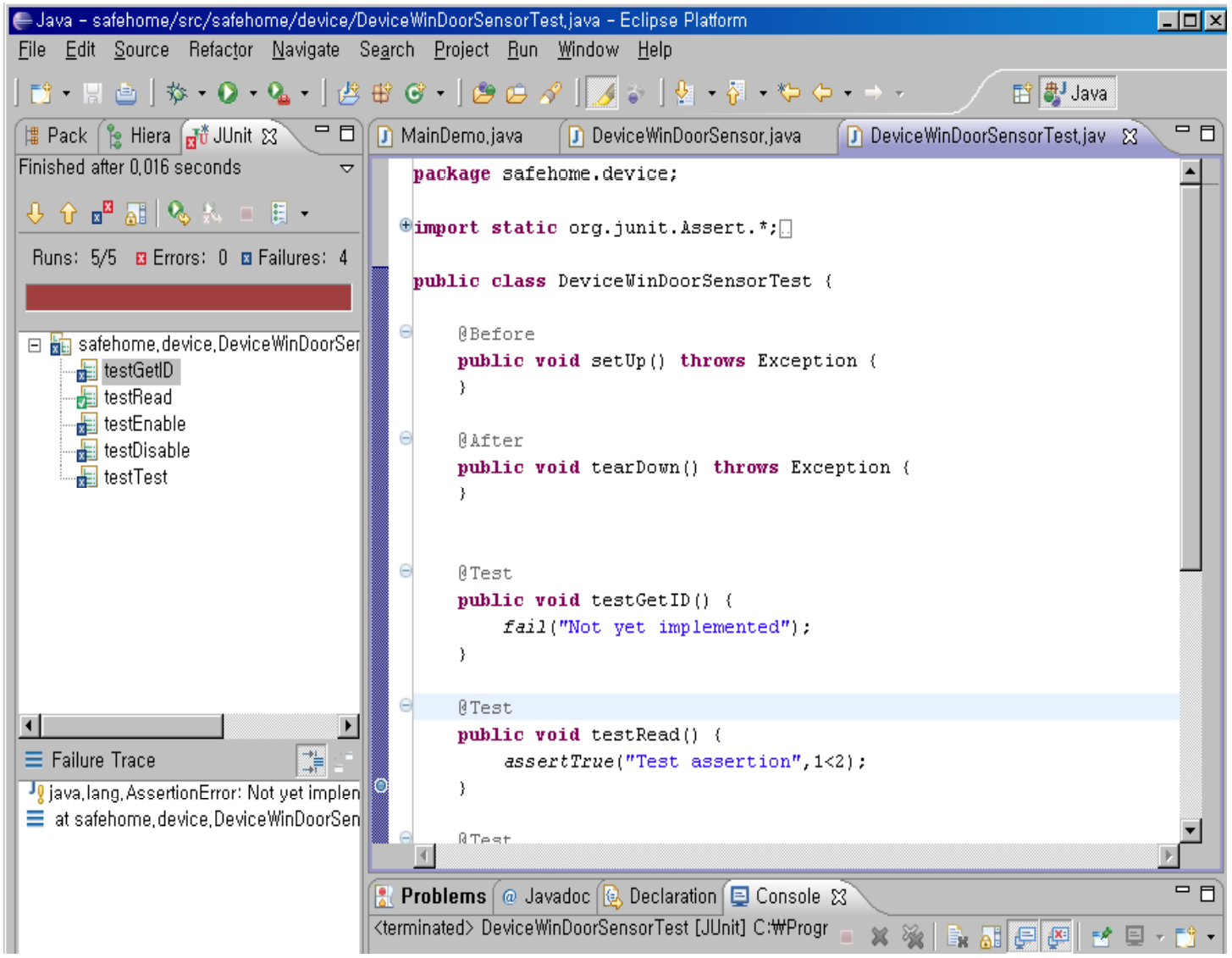



```
/**
 * This class is a simple implementation of the abstract
 * class <code>{@link DeviceControlPanelAbstract}</code>. All abstract methods are
 * implemented simply to show that each button is working well.
 * <p>
 * The control panel consists of three parts. It looks like this
 * <p>
 * <img src = "doc-files/initial.PNG">
 * <p>
 * LCD on the left-up shows the current status, which button is pressed,
 * the system is in stay or away mode, and the system is ready or not.
 * Two LEDs on the left-down indicate the system is armed or not and
 * the system is power-on or off. The keypad on the right-side has 12 buttons,
 * 0 ~ 9, *, #. The corresponding action occurs when a user presses each button.
 *
 *
 * @author cs350 TA
 * @see safehome.device.DeviceControlPanelAbstract
 */
class ControlPanel extends DeviceControlPanelAbstract
{
        /**
         * Constructs a control panel frame.
         *  <p>
         * Pre-condition: None.
         * <p>
         * Post-condition: The control panel frame is ready. It is initially invisible.
         * @see safehome.device.DeviceControlPanelAbstract#DeviceControlPanelAbstract().
         */
        public ControlPanel()
        {
                super();
                setDisplayShortMessage1("        Starting system");
                setDisplayShortMessage2("        please wait...");
        }
```

Model-based Testing



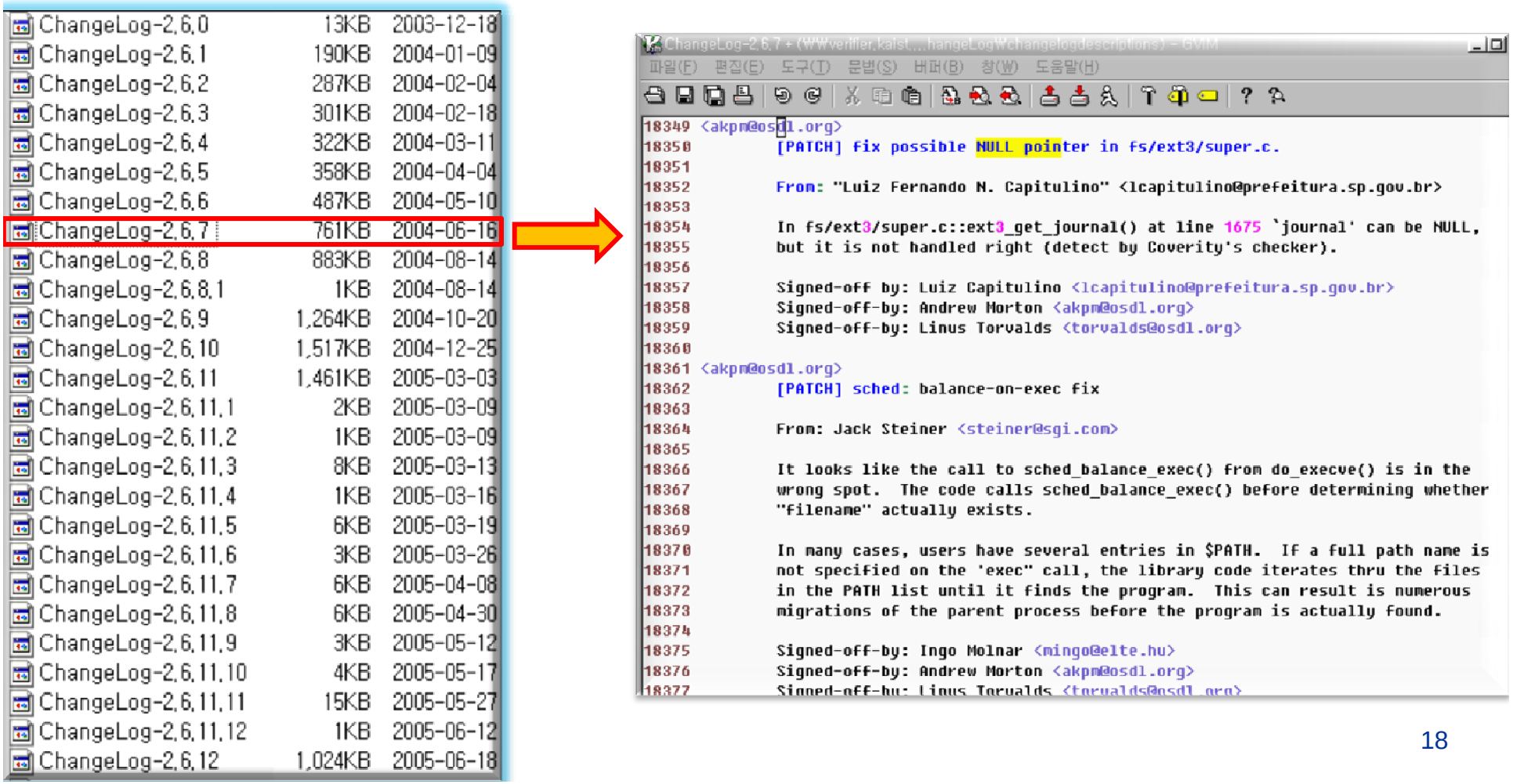
JUnit in Eclipse for Testing



To Do List (1/2)

- Develop software components for devices based on the provided virtual devices in Java and CP in C.
  - You will need network programming to make communication between the devices and CP
  - Make detailed comments as much as possible using JavaDoc
    - Your C code should have rich comments as well
- Perform quality testing
  - Document your test plan and test cases in detail.
    - Provide rationale why selected test cases are good enough
  - You may use advanced testing techniques such as Java Modeling Language (JML), AspectJ or C-BMC (C-bounded model checker)
  - Also, describe bug/revision history
    - See linux kernel changelog as examples

Example of Linux Kernel ChangeLog-2.6.7



To Do List (2/2)

- You have to show traceability between your implementation and your design
  - Each state diagram can be used as a test oracle for unit/component tests
  - Similarly, activity diagram and sequence diagram can be used for component/integration/acceptance testing
- Do not forget to submit the revised design document

Ex. Security function