College of Charleston

Garage Door Opener System
Case Study

Names:
Hector Mojica
Cassel Sloan
Guilherme Menezes

Professor Jim Bowring
Software Architecture and Design

April 23, 2007

Table of Contents

Section 1: Software Requirements Specification

User Level Requirements

1. **Introduction**
   1.1. The product must be a software-based garage door controlling system that operates the garage door motor and the work light through user input.
   1.2. The software must consist of two parts:

   - Software to control the hardware
   - Web-based simulation

2. **Functional Requirements**
   2.1. Control Door
   2.1.1. The door must be stopped at maximum and minimum boundaries.
   2.1.2. The user must be able to make additional input while door is moving.
   2.1.3. The door must move in the opposite direction it moved last time.
   2.1.4. Using Remote Device
   2.1.4.1. The user must not be able to control door with remote device if it is disabled.
   2.1.4.2. The user must be able to open and close door with the remote device.
   2.1.5. Using Stationary Device
   2.1.5.1. The user must be able to open and close door with the first button of the stationary device.
   2.2. Toggle Remote Device
   2.2.1. User must be able to turn the remote device on and off.
   2.2.2. The user must not be able to control door with remote device if it is disabled.
   2.3. Toggle Work Light
   2.3.1. The user must be able to toggle the work light only if the motor has not run in the after a certain amount of time.
   2.3.2. The motor must be able to automatically turn on work light when it runs and keep it on until some time has passed since it has last run.
   2.3.3. The motor must not be able to change the internal state of the work light. Internal state refers to true state of the work light without the auto toggle from the motor.
   2.3.4. If motor has recently run, the user must be able to change the work light's internal state.
   2.3.5. After some time, the work light must return to its internal state.
3. **Non-functional Requirements**
   3.1. Devices should correspond to user input properly.
   3.2. The control of the work light should be returned to the user after it has been five minutes since the motor had last run.
   3.3. Telltale lights must correspond to internal states of the system.
   3.4. The simulation user must be able to perform the same actions as a normal user but in a simulated environment over a web browser.
   3.5. The product must interface with the physical and simulated environment.
   3.6. The product must provide effective feedback and results in web simulation.
4. **Data Requirements**
   4.1. The following should be stored by the system:

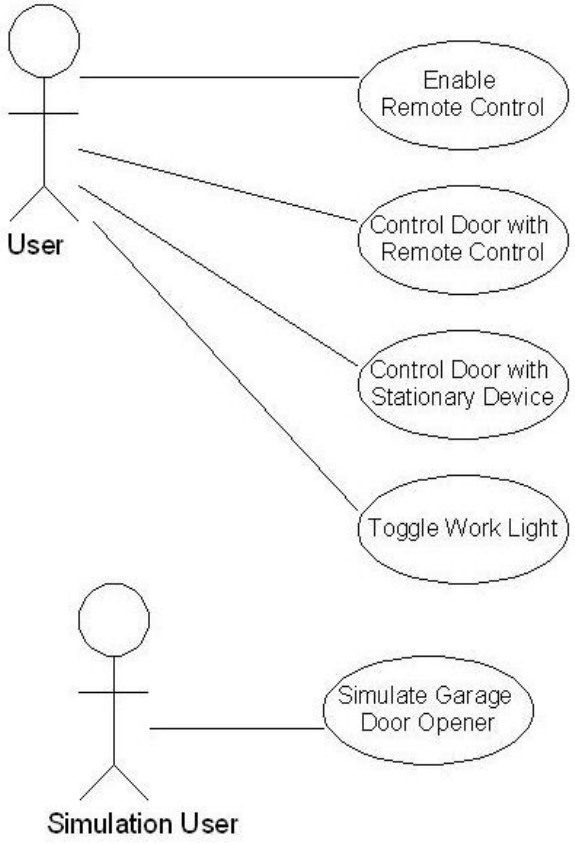
- Time since motor was last run (until five minutes)
- Even-odd count of work light panel button presses
- Last direction motor moved door
- Whether or not the door is running

Use Case Model

Actors

1. **User** – The person who operates the garage door opener.
2. **Simulation user** – The person who interfaces with the Web simulation, and operates the simulated garage door opener.

Use Case Diagram



Use Case Descriptions

Use case 1: Control door with stationary device
Actors: User
Stakeholders and needs:
User – be able to control the door in a timely and convenient manner.
Preconditions:
The door is operational and the interface between user and stationary device is unimpeded.
Post-conditions:
The door must stop moving.
Trigger:
1 – User presses the first button on the stationary device.
Basic Flow:
1 – User presses the first button on the stationary device which turns on the first telltale light on the stationary device until user releases the button.
2 – The non-running motor starts moving the door in the opposite direction as it has moved last time, which is either upwards or downwards.
3 – The motor will continue in this fashion until the door reaches a maximum or minimum height.
Extensions:
2a The motor is running and moving the door downward:
2a1 – The motor will start to move it upwards; the execution continues as planned.
2b The motor is running and moving the door upward:
2b1 – The motor will stop; the use case finishes.
3a The first button of the stationary device is pressed again before door stops:
3a1 – The use case restarts.
3b The button of the remote control is pressed before door stops:
3b1 The basic flow goes to Use Case 2: Control door with remote control.

Use case 2: Control door with remote device
Actors: User
Stakeholders and needs:
User – be able to control the door in a timely and convenient manner.
Preconditions:
The door is operational.
Post-conditions:
The door must stop moving.
Trigger:
1 – User presses the button on the remote device.
Basic Flow:
1 – User presses the button on the remote device which turns on remote's telltale light until user releases the button.
2 – The non-running motor starts moving the door in the opposite direction as it has moved last time; upwards or downwards.
3 – The motor will continue in this fashion until the door reaches a maximum or minimum

height.

Extensions:

1a If the remote device is disabled, then the use case ends.

2a The motor is running and moving the door downward:

2a1 –The motor will start to move it upwards; the execution continues as planned.

2b The motor is running and moving the door upward:

2b1 – The motor will stop; the use case finishes.

3a The first button of the stationary device is pressed before door stops:

3a1 – The basic flow goes to Use Case 1: Control door with stationary device.

3b The button of the remote control is pressed again before door stops:

3b1 – Use Case 2 restarts.

Use case 3: Toggle Work Light

Actors: User

Stakeholders and needs:

User – be able to toggle the light.

Preconditions:

The work light is operational.

Post-conditions:

None

Trigger:

1 – User presses the third button on the stationary device.

Basic Flow:

1 – User presses the third button on the stationary device.

2 – The work light is turned on if it is off and turned off if it is on; so does the telltale light.

Extensions:

2a The door has run within the last 5 minutes:

2a1 – The work light will not correspond exactly to user input and will always be on, but it will change the internal representation of the state of the light; however, the telltale light will respond, and will correspond to the internal representation of the state of the work light.

Use case 4: Enable Remote Control

Actors: User

Stakeholders and needs:

User – Needs turn on and off the remote control device.

Preconditions:

None

Post-conditions:

None

Trigger:

1 – The user presses the second button on the stationary device.

Basic Flow:

1 – The user presses the second button on the stationary device.

2 – The remote is enabled if disabled or disabled if enabled; if the remote control is deactivated

the light is on, else the light is off.

Use case 5: Simulate Garage Door Opener
Actors: Simulation User
Stakeholders and needs:
Simulation User – Needs to simulate the operation of the Garage Door Opener system in a Web environment
Preconditions:
None
Post-conditions:
The simulation was accurate.
Trigger:
1 – The simulation user loads the simulation Web page
Basic Flow:
The actions listed bellow can be done in any order, or repeatedly.

1 – The simulation user uses the simulated remote or stationary device to control the door.
2 – The simulation user uses the simulated stationary device to toggle the simulated work light.
3 – The simulation user uses the simulated stationary device to toggle the simulated remote device.
4 – The system simulator displays the state of the motor, the door, the work light, the remote device and the telltale lights.

The simulation ends when the simulation Web page is exited.

Design rationale

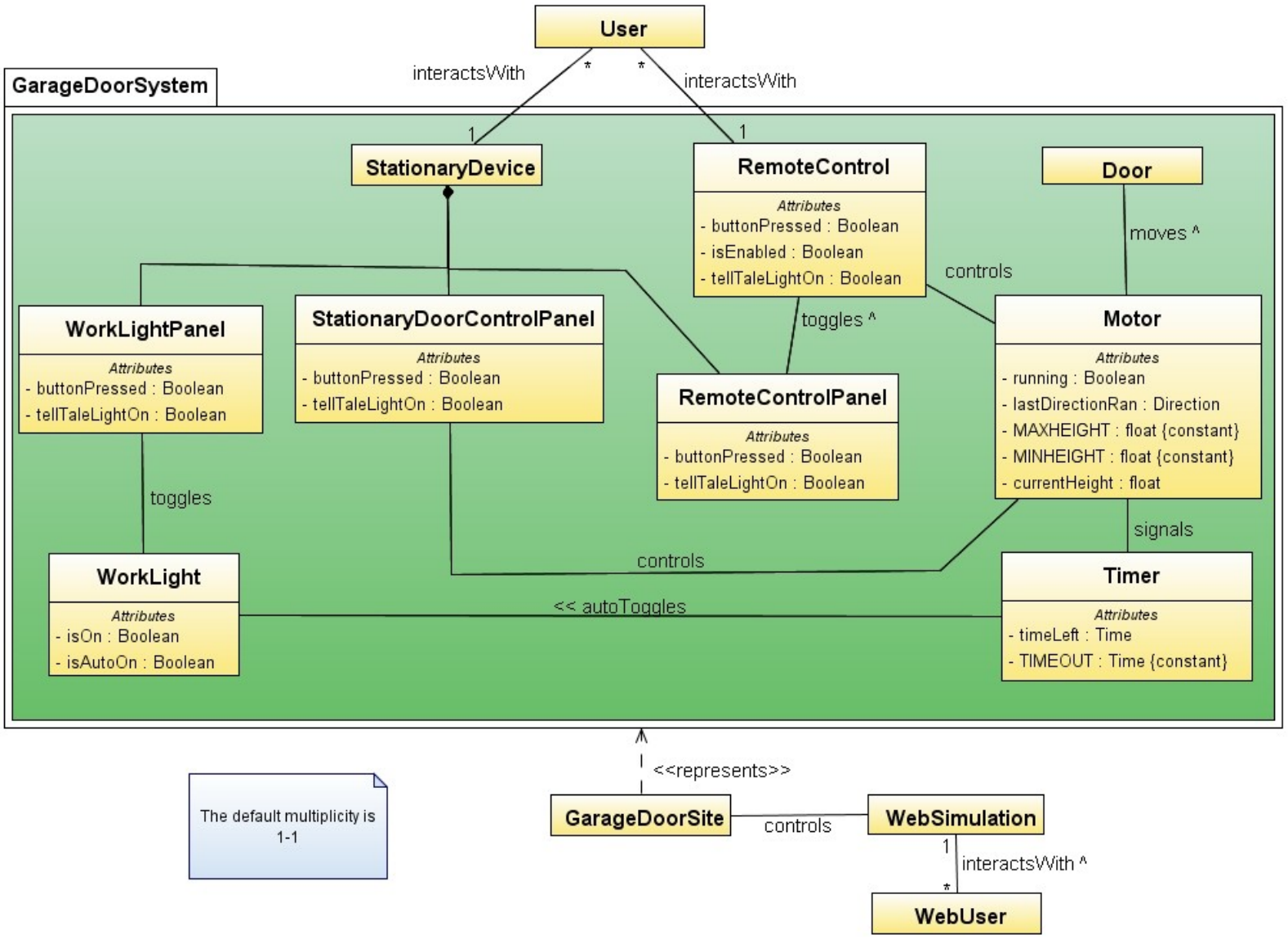
We decided to write two different Use Cases for controlling the door because they have different triggers and different preconditions.

At first, we decided to have two separated Use Cases for toggling the work light. The first one (originally Use Case 3) is triggered by the user, while the second one (originally Use Case 4) is triggered by the motor automatically. However, it was decided that the auto-toggling of the work light is just an internal function of the system that does not need a use case.

The “internal representation of the work light state”, referenced in Use Cases 3 and 4, was introduced here because the user can perceive it. We preferred to use the term “internal representation” as an abstraction for binary representation (even-odd) because “internal representation” corresponds to high level user requirements while binary representation corresponds with lower level requirements.

Section 2: Software Architecture Design

Garage Door Opener System Conceptual Model



Decomposition Rationale

We decided to use the *GarageDoorSystem* package to represent the whole system for the *GarageDoorSite* to represent. It should be noted that all the *GarageDoorSystem* components are simulated in the Web simulation by the *GarageDoorSite*.

The *GarageDoorSite* represents the system inside the *WebSimulation*. The *WebSimulation* is used by the *WebUser* to simulate the control of the Garage Door Opener System.

Since the *StationaryDevice* contains three different control panels, with different responsibilities, we thought that it would be best to represent the stationary device as a composition of the three controllers. It also creates a way to represent the interaction of the user and the control panels in a compact way.

The *Motor* signals the *Timer* when it starts to move. The *Timer* starts to count down until it reaches zero or until the *Motor* moves again. While counting down, the *Timer* auto-toggles *WorkLight* on, making isAutoOn true. When the timer reaches zero, the *Timer* auto-toggles it off, making isAutoOn false.

If isAutoOn is true, the *WorkLight* is on automatically. If isAutoOn is false, then the *WorkLight* takes on the state controlled by the *WorkLightPanel*.

As noted from above, the default multiplicity of all associations was defined as being 1-1. We preferred to define a default to make the diagram easier to read.

The attributes running and lastDirectionRan in *Motor* class can describe all possible states the motor may have; if it is running, the direction it is running is the opposite direction it has last ran. Keeping the direction it last ran is necessary to know at the time of activation of the motor the direction the motor should go.

Profiles and Scenarios

Scenarios

1. Utility
    1. Usage
        1. Simulate Garage Door Opener (High)
        2. Control Door (High)
        3. Toggle Work Light (Medium)
        4. Disable/enable Remote Control (Medium)
    2. Reliability
        1. Door Stuck (High)
        2. No Signal From Devices (Medium)
    3. Modifiability
        1. Change Work Light Timeout (Low)
        2. Modify Motor Behavior (Low)
    4. Performance
        1. Web Page Loading (Medium)
    5. Hardware Adaptability
        1. Replace Motor (Medium)
        2. Replace Remote Control (Medium)
    6. Software Availability
        1. Web Server Availability (Medium)

Usage Profiles

Simulate Garage Door Opener
The user loads the Web simulation page and starts the simulation. The user interacts with the simulation as if it was the real product.

Control Door
The user presses the Remote Control or the Stationary Door Control Panel button. The door reacts, depending on its last behavior and current state.

Toggle Work Light
The user can turn the work light on and off, depending on when the motor has last run.

Disable/enable Remote Control
The user presses the button on the Remote Control Panel to turn the remote device on and off. When off, the remote control cannot send any signals to the motor.

Reliability

Door Stuck
The door becomes stuck. The system realizes it if the motor takes too long to stop running. The system turns off the whole Garage Door Opener and all its devices.

No Signal From Door Controlling Devices
If no signal is being sent from the Remote Control Device or from the Stationary Door Control Panel while the user is pressing the button, the telltale light in these devices is not turned on.

Modifiability

Change Work Light Timeout
The developers need to change the amount of time the work light will be on after the motor has last run. They could make this modification in less than one hour.

Change Motor Behavior
The developers need to change the behavior the running motor has when responding to user input, e.g. changing the behavior of motor when pulling the door up. They should be able to make this modification in one week.

Performance

Web Page Loading
The Web simulation should be able to respond to user input in a timely fashion. Particularly the initial loading time should not take long.

Hardware Adaptability

Replace Motor
The maintenance worker should be able to change the motor and the system should be able to interface with the new motor. Minor changes in software might be necessary.

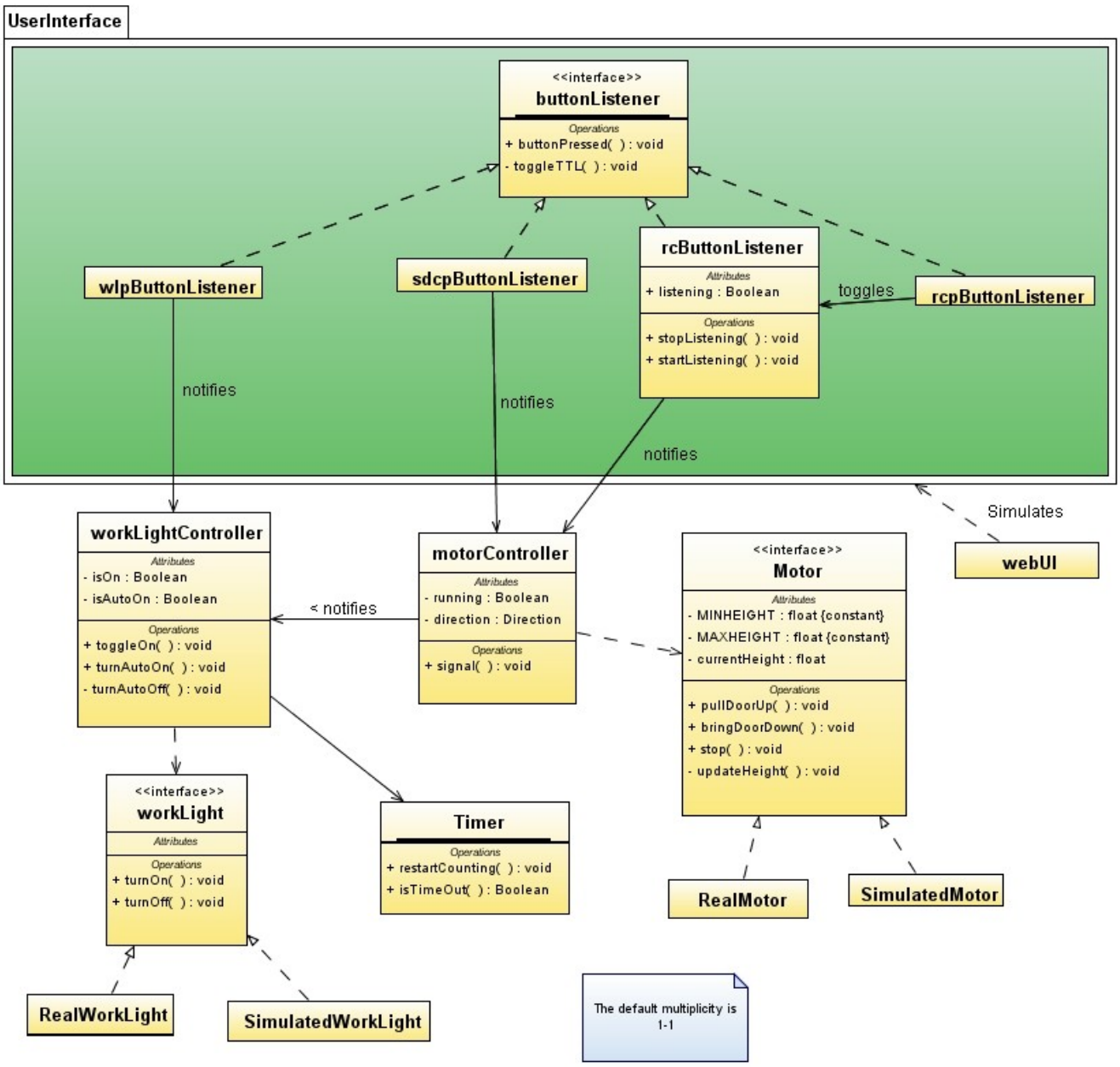
Replace Remote Control
The user should be able to replace the remote control with a compatible one with little technical assistance, and the system should be able to recognize it.

Software Availability

Web Server Availability
The user should be able to access the Web simulation page 95% of the day.

Section 3: Detailed Design Document

Low-Level Class Diagram



Rationale

The idea behind the button listener classes is to provide an observer class that reacts when a button is pressed. It was based on diagram 12-1-8 at page 365.

The Remote Control Panel Button Listener just tells the Remote Control Button Listener to stop listening when it has to disable the Remote Control.

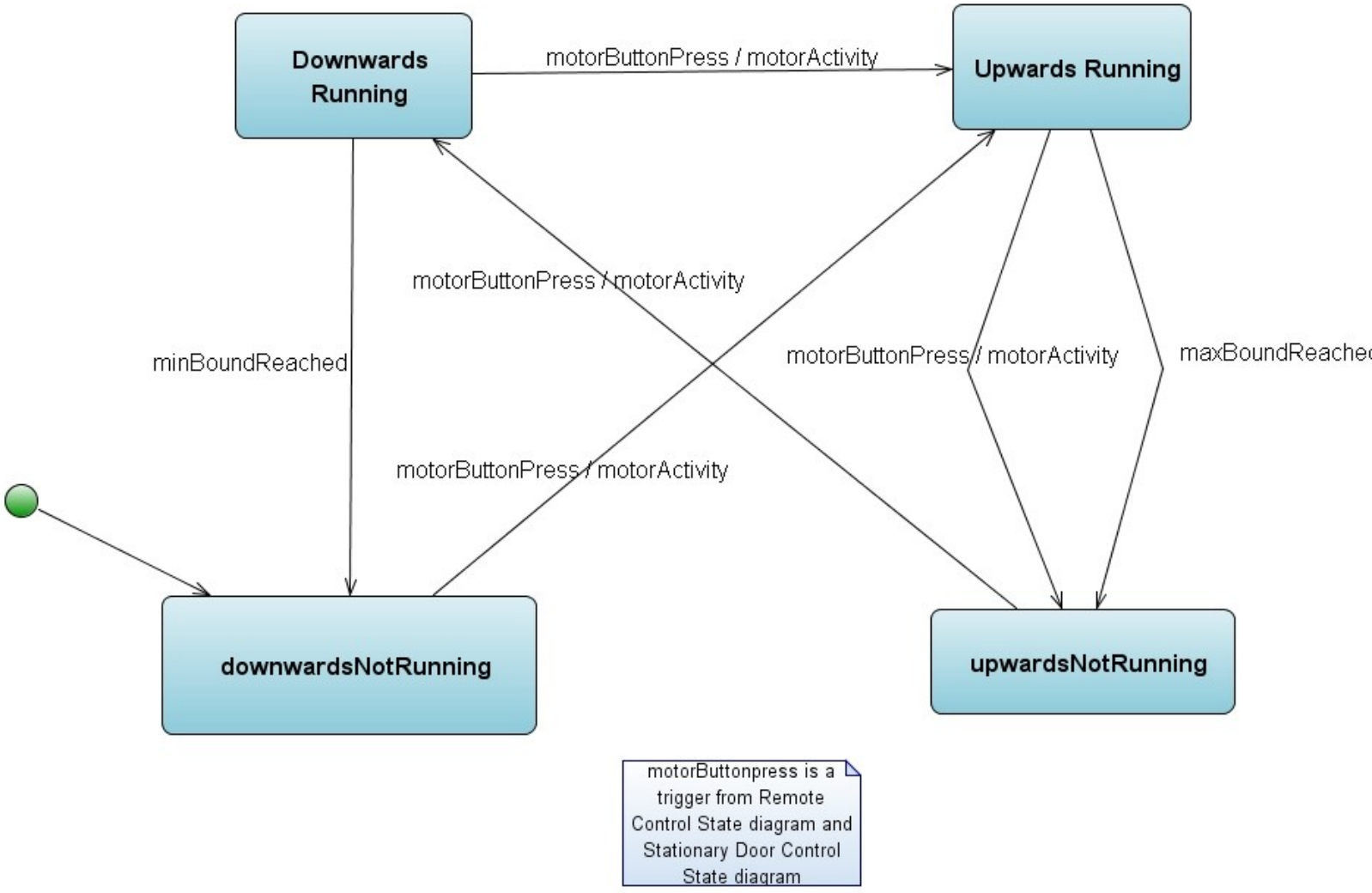
We decided to use the package UserInterface to represent the different components the user directly interacts with, as well as the Web user (which will use the webUI class).
Worklight and Motor are defined as interfaces so that we can directly interact with both the real and the simulated devices.

The webUI simulates the interface between the simulation user and the system. The simulation user can only interact with the system using webUI.

It should be noted that the web site will show the entire garage door opener system, including components such as the motor and work light, represented in the diagram as SimulatedWorkLight and SimulatedMotor.

State Diagrams

Motor State



Description

Describes the behavior of the motor given user input.

Events:

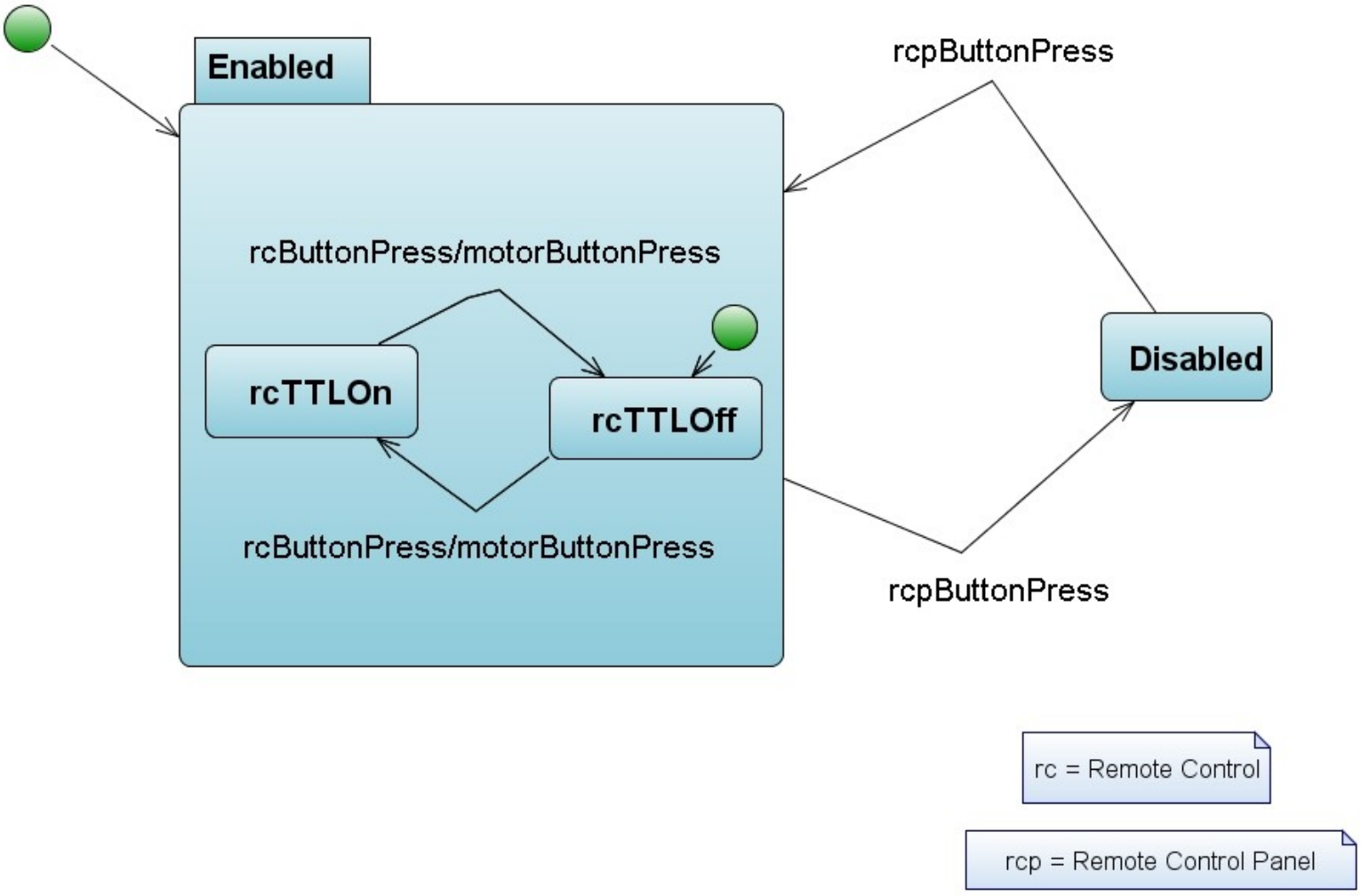
motorButtonPress – Remote Control or Stationary Door Control button press
minBoundReached – minimum door height was reached
maxBoundReached – maximum door height was reached

Actions:
motorActivity – event triggered when a button is pressed;

States:
Downwards Running – motor is running and bringing the door down
Upwards Running – motor is running and pulling the door up
Downwards Not Running (initial) – the door is stopped and the last direction ran was downwards
Upwards Not Running - the door is stopped and the last direction ran was upwards

Relationships with other models:
motorActivity action will be listened by other diagrams (Work Light state diagram)
motorButtonPress is triggered by Remote Control State diagram and Stationary Door Control State diagram

Remote Control



Description

Describes the toggling of the remote control.

Events:

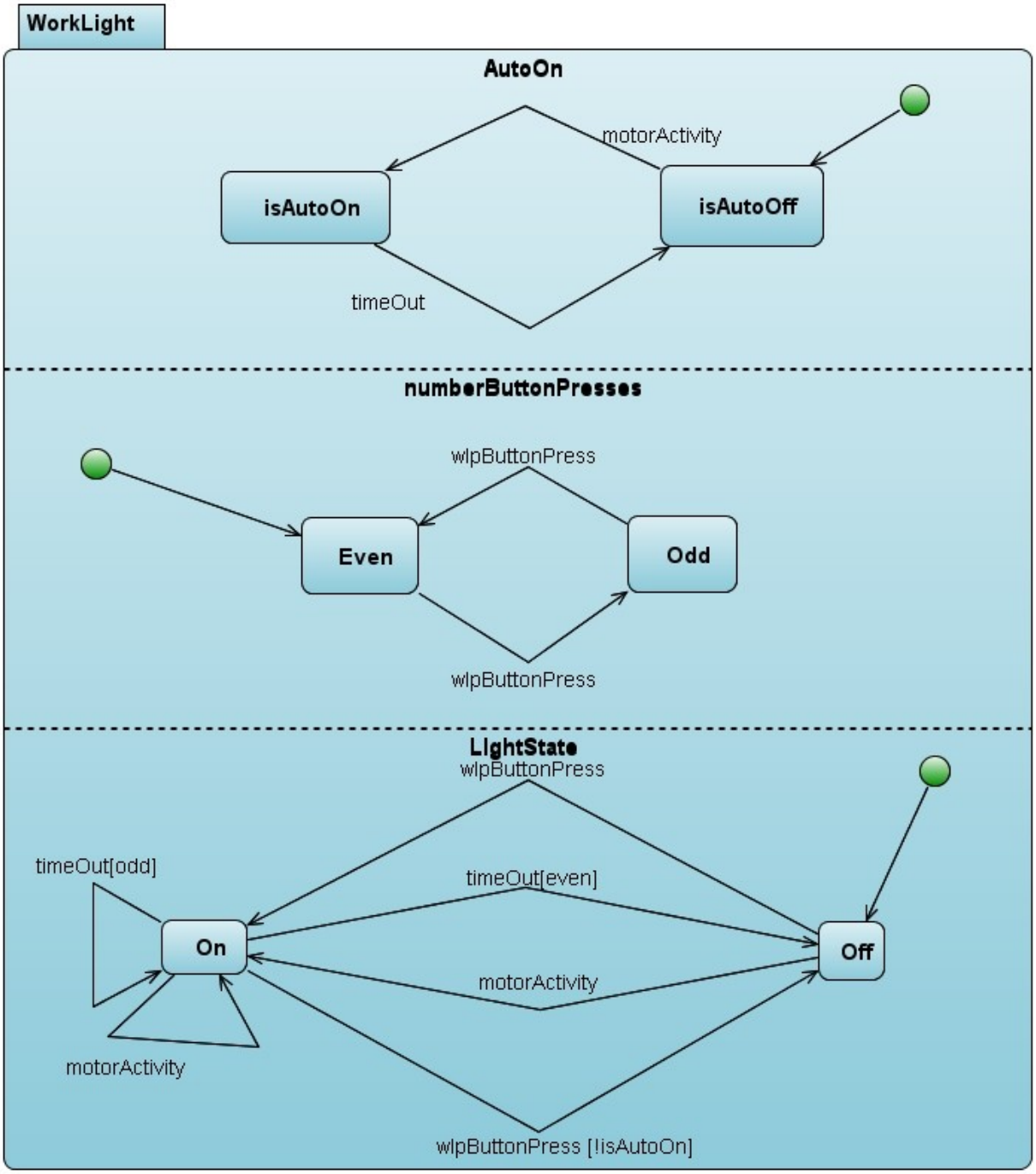
rcpButtonPress – Remote Control Toggling Panel button press
rcButtonPress – Remote Control button press

Actions:
motorButtonPress – triggers motor activity

States:
Enable (initial) – remote control is enabled
  rcTTLOn – telltale light of remote control is on
  rcTTLOff – telltale light of remote control is off
Disable – remote control is disabled

Relationships with other models:
motorButtonPress triggers activity in Motor State diagram

Work Light



Description

It has three concurrent diagrams specifying the state of the Work Light. This includes the whether or not the light will automatically turn and remain on, the number of button presses (even or odd count)

and the light state (on or off).

AutoOn concurrent diagram

Events:
motorActivity – self-descriptive and described earlier
timeOut– timer counting is over

States:
isAutoOn
IsAutoOff (initial)

numberButtonPresses concurrent diagram

Events:
wlpButtonPress – Work Light Panel button press

LightState concurrent diagram

Events:
wlpButtonPress - Work Light Panel button press
timeOut – time counting is over
motorActivity

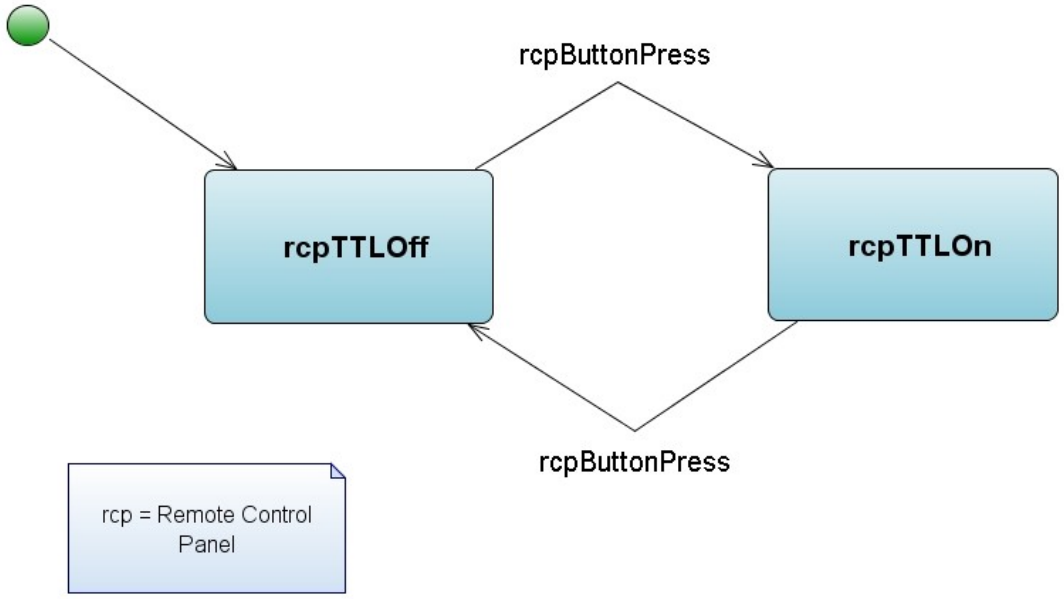
Guards:
Odd – numberButtonPresses is in Odd state
Even – numberButtonPresses is in Even state
!isAutoOn – the variable isAutoOn is false.

States:
On – work light is on
Off (initial) – work light is off

Relationships with other models

motorActivity is triggered by the **Motor State** state diagram

Remote Control Panel



Description

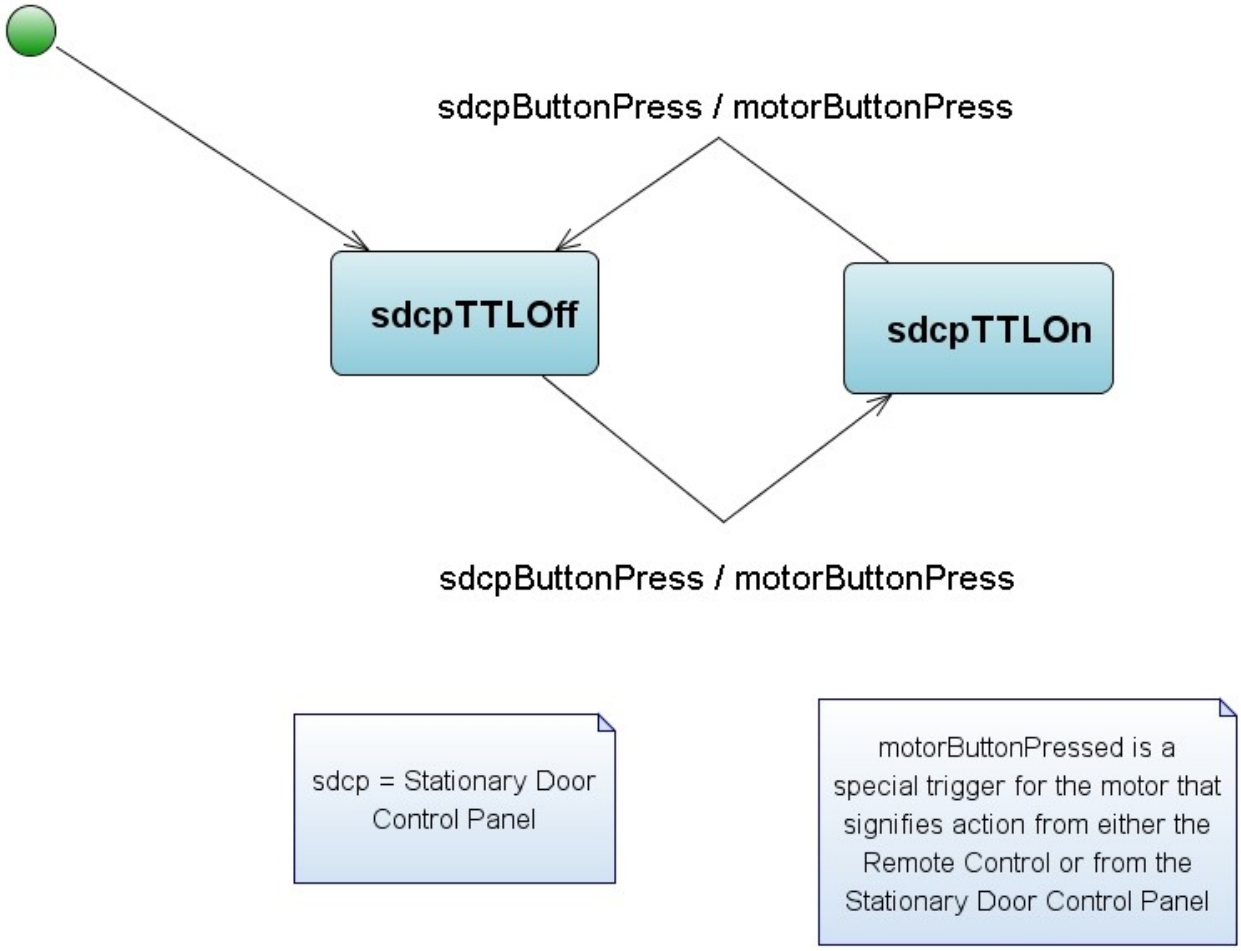
Describes the state of the telltale light of the remote control panel.

Events:
rcpButtonPress – Remote Control Toggling Panel button press

States:
rcpTTLOn – telltale light of the remote control panel is on
rcpTTLOff – telltale light of the remote control panel is off

Relationships with other models:
Remote Control State diagram is closely related to this diagram. The event *rcpButtonPress* is the same in the two diagrams.

Stationary Door Control Panel



Description

Describes the state of the telltale light of the stationary door control panel.

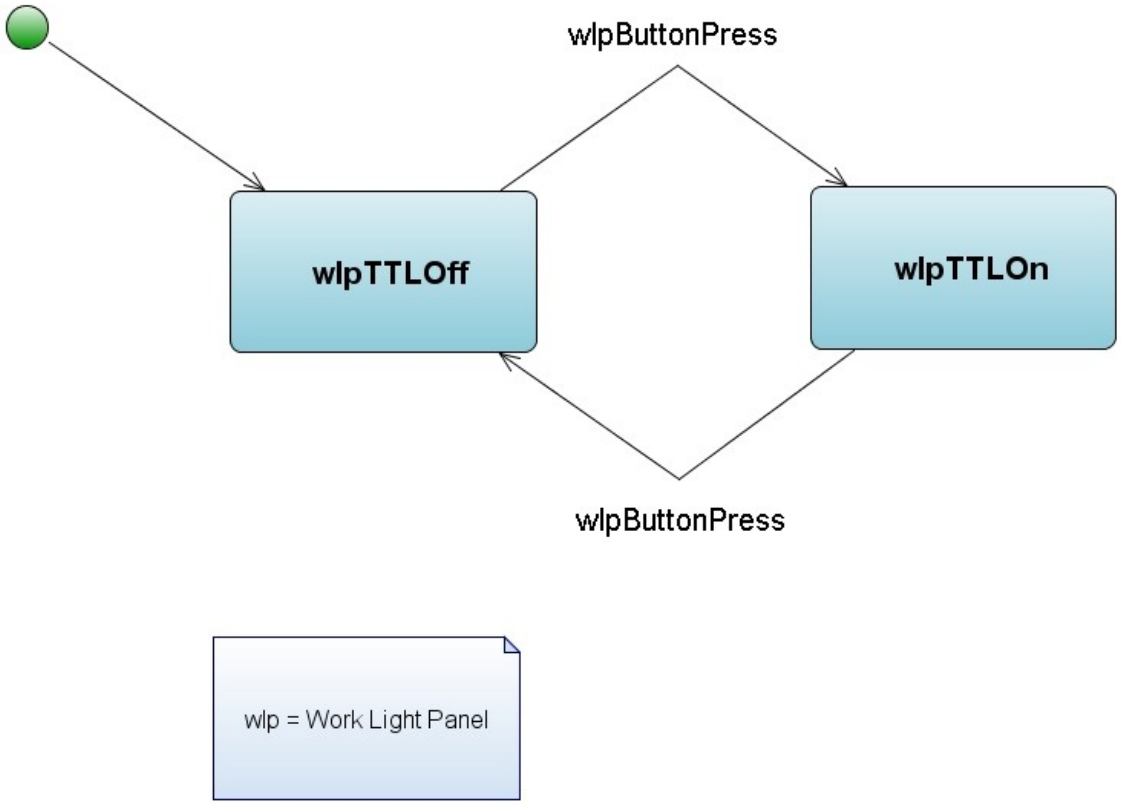
Events:
sdcpButtonPress – Stationary Door Control Panel button press

Actions:
motorButtonPress – triggers motor activity

States:
sdcpTTLOn – telltale light of the stationary door control panel is on
sdcpTTLOff – telltale light of the stationary door control panel is off

Relationships with other models:
Remote Control State diagram is closely related to this diagram. The event *rcpButtonPress* is the same in the two diagrams.

Work Light Panel



Description

Describes the state of the telltale light of the work light panel.

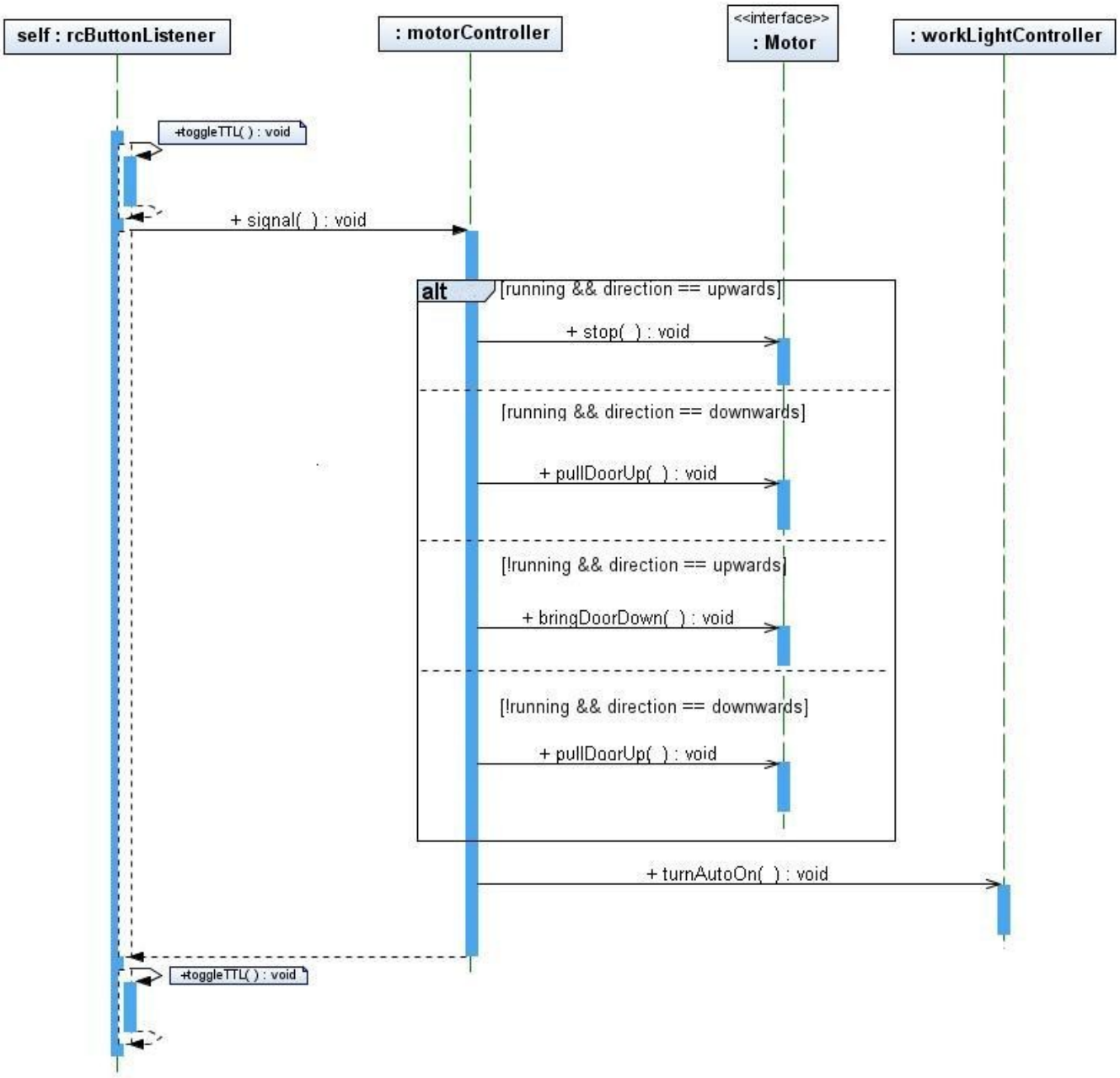
Events:
wlpButtonPress – Work Light Panel button press

States:
wlpTTLOn – telltale light of the work light panel is on
wlpTTLOff – telltale light of the work light panel is off

Relationships with other models:
Work Light State diagram is closely related to this diagram. The event *wlpButtonPress* is the same in the two diagrams.

Sequence Diagrams

Remote Door Control - **rcButtonListerner.buttonPressed()**



Description

This diagram specifies interactions between objects after a call of *rcButtonListerner.buttonPressed()*.

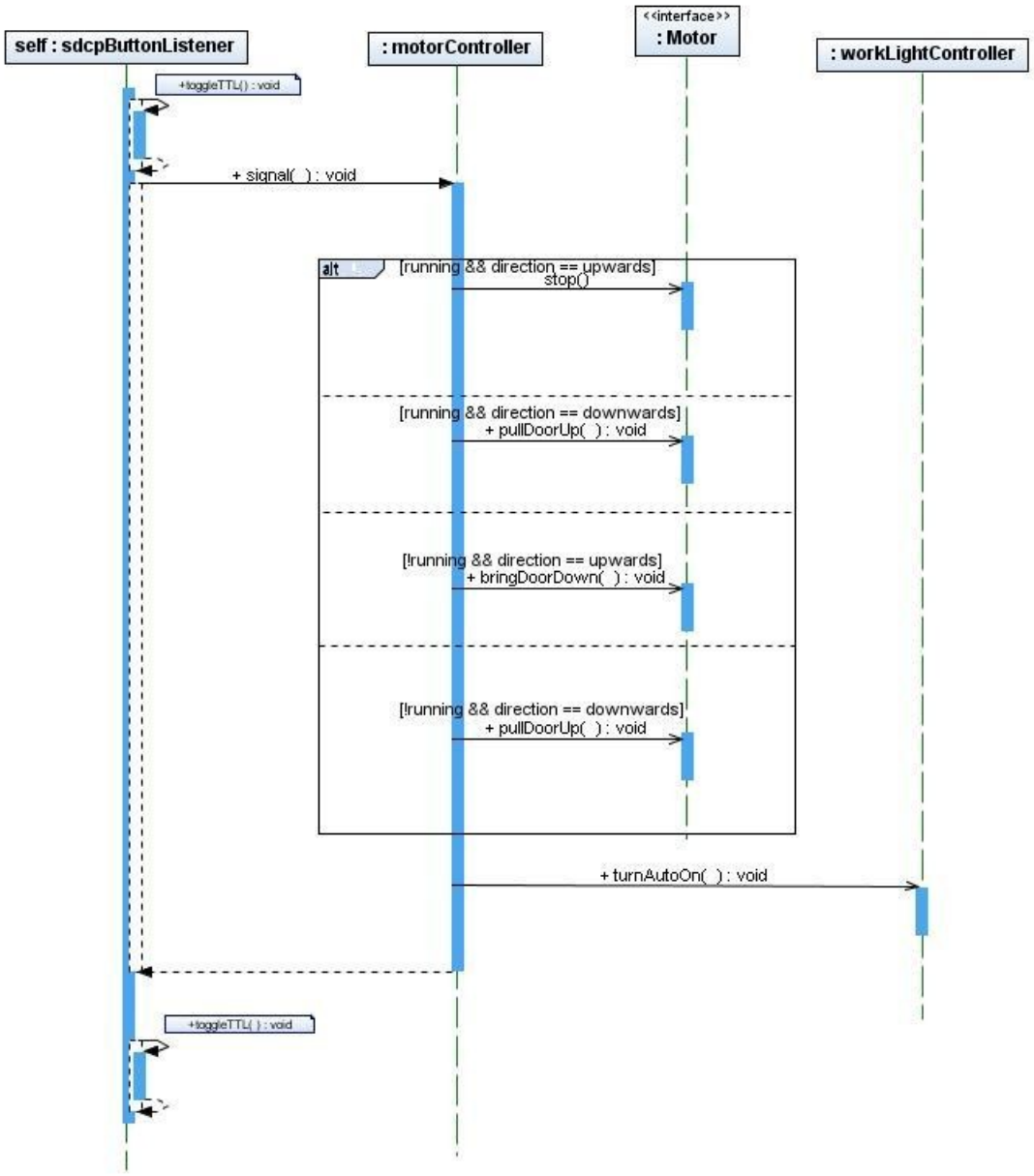
It is important to notice that, although not explicitly expressed in the diagram, this interactions will

only occur if *rcButtonListerner.listening* is true.

First, the *rcButtonListerner* (self) toggle its own telltale light on. It then calls *motorController.signal()*. As specified in the state diagram **Motor State**, the motor can send one of three messages to the *Motor* interface, depending on the current state of the motor (running and direction). Then, *workLightController.turnAutoOn()* is called, and *rcButtonListener* turn is on telltale light off.

Relationships between **rcButtonListerner.buttonPressed()** and other models:
workLightController.turnAutoOn() is specified at the sequence diagram with the same name.
Which message is sent to *Motor* by *motorController* is specified in **Motor State** state diagram.
The name of the operations are the same as in the Low Level Class Diagram.

Stationary Door Control – **sdcpButtonListener.buttonPressed()**

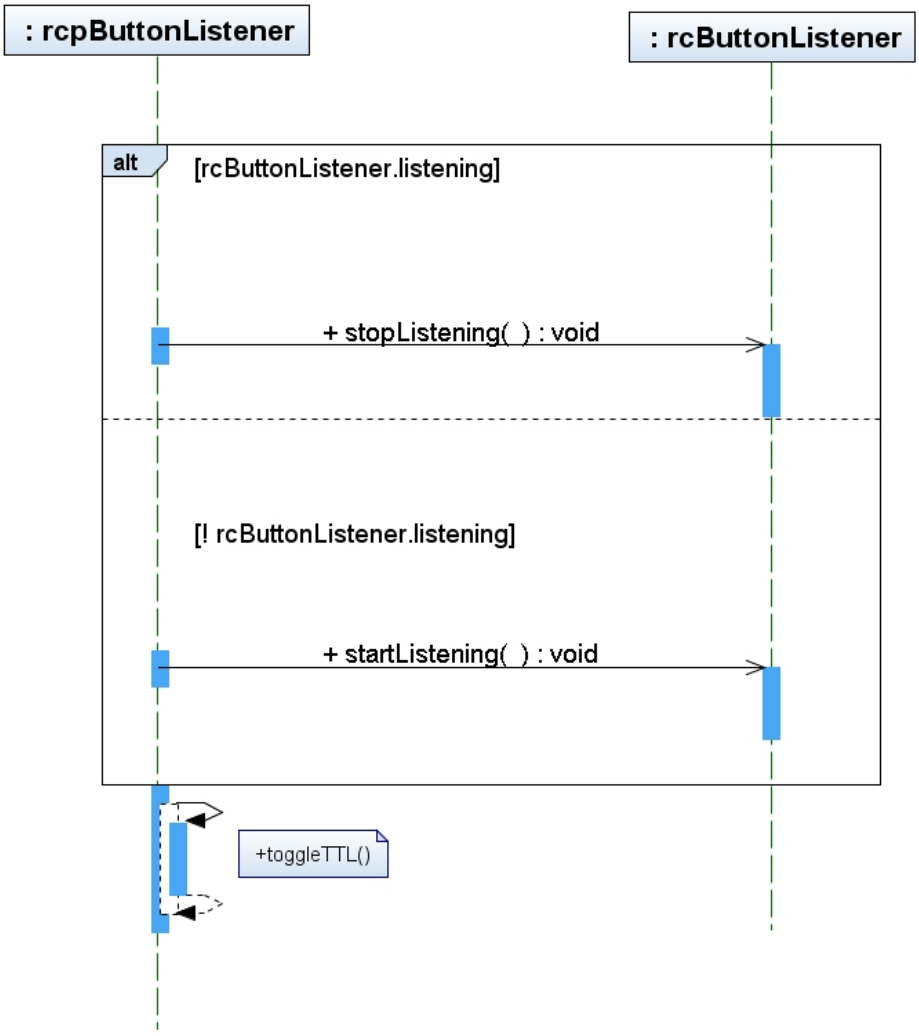


Description

This diagram is exactly the same as *rcButtonListerner.buttonPressed();* the only difference here is that *self* is a sdcpButtonListerner.

Relationships between **rdcpButtonListerner.buttonPressed()** and other models:
workLightController.turnAutoOn() is specified at the sequence diagram with the same name.
Which message is sent to *Motor* by *motorController* is specified in **Motor State** state diagram.
The name of the operations are the same as in the Low Level Class Diagram.

Remote Control Toggling – **rcpButtonListener.buttonPressed()**



Description

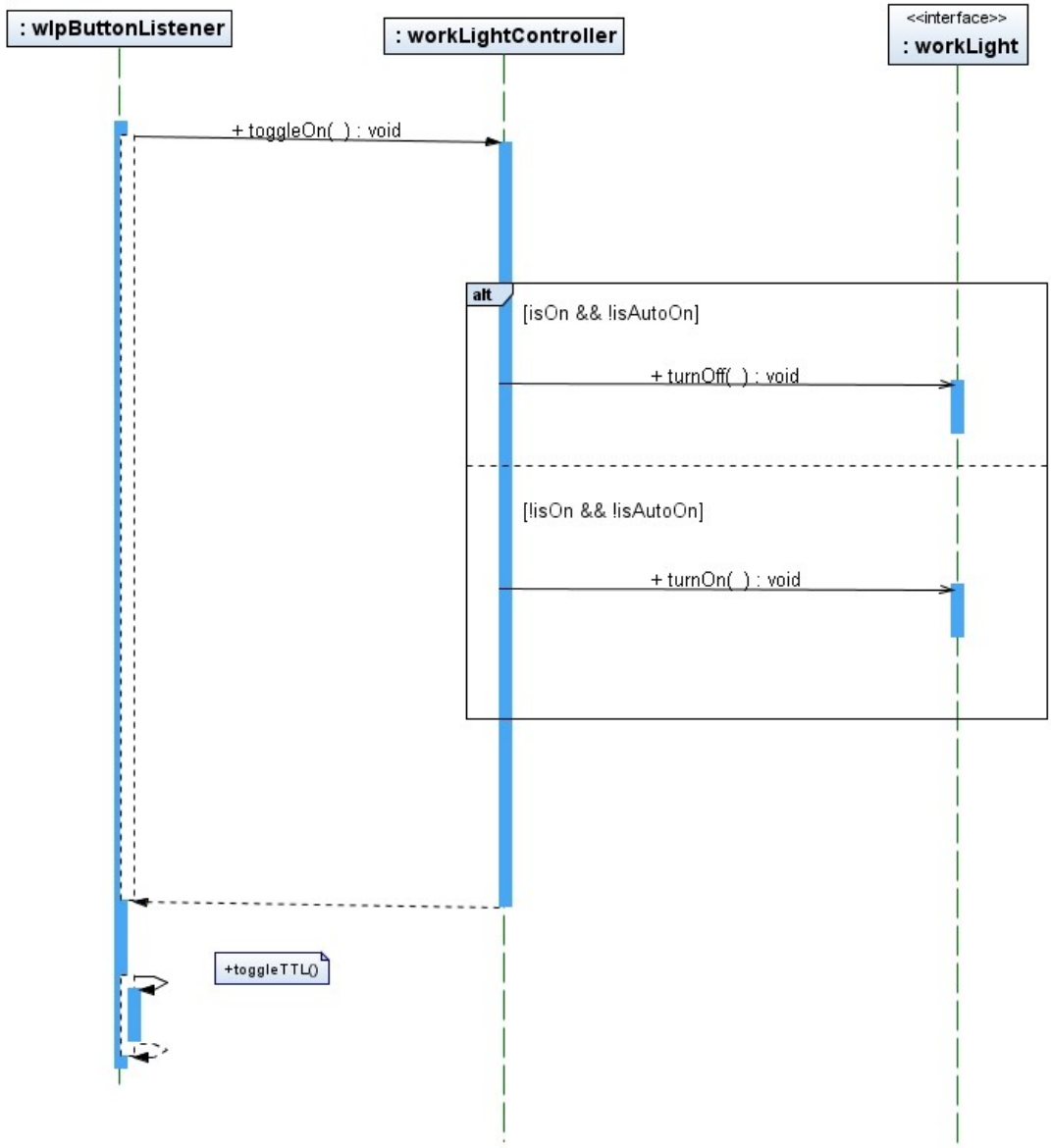
This diagram models what happens when *rcpButtonListener.buttonPressed()* is called.

rcpButtonListener is going to send a *rcButtonLister.stopListening()* message if *rcButtonListener.listening* is true, and a *rcButtonLister.startListening()* message otherwise.

Then, the telltale light of *rcpButtonListener* is toggled.

Relationships between **rcpButtonListerner.buttonPressed()** and other models:
The name of the operations are the same as in the Low-Level Class Diagram.

Work Light Panel - **wlpButtonListener.buttonPressed()**



Description

This diagram models what happens when wlp*ButtonListener.buttonPressed()* is called.

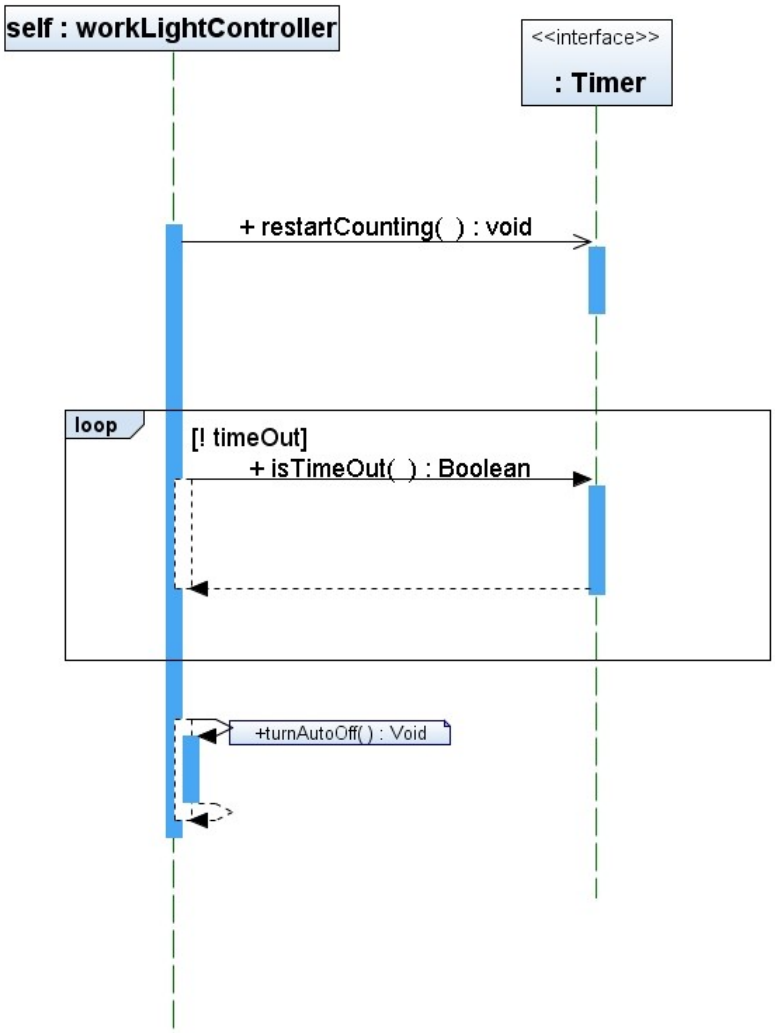
The *wlpButtonListener* sends a *toggleOn()* message to *workLightController*, that in turn sends a *turnOff()* or *turnOn()* message to the *workLight* interface, depending on the state of the *isOn* variable in *workLightController*. (See **Work Light** state diagram)

Notice that messages are sent to the *workLight* interface only if *isAutoOn* is false. (See **Work Light** state diagram)

The Work Light Panel telltale light is then toggled.

Relationships between **wlpButtonListerner.buttonPressed()** and other models:
Which message is sent to the workLight interface is define by the **Work light** state diagram.
The name of the operations are the same as in the Low Level Class Diagram.

Auto Work Light Toggle - **workLightController.turnAutoOn()**



Description

This diagram models what happens when *workLightController.turnAutoOn()* is called.

The counter is restarted in *Timer.* The the Timer indicates a timeout, *turnAutoOn()* is called.

Relationships between **workLightController.turnAutoOn()** and other models:
The fact that autoOn is true of false affects the operation of *wlpButtonListener.buttonPressed()*.
The name of the operations are the same as in the Low Level Class Diagram.