Проектирование и разработка системы «Умный дом». Клиентская часть

Угольникова Н.Б., студентка 3 курса физико-математического факультета

Руководитель Гилёв А.Ю.

г. Бирск, ФГБОУ ВПО Бирский филиал БашГУ

Введение

Умный дом — жилой дом современного типа, организованный для проживания людей при помощи автоматизации и высокотехнологичных устройств. Под «умным» домом следует понимать систему, которая обеспечивает безопасность, комфорт и ресурсосбережение для всех пользователей. В простейшем случае она должна уметь распознавать конкретные ситуации, происходящие в доме, и соответствующим образом на них реагировать: одна из систем может управлять поведением других по заранее выработанным алгоритмам. Кроме того, от автоматизации нескольких подсистем обеспечивается синергетический эффект для всего комплекса.

Клиентская часть системы «SmartHouse» должна делать следующее:

1. получать информацию о состоянии системы от сервера
2. осуществлять визуализацию этой информации
3. предоставлять возможность пользователю вручную управлять отдельными устройствами
4. предоставлять возможность пользователю добавлять новые устройства и правила для них

1. Дерево форм

На рисунке 1 представлено дерево форм.

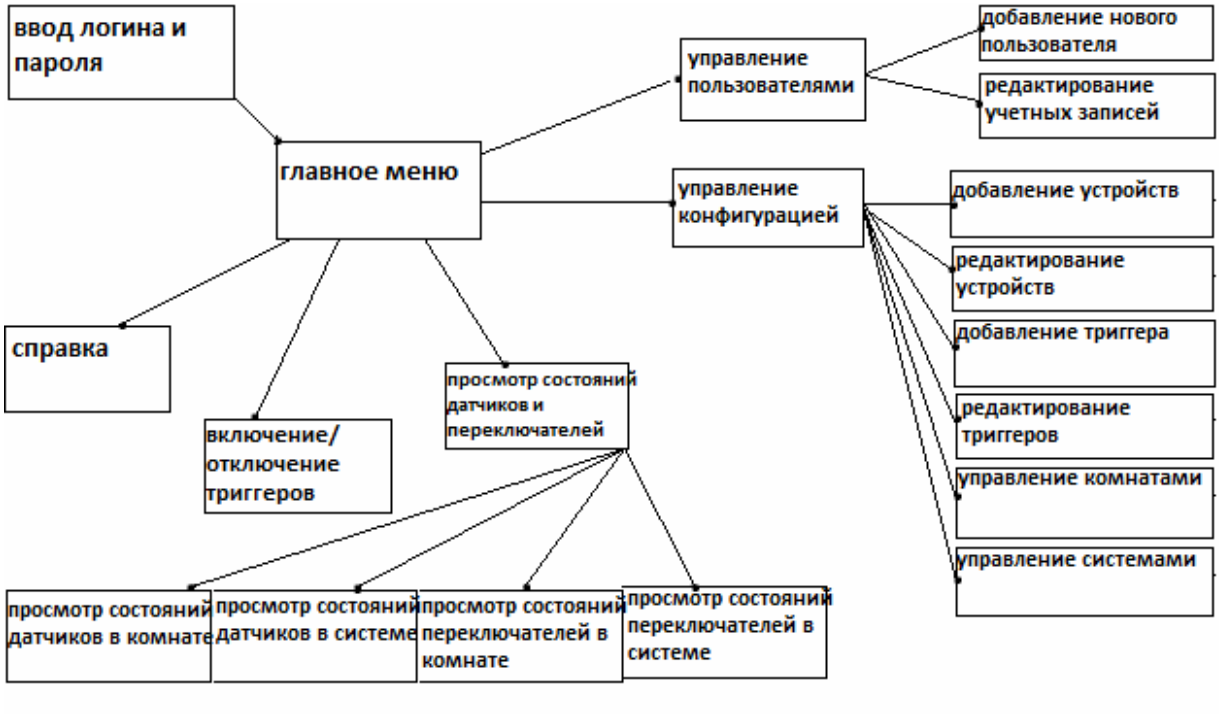

Рис.1 Дерево форм

2. Описание программы

В главном классе SmartHouseClient объявляем несколько полей:

```
public static Stage myStage;
public static RMIConnection connection;
public static RemoteCoreController controller;
public static FXMLLoader ldr;
```

RMIConnection connection - соединение с сервером, RemoteCoreController controller - контроллер сервера, полученный при авторизации, FXMLLoader ldr - загружает иерархию объектов из XML-документа.

Метод connectToServer() осуществляет подключение к серверу:

`connection = new RMIConnection();-` создается соединение

`connection.connectToRMIServer("127.0.0.1",20202);` - осуществляет подключение к серверу: 127.0.0.1 - адрес сервера, 20202 - порт.

При запуске приложения появляется форма «Вход в систему»:

```
@Override
    public void start(Stage stage) throws Exception {
        connectToServer();
        myStage=stage;
```

```
        ldr = new FXMLLoader();
        ldr.setLocation(getClass().getResource("enter.fxml"));
        Parent root = (Parent)ldr.load();
        Scene scene = new Scene(root);
        myStage.setScene(scene);
        myStage.show();
       }
```

В контроллере для этой формы объявляем поля:

`@FXML`

`private TextField username;` - текстовое поле для ввода логина

`@FXML`

`private TextField password;` - текстовое поле для ввода пароля

`@FXML`

`private Button in;` - кнопка «Вход»

`@FXML`

`private Label error;` - текстовая метка для вывода сообщений об ошибках

`public static InfoUserAccess userPrivilegies;` - поле для указания привилегий пользователя.

Для @FXML-полей размещаем соответствующие элементы управления на форму в JavaFX Scene Builder (Рис.2).

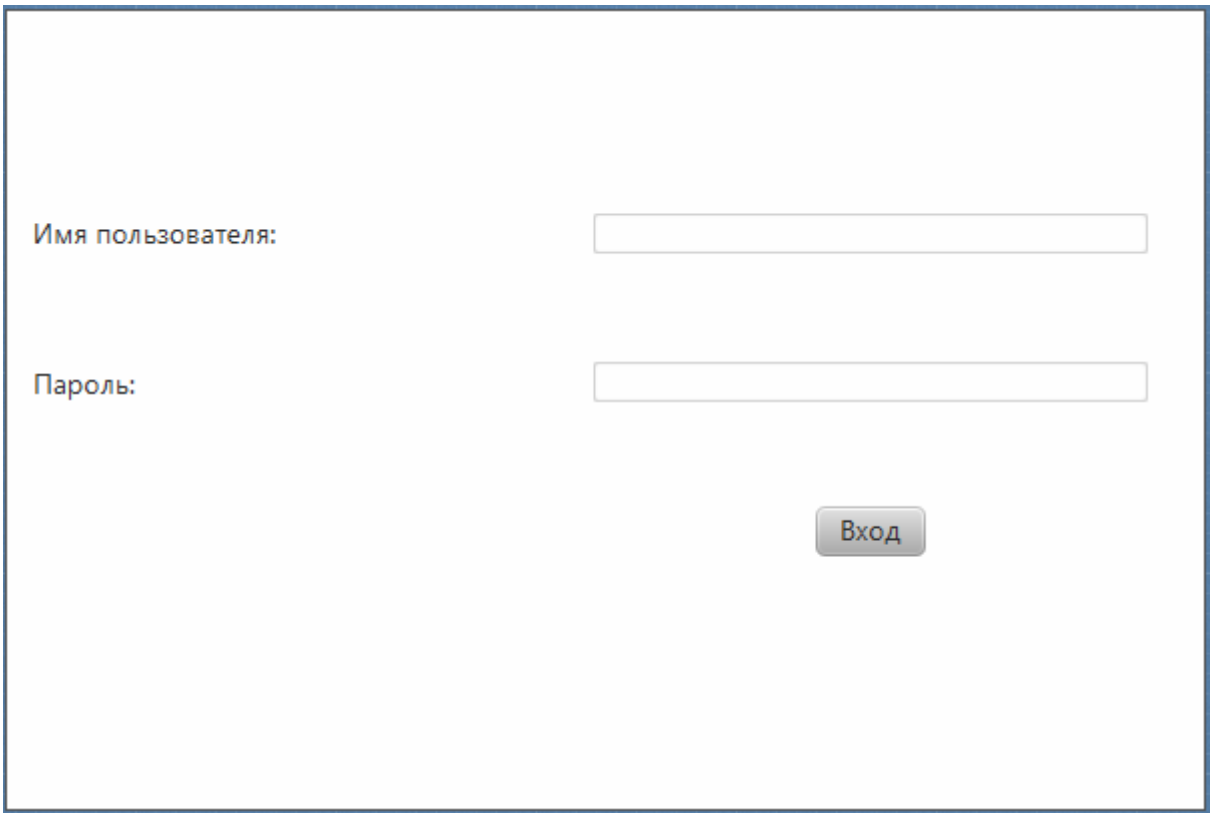

Рис.2 Форма «Вход в систему»

Пишем метод для входа в главное меню openmenu(ActionEvent event). запрашиваем объект, который отвечает за авторизацию:

```
RemoteAccountDispatcher dispatcher;
dis-
patcher=(RemoteAccountDispatcher)(connection.requestRemoteServic
e ("SHAccountDispatcher"));
```

Берем значения текстовых полей и вызываем dispatcher.logIn(username.getText(), password.getText()); Этот метод возвращает объект управления сервером controller, который реализует интерфейс RemoteCoreController.

```
controller = (RemoteCoreController) dis-
patcher.logIn(username.getText(), password.getText());
```

В зависимости от введенных логина и пароля определяются права доступа пользователя:

```
userPrivilegies = controller.getUserPermissions();
```

Открывается форма «Главное меню»:

```
SmartHouseClient.myStage.setScene(new Scene((Parent)
FXMLLoader.load(getClass().getResource("menu.fxml"))));
SmartHouseClient.myStage.show();
```

На этом этапе может произойти ряд ошибок, например, введен неправильный логин или пароль, либо не удалось подключиться к базе данных, либо не удается установить соединение с сервером. Тогда dispatcher выбросит соответствующие исключения, и в метку error будут вписаны сообщения об ошибках:

```
try {
            RemoteAccountDispatcher dispatcher;//запросить объект, который отвечает за авторизацию
            dispatcher = (RemoteAccountDispatcher) (connection.requestRemoteService("SHAccountDispatcher"));
            controller = (RemoteCoreController) dispatcher.logIn(username.getText(), password.getText());
            userPrivilegies = controller.getUserPermissions();
            SmartHouseClient.myStage.setScene(new Scene((Parent)
FXMLLoader.load(getClass().getResource("menu.fxml"))));
            SmartHouseClient.myStage.show();
        } catch (BadPasswordException ex) {
            error.setVisible(true);
            error.setText("Пароль неверный");
        } catch (BadUsernameException ex) {
            error.setVisible(true);
            error.setText("Логин неверный");
        } catch (OperationFailedException ex) {
            error.setVisible(true);
            error.setText("Невозможно подключиться к базе данных");

        } catch (RemoteException ex) {
            error.setVisible(true);
            error.setText("Проблемы соединения с сервером");
        }
```

На вторую форму добавляем текстовые метки и в контроллере пишем для них процедуры открытия форм, в Scene Builder указываем, что процедуры срабатывают при нажатии мышью на метку. Также добавляем кнопку для возврата на форму ввода логина и пароля. (Рис.3)

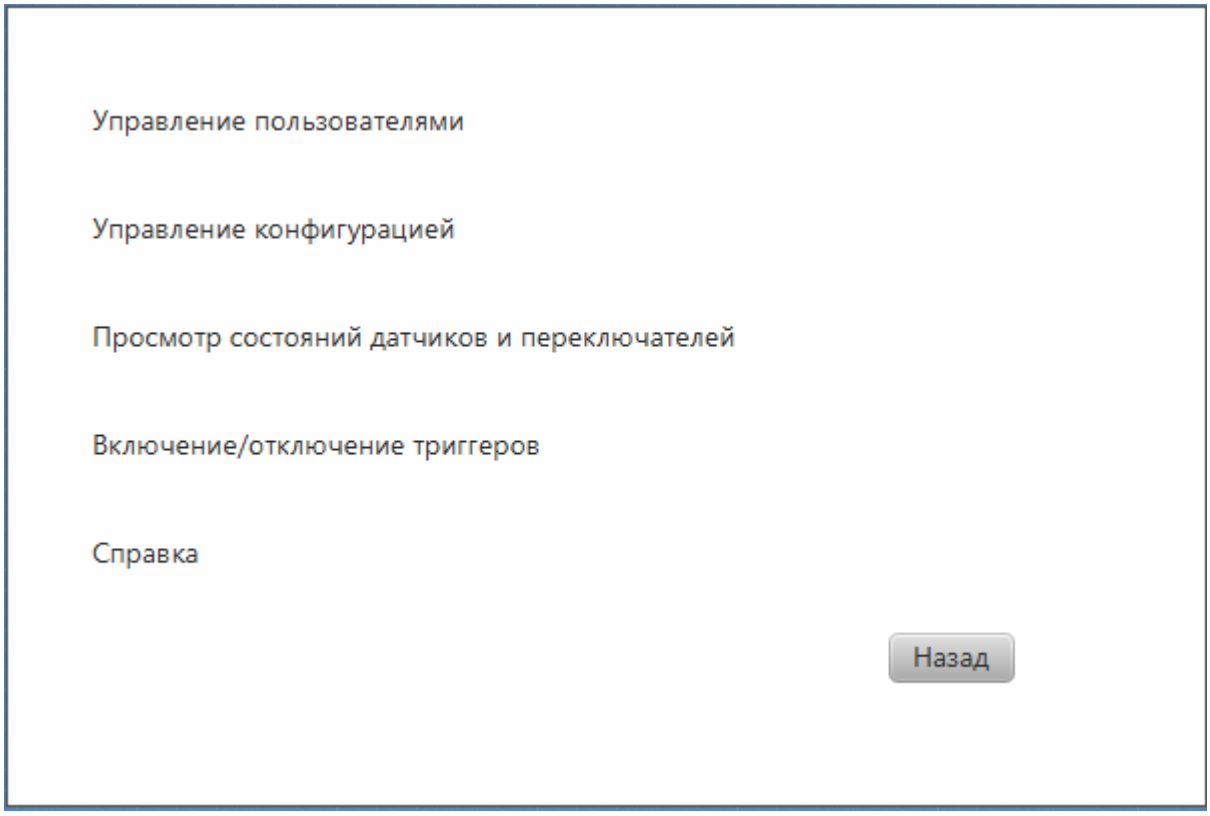

Рис.3 Форма «Главное меню»

Если у пользователя недостаточно прав для управления пользователями, конфигурацией и включения/отключения триггеров, то эти метки будут для него недоступны:

```
  @Override
    public void initialize(URL url, ResourceBundle rb) {
        if (userPrivilegies.addUser = false) {
            user_control.setDisable(true);
        }
        if (userPrivilegies.addStructure == false || userPrivi-
legies.addElement == false) {
            config_control.setDisable(true);
        }
        if (userPrivilegies.activateTrigger == false) {
            on_off_trigger.setDisable(true);
```

```
    }
}
```

Пишем контроллер для формы «Добавление нового пользователя». Объявляем поля:

`@FXML`

`private TextField username;` - логин

`@FXML`

`private TextField password;` - пароль

`@FXML`

`private ChoiceBox privilege;` - выпадающий список с уровнями доступа пользователей

`@FXML`

`private Button add;` - кнопка для добавления пользователя

`@FXML`

`private Button backtomenu;` - кнопка для возврата на предыдущую форму

`@FXML`

`private Label error;` - метка для вывода сообщения об ошибке.

Добавим все элементы на форму (Рис.4).

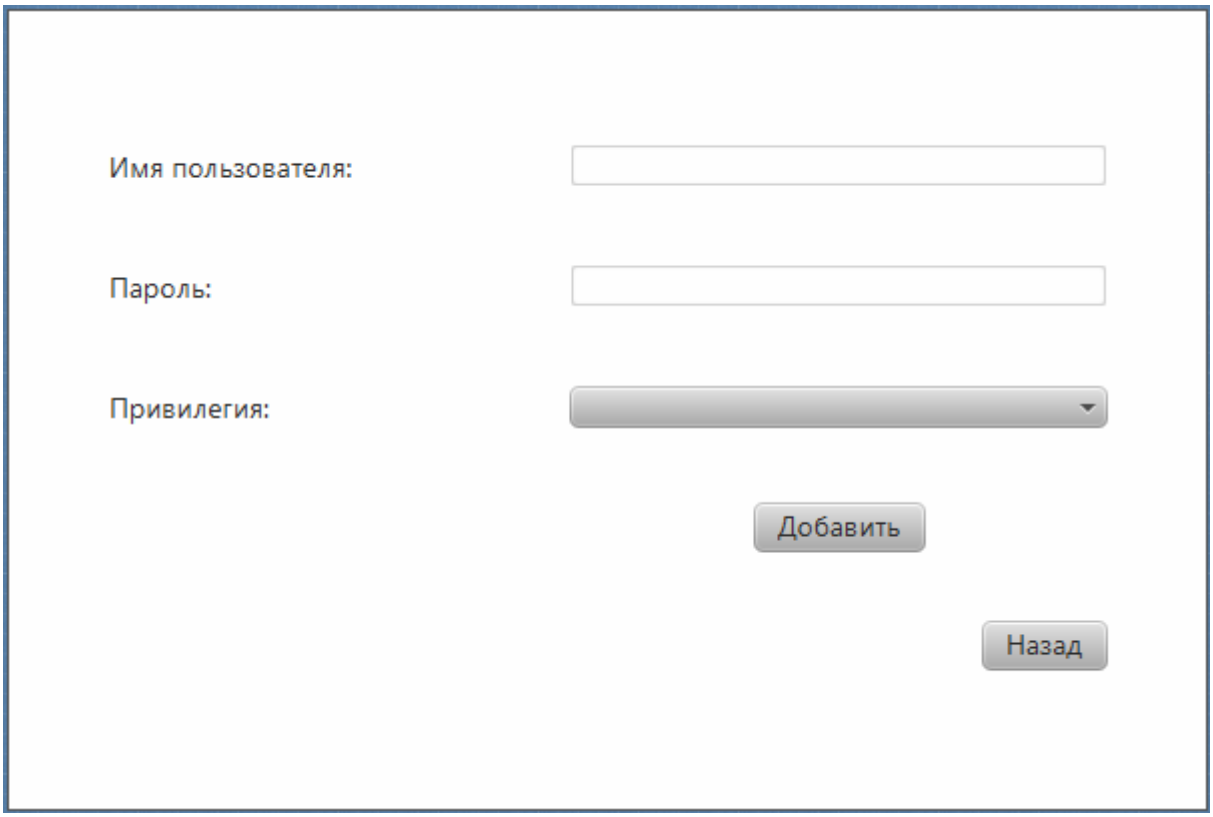

Рис.4 Форма «Добавление пользователя»

Заполняем ChoiceBox привилегиями:

```
@Override
public void initialize (URL url, ResourceBundle rb) {
privilege.getItems().clear();
privilege.getItems().add( new KeyValue<Integer>("Администратор",
9999) );
privilege.getItems().add( new KeyValue<Integer>("Простой пользо-
ватель", 4999) );
privilege.getItems().add( new KeyValue<Integer>("Только
просмотр", 0) );
    }
```

Класс KeyValue<T extends Object> для какого-либо объекта типа T задает его строковое представление:

```
public class KeyValue<T extends Object> {
    public T value;
    public String key;
    public KeyValue(String nKey, T nValue) {
        key = nKey;
        value = nValue;
```

```
    }
    @Override
    public String toString() {
        return this.key;
    }
}
```

Пишем контроллер для удаления и редактирования пользователей. Объявляем поля:

`@FXML`

`private ChoiceBox username;` - выпадающий список с логинами

`@FXML`

`private TextField password;` - текстовое поле для пароля

`@FXML`

`private TextField newpassword;` - текстовое поле для нового пароля

`@FXML`

`private Button delete;` - кнопка для удаления пользователя

`@FXML`

`private Button backtomenu;` - кнопка для возвращения на предыдущую форму

`@FXML`

`private Button rewrite;` - кнопка для редактирования

`private String login;` - логин

`@FXML`

`private Label error;` - метка для сообщения об ошибке

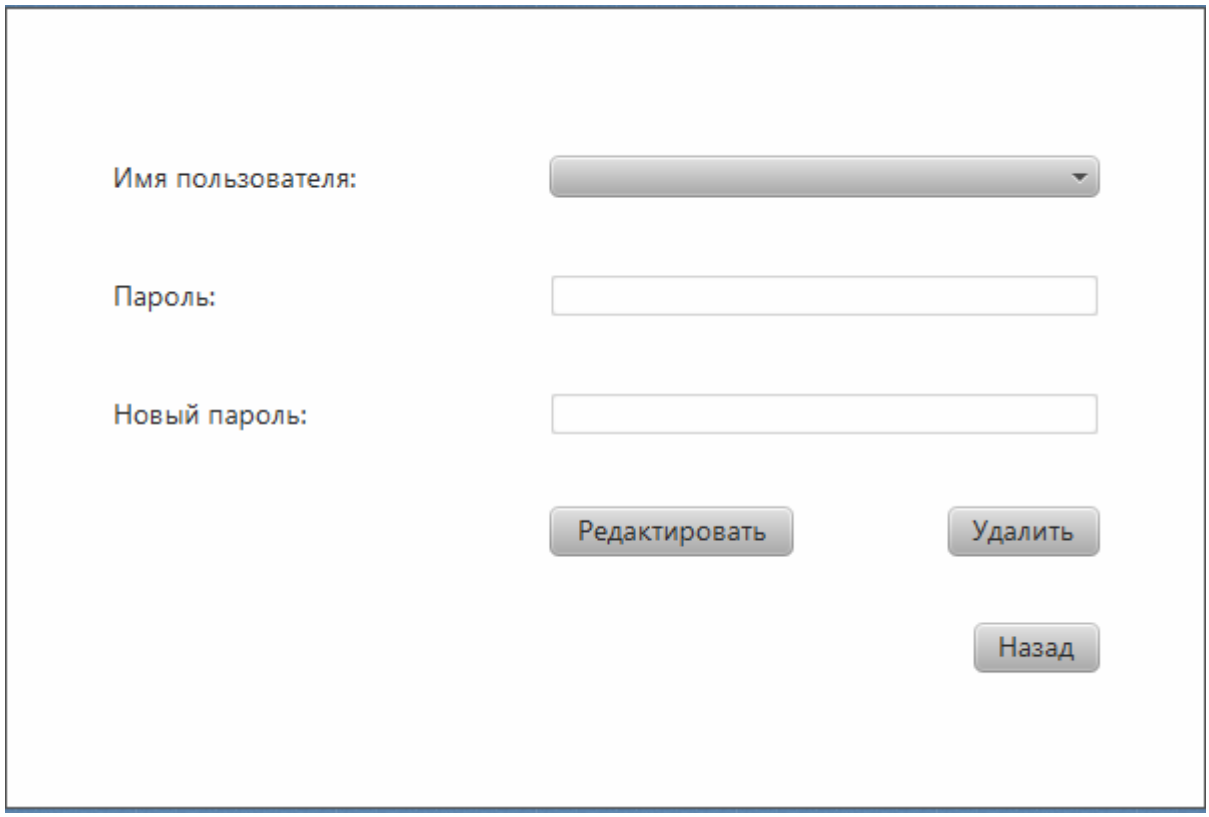

Рис.5 Форма «Редактирование и удаление пользователей»

Процедура удаления пользователя:

```
public void deleteUser() {
try {
controller.deleteUser(                                    ((Key-
Value<InfoUser>)(username.getValue())).value.login );
   } catch (NotAllowedException ex) {
            error.setVisible(true);
            error.setText("Недостаточно прав для удаления поль-
зователей");
        } catch (OperationFailedException ex) {
            error.setVisible(true);
            error.setText("Невозможно подключиться к базе дан-
ных");

        } catch (RemoteException ex) {
            error.setVisible(true);
            error.setText("Проблемы соединения с сервером");

        }
```

```
    }
```

Процедура для редактирования пользователей:

```
public void editUser() {
InfoUser eUser = (KeyValue<InfoUser>)username.getValue()).value;
        if (password.getText().equals(newpassword.getText())) {
            eUser.password = password.getText();
            try {
                controller.editUser(eUser);
            } catch (NotAllowedException ex) {
                error.setVisible(true);
                error.setText("Недостаточно прав для редактирования пользователей");
            } catch (OperationFailedException ex) {
                error.setVisible(true);
                error.setText("Невозможно подключиться к базе данных");

            } catch (RemoteException ex) {
                error.setVisible(true);
                error.setText("Проблемы соединения с сервером");
            }
        }
    }
```

Контроллер сервера запрашивает список пользователей и заполняет им выпадающий список username.

```
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        try {
            LinkedList<InfoUser> userList = controller.getUserList();
            username.getItems().clear();
            Iterator<InfoUser> it = userList.iterator();
            while (it.hasNext()) {
                InfoUser cUser = it.next();
```

```
                username.getItems().add(new                    Key-
Value<InfoUser>(cUser.login, cUser));

            }
        } catch (NotAllowedException ex) {
            error.setVisible(true);
            error.setText("Недостаточно прав");
        } catch (OperationFailedException ex) {
            error.setVisible(true);
            error.setText("Невозможно подключиться к базе дан-
ных");

        } catch (RemoteException ex) {
            error.setVisible(true);
            error.setText("Проблемы соединения с сервером");

         }
```

Создадим форму для просмотра датчиков в комнате. Объявим поля:

`@FXML`

`private ChoiceBox rooms;` - выпадающий список с комнатами

`@FXML`

`private ScrollPane target;` - полоса прокрутки

`@FXML`

`private Label error;` - метка для вывода сообщения об ошибке

Добавим компоненты на форму «Просмотр состояний датчиков в комнате» (Рис.6)

Рис.6 Форма «Просмотр состояний датчиков в комнате»

В initialize(URL url, ResourceBundle rb) заполняем ChoiceBox комнатами. На этот ChoiceBox добавляем слушатель, отслеживающий состояние свойства selectedItemProperty. При изменении этого свойства мы получаем с сервера список переключателей, соответствующих выбранной комнате, и для каждого из них добавляем на форму управляющий элемент «шкала». К каждому элементу шкалы привязываем слушателя, который отслеживает свойство activeProperty элемента шкалы. Свойство activeProperty используется для управления состоянием датчиков. При его изменении связанный с ним слушатель инициирует процессы активации и деактивации датчиков, отправляя соответствующие команды серверу. Кроме этого, для каждого элемента шкалы добавляется объект, обновляющий её показания. Этот объект представляет собой побочный поток, периодически запрашивающий у сервера показания датчика.

Остальные формы с добавлением, удалением, редактированием и просмотром датчиков, переключателей, триггеров, комнат и систем делаем по тому же принципу.