Обмен с GUI

Существует два способа взаимодействия с окружающей системой: либо путем выборки непосредственно из задачи GUI, либо путем выборки из вторичной задачи.

Я выбрал вариант создания вторичной задачи для обмена данными с задачей GUI. У нас создана задача ReadData, которая считывает данные с датчиков и заполняет массив хранения данных. Эта же задача будет обмениваться данными с задачей GUI.

Обмен данными между задачами будет производиться с помощью системы обмена сообщениями между задачами, встроенной в RTOS. Функции RTOS для обмена сообщениями:

osEventFlagsSet – установка сообщения для задачи

osEventFlagsWait – ожидание сообщения от задачи

Как только появится необходимость изменить значение показателя на экране, задача…

# Класс Model

Класс Model не только выполняет функции интерфейса к нашей окружающей системе, но и отвечает за хранение данных о состоянии, поэтому могут существовать некоторые переменные состояния, которые также нуждаются в обновлении. К нашей системе подключены пять датчиков температуры, и текущая температура каждого должна отображаться в пользовательском интерфейсе. При подготовке мы расширим класс Model для поддержки этого в заголовочном файле Model.hpp добавляем пять защищённых переменных float:

#ifndef MODEL\_HPP

#define MODEL\_HPP

class ModelListener;

class Model

{

public:

    Model();

    void bind(ModelListener\* listener)

    {

        modelListener = listener;

    }

    void tick();

protected:

    ModelListener\* modelListener;

    float currentTemperatureLeft;

    float currentTemperatureRight;

    float currentTemperatureCenter;

    float ProductTemperatureLeft;

    float ProductTemperatureRight;

};

#endif // MODEL\_HPP

Благодаря вышесказанному наш Presenters (ведущий) может запрашивать модель о текущей температуре, позволяя ведущему устанавливать это значение в пользовательском интерфейсе (View) при переключении на экран, отображающий температуру. Что нам нужно сделать сейчас, так это иметь возможность снова обновить пользовательский интерфейс при получении новой информации о температуре. Для этого мы используем тот факт, что Модель имеет указатель на наш текущий активный презентатор. Тип этого указателя — это интерфейс (ModelListener), который мы можем изменить, чтобы отразить соответствующие события, относящиеся к конкретному приложению. Сообщать презентеру об изменении температуры будет функция notifyTemperatureLeftChanged:

#ifndef MODELLISTENER\_HPP

#define MODELLISTENER\_HPP

#include <gui/model/Model.hpp>

class ModelListener

{

public:

    ModelListener() : model(0) {}

    virtual ~ModelListener() {}

    void bind(Model\* m)

    {

        model = m;

    }

    virtual void notifyTemperatureLeftChanged(float currentTemperatureLeft) {}

protected:

    Model\* model;

};

#endif // MODELLISTENER\_HPP

Поскольку notifyTemperatureChanged функция имеет пустую реализацию по умолчанию, это уведомление просто игнорируется презентерами других экранов. С другой стороны, температуры отображаются на экране Visualization, и я могу в этом презентаторе переопределить notifyTemperatureChanged функцию и сообщить представлению, что оно должно отображать обновленную температуру: