Network Embedded Systems [VU]
Semster Project - Protocol

Vienna University of Technology
January 22, 2017

Helmut Bergmann, Matr.Nr. 0325535

0325535@student.tuwien.ac.at

Bernhard Fritz, Matr.Nr. 0828317

0828317@student.tuwien.ac.at

Marko Stanisic, Matr.Nr. 0325230

0325230@student.tuwien.ac.at

1 Requirements

The requirements come here..

It should water the flowers...

- Reliable flower watering
- It should be quiet
- An efficient solution
- The hardware solution should be visually appealing
- Data should be logged to perform statistical analysis on it.
- The data connection should be wirelessly

2 Design

The network infrastructure consist of a IOT controller, a data logger, sensor nodes and pump nodes.

2.1 Sensor Node

The sensor node provides the current state of the flower environment to the controller. It is a low power device which guarantees long battery duration.

The sensors used are:

- Temperature sensor It shows the temperature...
- The second sensor is a moisture sensor. It consists of two parts, a resistance probe (YL-69) and a control board (YL-38). The probe will be put into the measured soil, while the board should stay dry. The Arduino nano has a 10 bit ADC, so the result read from the ADC is an integer between 0 and 1023.

  The test showed that in a dry state the sensor shows 1023. Connecting the electrodes with a finger leads to a value around 1000.
  Also in dry soil the readout is 1023, but as we pour water into the pot, it continuously decreases down to a few hundred. (I did not want to drown the flower, but I

assume it goes to 0 if we put the probe into water...)

We might have to calibrate the moisture levels for different flower or soil types, but in general the sensor seems to do the job very well and also is usable with a XBee board because of its analog output. The only downside is that it's written in forums that after some weeks this type of sensor will be corroded, so we cleaned it as well as possible after the test. For developing and testing it is fine, in a real environment one might want to use another sensor like a capacitive one or with other electrode materials. Also turning off the sensor when not using it is a good approach to avoid this problem.

- The last sensor to test was a photoresistor which is a semiconductor that reacts to light. It was also provided in the Arduino kit. The manual suggested the range of the photoresistor is between 500Ω (bright) and 50$k\Omega$ (dark). Further it suggested to use a $1k\Omega$ resistor for the voltage divider. An arduino tutorial [**?**] suggested 10 $k\Omega$. So we decided to measure the actual resistor values and use the formula from the lecture to calculate the matching resistor of the voltage divider.

  The measurements showed that the actual photoresistor range is 500Ω to $1.7M\Omega$.

$$R_1 = \sqrt{R_{max} * R_{max}} = \sqrt{500 * 1700000} = \sqrt{850000000} = 29k\Omega \qquad (1)$$

  The closest resistor available to us was $26k\Omega$.
  These values differ to those suggested by the tutorials, but since also the photoresistor has different values, our choice seems feasible. Also we are convinced a higher resistor will not harm the components in contrast to a low resistor.
  So we connected the chosen pulldown resistor between ground and the A1 pin of the Arduino Nano. To complete the voltage divider the photoresistor was connected between VCC and the A1 pin.

  Sketches again are provided by online tutorials and the Arduino kit producer. Although it is very similar to the one for the moisture sensor, just another pin is used.

  The test worked very well, close to a light bulb we get values over 950 and if we cover the photoresistor it drops to under 30. That confirmed our choice of the resistor.

  We can conclude that this sensor is easy to work with and compatible with XBee, as no special protocols are required.

2.2 Controller

The controller keeps track of all nodes and handles their registration. It executes the watering policy that the user has selected for each flower. For this task it wirelessly communicates with the sensor node situated in the flower pot and the pump node situated next to the water tank. The sensor node provides the controller with the most recent measurement values, whereas the pump node executes the pump commands received from the controller.

2.2.1 State diagram

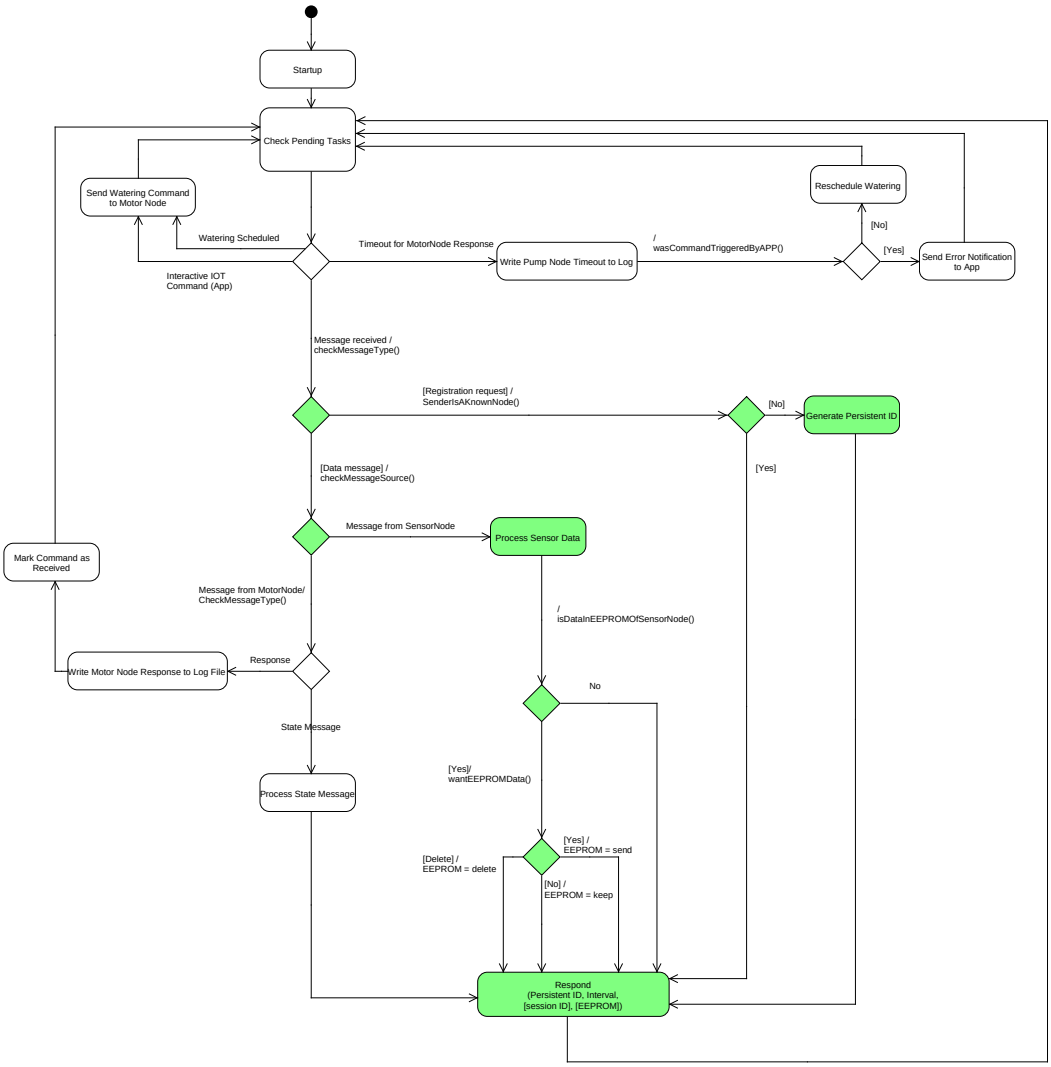

Figure 1: Overview of the chosen setup.

States:

- Check pending tasks

  This is the begin of the loop() Function. The controller will always come back to this state to check which tasks are pending. The following actions are possible:

  - A NRF24 message is arrived, process this message.
  - Also if an interactive command has arrived via the IOT framework, this command has to be processed. (Typically sending an instruction to a Motor Node)
  - If, according to the watering schedule, a pump node should take an action now, the controller goes to the "Send Watering Command to Motor Node" state
  - If a Motor Node did not respond as expected, an error handling sequence should be started (writing to log and if needed respond to the IOT framework)

- Send Watering Command to Motor Node Whenever the controller decides a pump node should start to pump, this function is called. It prepares and sends a message to concerned the pump node. It also writes the message content to a log file.

- Write Pump Node Timeout to Log If a pump node does not reply within the specified timeout duration, an entry into the log file has to be written. In case the command of the concerned message has been triggered by the IOT framework, the controller will move to the "Send Error Notification to App" state. Otherwise it enters the "Reschedule Watering" state.

- Send Error Notification to App If a IOT command could not be successfully executed, the IOT framework has to be notified about the problem. Afterwards the controller goes back to checking for pending tasks.

- Reschedule Watering If a scheduled watering task failed, the controller needs to reschedule this watering task. There are many possible algorithms for this such as:

  - Skip this watering task and proceed as usual
  - Retry the watering task a certain number of times. If it fails, proceed as usual
  - Reschedule the watering of this pump node in a given fashion.

  For a first version it is sufficient to skip the watering, if it failed.

- Write Motor Node Response to Log File If a motor node responds to a control command, this response has to be logged to a log file. Afterwards the controller processes the response.

- Mark Command as Read The watering command is marked in the controller internal command list as "read by the node". Otherwise a timeout would occur.

- Process State Message As the pump node finished to pump, it will send a state message to the controller indicating the end of the pump sequence. This message can be responded to by the controller, but it doesn't have to be.
- Process Sensor Data When the controller receives data from a sensor node, it stores the data. Later this data will be used for the watering algorithm and statistical analysis. Afterwards it will check whether the sensor node has data stored in its EEPROM. This will be the case if the controller was unavailable to the sensor node for some time. If there is EEPROM data the controller can choose whether it wants the data to be transmitted now, later or not at all. If the latter is the case, the node will delete its sensor data. If the controller has currently time to receive the EEPROM data, it should request it. Otherwise the data might get lost. If there is a lot of traffic scheduled in the next slots, the controller can postpone the EEPROM data transmission. In this case the next sensor node transmission should be scheduled in a slot that is followed by empty slots. This way it is guaranteed that there is enough time to transmit all EEPROM data.
- Respond In the respond state a response is prepared and sent to the node from which the last message has been received. Its content is based on the previous states.

2.3 Protocol

For registering nodes and perform data exchanges a protocol has been developed. The developed library provides access to the protocol registers to use the functionality of the protocol.

Status register in the node –> controller message:

Bit	name	description	0 - meaning	1 - meaning
0	RTC_RUNNING_BIT	RTC is paired on the node	no RTC	RTC paired
1	MSG_TYPE_BIT	type of message	registration request	data
2	NEW_NODE_BIT	new or known node	known node	new node
3	EEPROM_DATA_AVAILABLE	is data in the EEPROM	no data	data available
4	EEPROM_DATA_PACKED	kind of data	live data	EEPROM data
5	EEPROM_DATA_LAST	more EEPROM data pending	more data	not more data
6	NODE_TYPE	type of node	sensor node	pump node

Status register in the controller –> node message:

Further the combination "00" in the FETCH_EEPROM_DATA1/2 bits indicates the EEPROM data shouldn't be transmitted now, "01" means the sensor node should send the data now and "10" means the node should delete the stored EEPROM data.

Bit	name	description	0 - meaning	1 - meaning
0	REGISTER_ACK_BIT	registration acknowledge	no reg	registration
1	FETCH_EEPROM_DATA1			
2	FETCH_EEPROM_DATA2			
3	ID_INEXISTENT	id invalid	id valid	id invalid