Automatically Quantitative Analysis and Code Generator for Sensor Systems: The Example of Great Lakes Water Quality Monitoring

Bojan Nokovic(✉) and Emil Sekerinski

Computing and Software Department, McMaster University, Hamilton, Canada
{nokovib,emil}@mcmaster.ca

Abstract. In model-driven development of embedded systems, one would ideally automate both the code generation from the model and the analysis of the model for functional correctness, liveness, timing guarantees, and quantitative properties. Characteristically for embedded systems, analyzing quantitative properties like resource consumption and performance requires a model of the environment as well. We use *pState* to analyze the power consumption of motes intended for water quality monitoring of recreational beaches in Lake Ontario. We show how system properties can be analyzed by model checking rather than by classical approach based on a functional breakdown and spreadsheet calculation. From the same model, it is possible to generate a framework of executable code to be run on the sensor's microcontroller. The goal of model checking approach is an improvement of engineering efficiency.

Keywords: Water quality monitoring · Probabilistic model checking · Validation · Verification

1 Introduction

In this work we build a model for and analyze the power consumption of water monitoring motes developed in the MacWater [1] project. The sensors are intended for water quality monitoring of beaches on Lake Ontario, to supplement and speed up the existing practice of manually taking water samples and analyzing them in a lab. For battery-powered motes, power consumption has the main impact on product usability. A shorter battery life requires more frequent battery replacements. As the motes are deployed in buoys (placed on a specific distance from the shore according to local regulations for testing water quality of beaches), there is a significant effort in battery replacements or any kind of maintenance.

There are two ways of power consumption evaluation (1) on the physical hardware, by periodically measuring the remaining battery, or (2) by modelling. In both cases, the interaction with the environment determines the power consumption. While in the measuring approach inputs are real, in the modelling approach they are simulated or *synthetic*. Modelling can be less accurate than

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2016

measuring, but it can give designers flexibility and agility to evaluate complex power consumption scenarios [2]. The classical approach to power model design is based on a *functional breakdown*. First, power consumption is calculated following a design process similar to the one described in [3,4]. Next, all activities that are possible sources of power consumption or *logical activities* [5] are identified. Finally power consumption is calculated manually by standard mathematical operations or with the help of some tools, e.g. spreadsheet.

In our approach the system is first described by pCharts, a visual language for specifying reactive behaviour. Then, after specifying power consumption in relevant states, input code for a probabilistic model checker is automatically created and power consumption calculated as a *cost* over probabilistic computational tree logic (PCTL) formulae. On the example of Waspmotes, commercial Arduino-based motes by Libelium, we present the interaction between the environment and a device as a complex probabilistic timed automaton (PTA), on which it is still feasible to perform quantitative analysis by an off-the-shelf probabilistic model checker. In addition to the calculation of power consumption, we generate the framework of executable code to be run on a microcontroller. We model complex embedded systems, but the code is executed on 8-bit microcontrollers with restricted resources.

2 Related Work

For power consumption evaluation by analytical modelling there are two approaches: the evaluation of whole wireless sensor networks (WSN) [6,7], or the evaluation of WSN applications [2]. The difference is that the evaluation of whole wireless sensor networks includes an in-depth evaluation of communication protocols. This work is about evaluation of WSN application. It is already shown that models created by Coloured Petri Nets (CPNs) can be used to estimate power consumption of sensors [2]. We follow a similar approach but use pCharts, extended hierarchical state machines. The main difference in our approach is in the fact that power consumption on pCharts models is calculated by probabilistic model checker. The random nature of environments impact implies a need for a probabilistic evaluation of different power consumption scenarios. In addition to this, from pCharts it is possible to generate framework of mote's executable code.

The experimental validation of probabilistic systems needs a bigger number of tests to acquire credible results [8]. That may be a time-consuming task. A short-cut to this problem is modelling together the device and environment impact.

In our previous works we introduced the basic features of *pState* [9] and described timed transitions [10]. In [11] we explained how the tool is designed, and we show how a communication protocol for radio-frequency identification (RFID) tags can be analyzed. In [12] we show on few simple examples how properties are specified in an intuitive way such that they can be written without knowledge of temporal logic. In this paper we are focused on the methodological aspect and show that systems with tens of thousands states can be effectively analyzed.

3 Waspmote Sensor Power Consumption

We show how pCharts can be used to model the power consumption of the *end-unit* devices, and how to generate the framework for device-executable code. In a collaborative research effort MacWater [1], new sensor types for water quality indicators are developed. MacWater isa research project for mobile sensor devices that can analyze water samples for biological and chemical contaminants in real-time. The sensors and communication boards are connected to Waspmote, a type of Arduino board [13]. Collected data, includes water *pH* factor, temperature, and a current location of the sensor (longitude and latitude), is transmitted to a nearby base station by a low power wireless protocol. The base station is a multi-protocol router Meshlium [13] that sends data to the central server either by WiFi or GPRS. Collected data are updated on the site, and all system operates as soft real-time process.

For the purpose of this paper, we use commercially available sensors to measure pH of lake water, to read the geographic position of the sensor by GPS, and to transmit data via the ZigBee protocol. In our experiment we also use sensors to measure water conductivity, dissolved oxygen, and dissolved ions, which we leave out here for brevity. We show how to specify the impact of the environment on the working device, and how to quantitatively verify that impact. The model in Fig. 1 has three concurrent states *Device*, *Environmnet* and *Test*. The state *Device* has itself four concurrent composite states *Board*, *pH*, *ZigBee*, and *GPS*. The state *Device* represents behaviour of the Waspmote [13] water monitoring mote. State *Environment* represents the impact of the environment on GPS communication. We add state *Test* to specify queries to be quantitatively verified by the model checker.

Initially, the state *Board* is in *DeepSleep*, state *pH* is in *pHOff*, state *ZigBee* in *ZigBeeOff*, and state *GPS* in the *GPSOff*. Every 10 s, *Board* wakes up, and broadcasts the event *pHOn*. On this event *pH* state goes from *pHOff* to *pHSensorOn* and executes the command *pHTurnOn*. This command is a separately written external function. For model checking, it is ignored, but it is used for executable code generation.

In the state *pHSensorOn*, *pH* stays only 5 ms, to measure water acidity, and then goes back to *pHOff* state. During this process it broadcasts *pHRead* event and call *TurnpHOff* command. On the event *pHRead*, *Board* goes from *pHWarmUp* to *GpsWarmUp*, and broadcasts event *GpsOn*.

On the event *GpsOn*, state *GPS* goes from initial state *GpsOff* to *GpsCheck* and broadcasts event *Connect*. On this event, *Environment* moves form *GpsEnvIdle* to *InitialDelay*. The GPS is used to read a position of the device. In normal operation, based on our measurements, is takes between 1.8 s and 2.2 s for GPS to get connected. No connection is possible in less than 1.8 s, in 50 % of the time a connection is establishes between 1.8 s and 2 s, in 60 % of time between

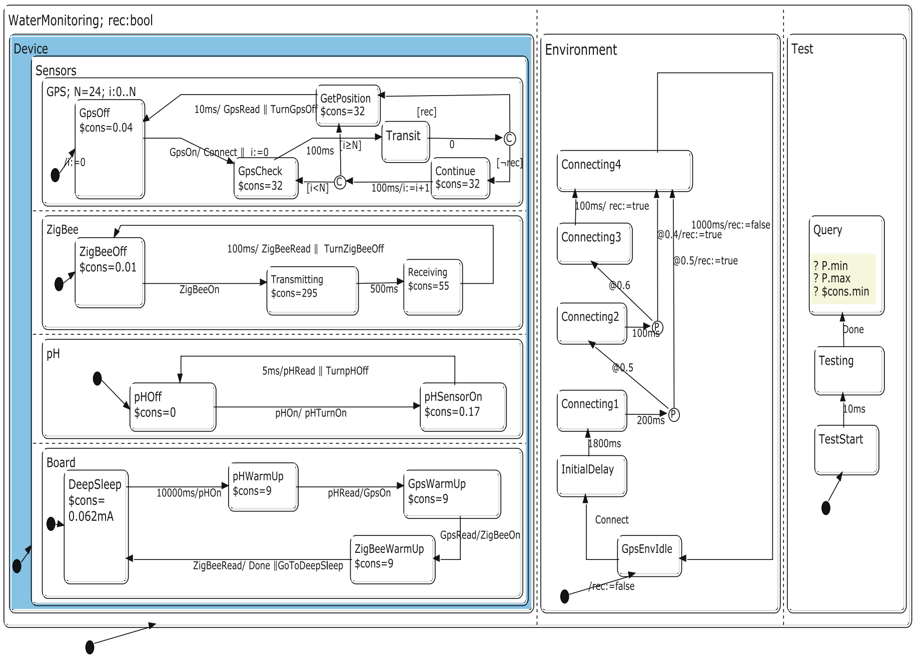

Fig. 1. Wireless sensor power model in pCharts

2 s and 2.1 s. By 2.2 s the connection is always established. This is modelled by probabilistic transitions between *GpsEvnIdle* state and *Connecting4*. When the connection is established, boolean variable *rec* is set to *true*. In our model GPS tries to acquire signal for 4.8 s, or every 200 ms for 24 times. Once the connection is established, GPS goes into *GetPosition* state. Consumption in GPS depends on how fast the connection is established, and that is modelled by probabilistic transitions in the *Environment* state. From state *GetPosition*, GPS goes back into *GPSOff*, broadcasts *GpsRead* event and executes the *TurnGpsOff* command. On broadcasted event *GpsRead*, *Board* goes from *GpsWarmUp* to *ZigBeeWarmUp* and broadcasts event *ZigBeeOn*.

ZigBee, a low-power secure networking protocol, is used to transmit the collected readings to a base station, from where data is further transmitted by a 3G connection to a database. We modelled the power consumption in the transmitting and receiving states, for data transmission and acknowledge reception. Once this process is finished *ZigBee* goes back to *ZigBeeOff*, broadcasts *ZigBeeRead* event and executes command *TurnZigBeeOff*. On the event *ZigBeeRead*, *Board* goes from *ZigBeeWarmUp* to *DeepSleep*, broadcast the event *Done* and executes the command *GoToDeepSleep*. The broadcasted event *Done* moves *Test* from *Testing* to *Query* state, where the queries

$$\text{“}?P.min\text{”} \qquad \text{“}?P.max\text{”} \qquad \text{“}?\$cons.min\text{”} \tag{1}$$

for *min* and *max* probabilities (P), and *min* costs of the consumption (\$*cons*) are specified 1. They are used for the calculation of the probability that the *Board* will go from initial *DeepSleep* back to *DeepSleep* mode, and to calculate the *consumption* in one cycle. Current consumption is specified in *mA* values according to the specification from Waspmote technical documentation [13]. From the pChart in Fig. 1, a PTA model is automatically generated by flattening the hierarchical structure and creating PRISM input code in the form of guarded commands.

The translation scheme of the pCharts model into input code for probabilistic model checker can be found in [10,12].

Rewards. Properties based on *costs* are specified on states or transition. In our example, for each state of *Device*, the cost of consumption *cons* is specified. That is passed to PRISM in the module `rewards ... endrewards`; for instance when the *board* is in *DeepSleep*, the current is only 0.062 mA.

Results. The verification is done by the PRISM *Digital Clock* engine. The created model has 17221 states and 17232 transitions. The calculated minimal and maximal probabilities (*P.min* and *P.max*) to reach *Query* are the same and 1, which means that the test always terminates. There are no nondeterministic transitions, so the *min* and *max* probabilities are equal. The calculated expected minimal consumption (*cons.min*) is 248581.78 mAms, and it took 126.65 s to do calculation. The maximum time of one cycle is a simple sum of deep sleep time (10000 ms) and the times to read pH (5 ms), get position of GPS (2210 ms), and send data by ZigBee (600 ms) which is 12815 ms. So, the average current consumption is 248581.78 mAms/12815 ms = 19.39 mA. Waspmote devices are usually powered by the battery of 6600 mAh, so according to our calculation the battery can last for approximately 340 h, or 14.1 days. Thus we are able to predict automatically the battery life from the model. All properties are verified on Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz, 12.0 GB of RAM and on 64-bit Operating System.

4 Conclusions

This paper presents power consumption evaluation of motes used for water quality monitoring in the MacWater project. We show that hierarchical state machines modelling formalism pCharts is suitable for representing both the device and environment impact. Since input from the environment is of random nature, we use probabilistic transitions for environment specification. Assigned probabilities are based on the experimental evaluation of device to GPS signal connection probability. Analyzing together the device and environmental impact on the same model, allows different scenarios of the environmental impact to be validated. That can be used for optimization of the device's hardware and communication protocol used in a sensor network. Obtaining same measurement by experiment may give more accurate results, but would take more time, which makes modelling as a more convenient approach.

The generated model has about 17000 states and transitions, but the verification performed by PRISM model checker is done reasonably fast. The pState architecture allows in principle other probabilistic model checkers like MRMC [14], or some tool from the MoDeSt [15] toolset to be added. For bigger models, statistical model checkers like Ymer [16] or Vesta [17] can be used.

From the same pCharts model, it is possible to generate code for embedded microprocessors. The goal is to have a seamless and automated approach from modelling and analysis to code generation that can be used by engineers to evaluate design alternatives and to generate trustworthy code.

References

1. McMaster: MacWater, June 2015. http://macwater.org/
2. Damaso, A., Freitas, D., Rosa, N., Silva, B., Maciel, P.: Evaluating the power consumption of wireless sensor network applications using models. Sensors **13**(3), 3473 (2013). http://www.mdpi.com/1424-8220/13/3/3473
3. Negri, L., Sami, M., Tran, Q.D., Zanetti, D.: Flexible power modeling for wireless systems: power modeling and optimization of two bluetooth implementations. In: Cantarella, J. (ed.) Proceedings of 6th IEEE International Symposium on World of Wireless Mobile and Multimedia Networks, pp. 408–416. IEEE Computer Society (2005)
4. Negri, L., Chiarini, A.: Power simulation of communication protocols with StateC. In: Vachoux, A. (ed.) Applications of Specification and Design Languages for SoCs, pp. 277–294. Springer, Berlin (2006)
5. Mura, M., Paolieri, M., Fabbri, F., Negri, L., Sami, M.G.: Power modeling, power analysis for IEEE 802.15.4: a concurrent state machine approach. In: Consumer Communications and Networking Conference, pp. 660–664 (2007)
6. Rusli, M., Harris, R., Punchihewa, A.: Markov chain-based analytical model of opportunistic routing protocol for wireless sensor networks. In: TENCON 2010–2010 IEEE Region 10 Conference, pp. 257–262 (2010)
7. Cano, C., Sfairopoulou, A., Bellalta, B., Barceló, J., Oliver, M.: Analytical model of the LPL with wake up after transmissions MAC protocol for WSNs. In: International Symposium on Wireless Communication Systems (ISWCS 2009), Siena, Italy, September 2009
8. Leopold, M.: Sensor network motes: portability and performance. Ph.D. dissertation, Department of Computer Science, University of Copenhagen (2007)
9. Nokovic, B., Sekerinski, E.: pState: a probabilistic statecharts translator. In: 2013 2nd Mediterranean Conference on Embedded Computing (MECO), pp. 29–32 (2013)
10. Nokovic, B., Sekerinski, E.: Verification and code generation for timed transitions in pCharts. In: Proceedings of the International C* Conference on Computer Sciençeand Software Engineering, Series, C3S2E 2014. ACM, New York (2014)
11. Nokovic, B., Sekerinski, E.: Analysis and implementation of embedded system models: example of tags in item management application. In: W01 1st Workshop on Model-Implementation Fidelity (MiFi), Grenoble, France, p. 10 (2015)
12. Nokovic, B., Sekerinski, E.: A holistic approach to embedded systems development. In: 2nd Workshop on Formal-IDE, Oslo, Norway, p. 14 (2015)
13. Libelium: Waspmote, July 2014. http://www.libelium.com/

14. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
15. Hartmanns, A.: Modest - a unified language for quantitative models. In: 2012 Forum on Specification and Design Languages (FDL), pp. 44–51, September 2012
16. Younes, H.L.S.: Ymer: a statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005)
17. Sen, K., Viswanathan, M., Agha, G.A.: VESTA: a statistical model-checker and analyzer for probabilistic systems. In: QEST, pp. 251–252 (2005)