Poster Abstract: Simulation Framework for Power Aware Wireless Sensors

Johann Glaser and Daniel Weber
Institute of Computer Technology
Vienna University of Technology
Vienna, Austria
Email: {glaser,weber}@ict.tuwien.ac.at

***Abstract*— The PAWiS (Power Aware Wireless Sensors) simulation framework facilitates design and simulation of wireless sensor network models. The main focus is given to power aware computing and therefore it can capture causes resulting in inefficiencies. The simulation framework covers all system aspects comprising of all layers of the communication system, the targeted class of application itself, the power supply and energy management, the central processing unit (CPU) and the sensor-actuator interface.**

I. Introduction

The PAWiS Simulation Framework assists in development and especially modelling, simulation and optimization of Wireless Sensor Networks (WSN) nodes and network protocols. The internal structure of nodes as well as communication between them are simulated. The wide range of applications that can be simulated begins with simple applications like tiny sensor nodes (e.g. the TinyMote [Roe04]), tire pressure monitoring and car climate control to as complex systems as home entertainment (e.g. Sindrion).

The internal structure of a node is built as a *virtual prototype*. This means that its function, the timing behavior and power consumption as well as failures are simulated. With the true top down development methodology, the design starts at a functional specification and implementation level. Guided by requirements, the design is refined via architectural models down to models reflecting the actual implementation. On the other hand, physical constraints affect the functional and architectural designs from bottom up.

II. Work flow

The WSN node is split into *functional blocks* (physical, MAC, routing and application layer, CPU, serial interface, AD converters, timer and so on). Each of these blocks is mapped to modules in the simulation. For every functional block its *module type* is defined which comprises its name and the standardized interfaces. For each of the module types several different implementations can be prepared.

Initially the node is composed from a module implementation of a certain module type out of a module implementations library. Then the modules are configured (e.g. clock frequency of the CPU, resolution of the ADC) to meet the target platform requirements. After that the model is simulated and the results are evaluated and potentially refined for further simulation iterations to improve the desired parameters and the system's behavior.

These cyclic improvements of the models are called *refinement cycles* and are the main track to enhance the development [MGH05]. When these refinement cycles are completed, the final outcome comprises

- the functional description and
- the architecture of the node, as well as
- the implementation details at various levels and
- the power specification of every module including its components.

III. Model Optimization

Several strategies for the optimization of the wireless sensor system are available. First of all a *system level optimization* is performed which includes node composition and modifications of the entire system behavior (e.g. changing the network layout or application pattern).

In parallel *cross-layer optimization* is performed where more than one network layer is modified at a time. Probably each of these changes itself would degrade the node performance, but the interaction of them leads to an improvement in overall behavior of the node.

All optimization is performed by applying the following strategies.

- Exchange the actual module *implementation* for one module type, i.e. a different implementation from the library, e.g. a dual-slope, a $\Sigma\Delta$ or an SAR ADC. Another example is choosing different MAC protocols.
- *Partitioning* of modules and/or functions by dividing the task between hardware and software, digital and analog or RF and baseband. For example a specific MAC protocol could be implemented in software or as dedicated hardware acceleration unit. A combination of both is also possible.
- The *scale* of a module, e.g. the resolution of an ADC or the register count of a CPU.

- *Parameterization* of modules, e.g. the timing and the transmission power of a radio transmitter.

IV. The Simulation Framework

A. Structure

The PAWiS framework is based on the OMNeT++ [Var01] discrete event simulator and the C++ programming language. Figure 1 depicts the structure of the framework from the users view. The model programmer mostly interacts with the framework and C++. Additionally basic knowledge of concepts of OMNeT is required to comprehend the simulation process.

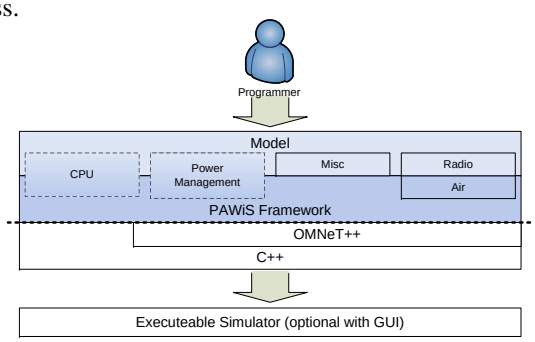

Fig. 1. Structure of the PAWiS simulation framework.

The node composition as well as the network layout are specified in configuration files. Completed models can be compiled (optionally with a GUI based on Tcl/Tk) to an executable simulator. With the optional GUI the workflow and communication of the model can be observed during the simulation. Additionally a log file with power and timing profile is generated for post-simulation analysis.

B. Basic Concepts

a) Modularization: A wireless sensor node is typically split up into various modules, e.g. a CPU, the network protocol layers (application, routing, MAC, physical), a timer and so on. Each of these modules is implemented as a C++ class derived from a framework base class. Decomposition of a sensor node depends strongly on the design requirements. If the main focus of the design is set to network layers it is recommended to provide a module for each layer in order to analyze multiple layer combinations. On the other hand a focus on hardware units would result in modules representing hardware or any applicable combination.

b) State machines: Every module is executing its specific tasks which can be represented as finite state machines (FSM), each one implemented as a class method of the user's module class. Within one module several FSMs can be implemented and even run in parallel. The framework even allows to pass parameters from and to such tasks.

c) Functional Interfaces: Control flow transitions between two modules are implemented as so called *Functional Interfaces*. These are similar to blocking subroutine calls but exceed module boundaries. Additionally modules can define a wait condition depending on other modules or conditions to be satisfied. This is complemented by a mechanism to trigger a parallel task.

d) Environment: All nodes are placed at 3D positions within an environment that manages the outer world of all nodes including their surroundings and the RF channel. Further implementations will include various obstacles and dynamic objects within the environment.

e) Air: The *Air* is a subcomponent of the environment that actually handles the RF channels, which are defined by the node placement and the obstacles between them. The current implementation of the Air handles attenuation effects with free space propagation and isotropic antennas with uniform antenna gain. A radio module can be implemented by deriving from a framework class. Providing a few abstract methods enables the model to get the full support of the Air class. The entire communication including features like bit error rate (BER), timing and SNR aspects is then handled by the framework.

f) Power Simulation: Every module reports its power consumption during simulation. With the framework it is possible to hierarchically combine power sources and adapt their behavior to meet the target platform's requirements. These power profiles are the main output of the simulation and are stored in a log file for further processing.

g) CPU: The submodules of a sensor node are usually either implemented as *firmware* (software), i.e. executed by a CPU, or as *dedicated hardware*. In order to model the power and time consumption of software tasks the PAWiS framework provides a CPU base class to be extended by the framework user. All power simulation issues of the CPU are processed by the framework.

h) Interrupts: The aforementioned CPU features basic interrupt handling methods for multiple *interrupt sources*. These sources, *interrupt vectors* and *interrupt service routines* (ISRs) can be freely configured (by overriding methods) by the user. The timing, dispatching and execution of an ISR is handled by the framework.

The proposed framework is available in a preliminary version and currently being evaluated with a model of a real sensor node [MMR06].

References

[MGH05] Stefan Mahlknecht, Johann Glaser, and Thomas Herndl. PAWIS: Towards a Power Aware System Architecture for a SoC/SiP Wireless Sensor and Actor Node Implementation. In *Proceedings of 6th IFAC International Conference on Fieldbus Systems and their Applications*, pages 129 – 134, Puebla, Mexiko, 14.-15. November 2005.

[MMR06] Stefan Mahlknecht, Sajjad Ahmad Madani, and Matthias Rtzer. Energy Aware Distance Vector Routing Scheme for Data Centric Low Power Wireless Sensor Networks. In *4th International IEEE Conference on Industrial Informatics INDIN’06*, 16-18 August 2006.

[Roe04] Matthias Roetzer. Routing in energieautarken Funksensornetzwerken, 2004.

[Var01] Andras Varga. The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, 2001.