MiXiM, PAWiS, and STEAM-Sim Integration – Combining Channel Models, Energy Awareness, and Real-life Application Code

Georg Möstl* and Andreas Springer‡
*Institute for Integrated Circuits, Johannes Kepler University Linz, georg.moestl@jku.at
‡Institute for Communications and RF-Systems, Johannes Kepler University Linz, a.springer@nthfs.jku.at

***Abstract*—After a decade of research in the field of wireless sensor networks (WSNs) there are still open issues. WSNs impose several severe requirements regarding energy consumption, processing capabilities, mobility, and robustness of wireless transmissions. Simulation has shown to be the most cost-efficient approach for evaluation of WSNs, thus a number of simulators are available. Unfortunately, these simulation environments typically consider WSNs from a special point of view. In this work we present the integration of three such specialized frameworks, namely MiXiM, PAWiS, and STEAM-Sim. This integration combines the strengths of the single frameworks such as realistic channel models, mobility patterns, accurate energy models, and inclusion of real-life application code. The result is a new simulation environment which enables a more general consideration of WSNs. We implemented and verified our proposed concept by means of static and mobile scenarios. As the presented results show, the combined framework gives the same results regarding the functionality and energy consumption as our ”golden model“. Therefore the system integration was successful and the framework is ready to be used by the community[1].**

I. Introduction

A network consisting of tiny, processing ressource limited, and battery powered sensor nodes which communicate wireless is called a wireless sensor network (WSN). The nodes monitor environmental quantities like tension, pressure, sound, temperature, and acceleration. WSNs are used in home automation, car-interior devices, container monitoring and tracking, health-related deployments, and military applications. Some of these applications require highly mobile nodes and therefore mobility comes into play. Further, the lifetime of a sensor network and thus the energy efficiency is a crucial requirement. Typically a node is expected to “live” several years powered by a coin cell. In an increasingly wireless world WSNs are used in industrial applications which imply closed-loop systems. The class of WSN is extended to the class of wireless sensor and actuator networks (WSANs). Therefore the time behavior of software and hardware components becomes essential to guarantee realtime constraints. Another quantity of main interest is the reliability of the network which is dominated by the behavior of the wireless channel.

We conclude that a feasible and accurate simulation of WSNs and WSANs must include mobility of the nodes, sophisticated channel models, fine granular modeled timing, execution of real-life application code, and energy awareness. Since the modeling and simulation of WSNs and WSANs is a complex task and requires an in-depth knowledge of various components, it is best practice to use existing simulation frameworks.

The MiXiM [1] simulation framework focuses on node mobility and accurate channel models. MiXiM itself is a combination of three simulation frameworks. The Mobility Framework provides the general structure, mobility, and connection management for the framework. Radio propagation models are included from the Channel Simulator. The models available in the Positif framework, MAC Simulator, and the Mobility Framework are used as protocol libraries. Another simulation framework is the PAWiS [4] framework which focuses especially on accurate simulation of the energy consumption of a network. With PAWiS it is possible to establish an electrical network consisting of power suppliers and consumers, which influence each other. Further, PAWiS provides a clear separation of software and hardware components and enables the modeling of arbitrary hardware such as a radio transceiver, CPU, or a sensor interface. STEAM-Sim [2] is a recently published simulation environment which enables the simulation of the timing and functionality of real-life firmware, i.e., the software executed by a nodes CPU. The so called *time annotation engine* parses arbitrary firmware code written in the high-level language C and determines the execution time of code blocks. The annotated code is afterwards combined with hardware models developed using PAWiS and simulated.

To get the best of each simulation framework we combine them into a new simulator:

- MiXiM – realistic channel models and mobility
- PAWiS – sophisticated energy models and modeling of arbitrary hardware
- STEAM-Sim – timing and functionality of real-life firmware

The overall simulation environment setup is shown in Fig. 1 which builds up on the discrete event simulator OMNeT++ [3]. As STEAM-Sim already uses hardware models developed in PAWiS, in this work we concentrate more on the discussion of the integration of the MiXiM and the PAWiS framework. As an outcome, it is possible to establish a simulation which

[1]available for download at *http://sourceforge.net/projects/steamsim*

provides accurate timing of hardware and software components, accurate energy consumption behavior, realistic channel models, and mobility of WSNs and WSANs.

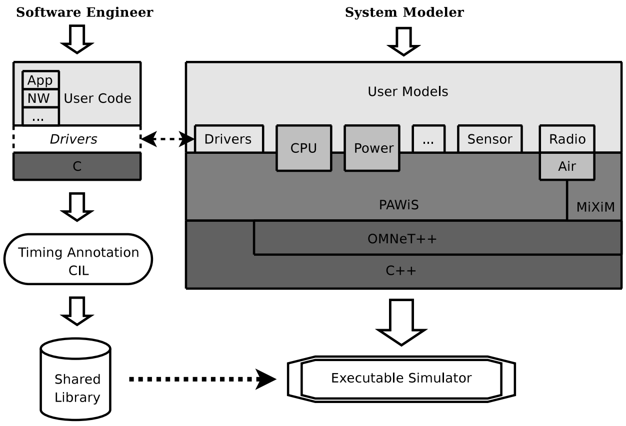


Fig. 1. Developed system integration of MiXiM, PAWiS, and STEAM-Sim

II. IMPLEMENTATION

In this section we discuss the integration of the MiXiM [1] and the PAWiS [4] simulation frameworks. This is a complex task as MiXiM implements a layered view of WSNs complying to the ISO/OSI model, whereas PAWiS provides a hardware/software partitioning of WSNs.

A. *Resolving version conflicts*

The first step was to solve the version conflicts between PAWiS and MiXiM, which prohibits a combined usage of the frameworks. Both frameworks are based on different versions of OMNeT++. PAWiS requires v3.2, whereas MiXiM requires v4.1 or higher. Unfortunately, there are major differences between versions 3.x and 4.x like the internal representation of the simulation time.

To keep up with the development of OMNeT++ we decided to port PAWiS to OMNeT++ v4.1. Although there are some scripts available to automatically port OMNeT++ simulation models from v3.2 to v4.1 we had to do some manual fine-tuning. For example the interfaces of the `sendDirect` and `sendDelayed` methods changed.

B. *Integration of MiXiM and PAWiS*

The integration was accomplished in three steps: (i) identifying how to map the functionality of MiXiM with PAWiS, (ii) determining the interfaces and deriving the inheritance of the developed modules, and (iii) establishing the interconnection in the corresponding NED files.

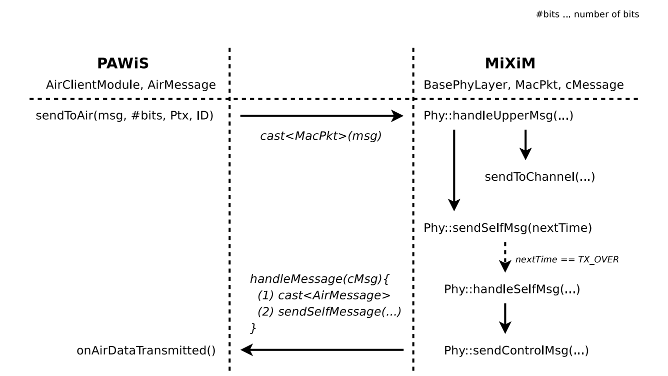
1) Functionality Mapping: Fig. 2(a) depicts the mapping of the functionality between MiXiM and PAWiS for transmissions. As can be seen on the left-hand side in Fig. 2(a), a transmission is originated in PAWiS calling the `sendToAir()` method of an `AirClientModule` object. The `msg` parameter points to a PAWiS `AirMessage` which comprisis data to be transmitted and some control information. The message is encapsulated in a new MiXiM MAC packet (`MacPkt`) and is sent to the MiXiM physical layer (Phy). Therefore the `AirClientModule` is interpreted as the MAC layer from the MiXiM point of view. In the Phy the packet is handled as a MAC packet and (i) is sent to the channel and (ii) results in a scheduled self message marking the end of the transmission. This TX_OVER message is encapsulated into a PAWiS `AirMessage` and is sent to the `AirClientModule` object. When the object receives the message, `onAirDataTransmitted()` is invoked and the transmission is completed at the sender.

As depicted in Fig. 2(b) an incoming packet is first handled in the MiXiM Phy by the `handleAirFrameStartRx()` method. This marks the start of the reception. Subsequently, two tasks have to be accomplished. Firstly, a new self message is scheduled in the Phy, which marks the end of the reception. Secondly, the MiXiM *decider* is invoked to process the new *signal*, thus a new PAWiS `AirMessage` is created and is sent to the PAWiS air client. This message is handled in the `handleMessage()` method. Subsequently, it is evaluated if the transceiver is ready to receive this message via a call to `acceptAirDataStart()`. Additionally, the interfering noise from neighbor channels is calculated executing `calcInterferingNoise()`. If the packet is accepted by the radio transceiver, the PAWiS preview mechanism is invoked (`onPreviewPacket()`) which allows to split the packet reception into interesting parts such as receiving the preamble, the synchronization word, the payload, etc. A change in the SNIR results in a call of the PAWiS `calcBitErrors()` method.

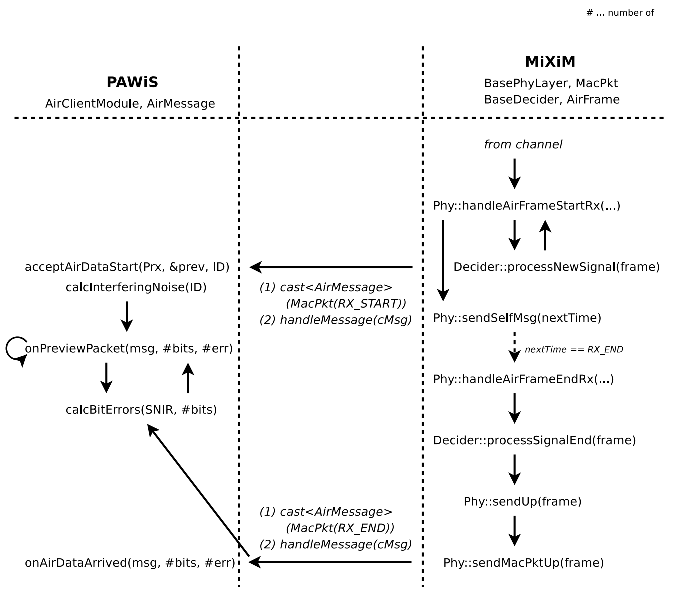
When the last byte of the packet has been received, the corresponding `handleAirFrameEnd()` method is called in the MiXiM Phy, which subsequently calls the `processSignalEnd()` method in the decider. Afterwards the received MAC packet is converted to a PAWiS `AirMessage` and is sent upwards to the PAWiS framework where the signal/noise level is updated accordingly. Finally, `onAirDataArrived()` is invoked in the `AirClientModule` which marks the end of the reception.

2) Structure, Interfaces and Inheritance: Fig. 3 depicts the necessary modules for integrating MiXiM and PAWiS in a simulation run. Modules highlighted in grey are mandatory running a combined simulation.

From the point of view of PAWiS there must be an instance of the `MiximAirClientModule` class which represents a model of a radio transceiver such as a CC2500. This `MiximAirClientModule` is derived from the PAWiS internal `AirClientModule` class. As can be seen in Fig. 3 this external transceiver is connected to a microcontroller via a serial interface (SPI). This interface is used by the firmware running on the microcontroller to configure and control the transceiver. Three methods in the `MiximAirClientModule` had to be developed. Firstly, the `onStartup()` method of every PAWiS module is invoked during initialization of the simulation. In this method OMNeT++ gates (`upperGate` and `upperCtrlGate`) and



(a) Transmit path



(b) Receive path

Fig. 2. MiXiM and PAWiS integration – mapping functionality for the transmit and receive operations

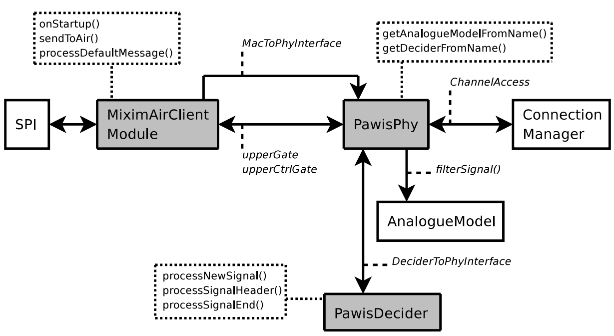


Fig. 3. MiXiM and PAWiS integration – structure and interfaces (modules highlighted in gray are mandatory)

a reference to the MiXiM Phy (`MacToPhyInterface`) are retrieved and locally stored. This ensures efficient handling of further interactions with the MiXiM Phy. Subsequently, the Phy is set to the receiving state. Secondly, the `sendToAir()` method is used to encapsulate a PAWiS `AirMessage` into a MiXiM `MacPkt` (including a MiXiM signal). Thirdly, `processDefaultMessage()` is invoked by the `handleMessage()` method and processes the reception of a MiXiM `MacPkt` sent from the Phy to the PAWiS radio transceiver. Depending on the type of message the corresponding actions are invoked such as an update of the SNIR in PAWiS.

In MiXiM we introduced two modules namely `PawisPhy` and `PawisDecider`. The `PawisPhy` class is derived from the MiXiM `PhyLayer` and implements methods to initialize the used MiXiM *analogue model* and the decider from NED parameters (`getAnalogueModelFromName()` and `getDeciderFromName()`). The `PawisDecider` class which is a subclass of the MiXiM `BaseDecider` is controlled by the Phy using the `DeciderToPhyInterface`. Both, the Phy and the decider have internal references to each other, therefore no OMNeT++ message passing is used. Three methods of the decider were overwritten. First, the `processNewSignal()` method is invoked when a reception starts and implements the cast of a MiXiM `AirFrame` to a message which can be handled by the PAWiS transceiver model. The `PawisDecider` was designed in such a way that it is possible to handle multiple concurrent signal receptions. Second, `processSignalHeader()` is just an empty stub and is left open for further development. Third, the `processSignalEnd()` method implements the indication of the end of reception to the PAWiS `MiximAirClientModule` object.

3) NED Structure: MiXiM defines a fixed structure of the network setup and the basic nodes. This structure is defined in OMNeT++ NED files. At the top level of a MiXiM simulation model there is the `BaseNetwork` which, for example, defines the used connection manager. To integrate PAWiS into the simulation the `BaseNetwork` instantiates two PAWiS models namely the `PawisLogRS232` for logging a nodes serial output and the `Config` module needed for configuration.

A node in MiXiM must be derived from type `BaseNode` where a mandatory mobility module, an ARP module, and a utility module (e.g., blackboard) are instantiated. To integrate PAWiS, there is only one change necessary, i.e., the definition of a unique node identifier `MiximNodeId`. As a result of the integration process, within a node there is only the `nic` (network interface) left. The network and application layer are left empty because they are linked into the simulation by means of the PAWiS node.

III. Experiments and Results

We evaluated the successfull integration of MiXiM, PAWiS, and STEAM-Sim investigating a real-world received signal strength indicator (RSSI) readout scenario. A base station transmits a synchronization frame (beacon) every second to trigger several sensor nodes to deliver data. The base station receives the data of the sensor nodes and logs the RSSI values of correctly received packets (CRC check passed) to a serial interface. Wireless transmissions are separated in time and therefore a time division multiple access (TDMA) scheme is implemented. For the purpose of simulation the output to the serial interface is saved in a log-file.

Each node comprises an MSP430FG4618 microcontroller and a CC2500 transceiver to transmit a packet consisting of 4 bytes of preamble, a 4-byte synchronization word, 3 bytes of header (length, address, packet type), 2 bytes payload and 2 bytes CRC using 2-FSK modulation at a datarate of 2.4 kBaud and an output power of 1 dBm. The chosen slot time is 60 ms.

We compared the results regarding RSSI values and energy consumption between our ”golden model“ and the combined framework (MiXiM, PAWiS, and STEAM-Sim). The ”golden model“ is build upon STEAM-Sim which integrates PAWiS and was verified by means of measurements in [2]. To ensure comparable results we had to include the PAWiS free space propagation model as described in [4] in the combined framework. The new analogue model was configured with attenuation exponent b equal to two and the effective antenna area equal to $9.87670\,\mathrm{cm}^2$.

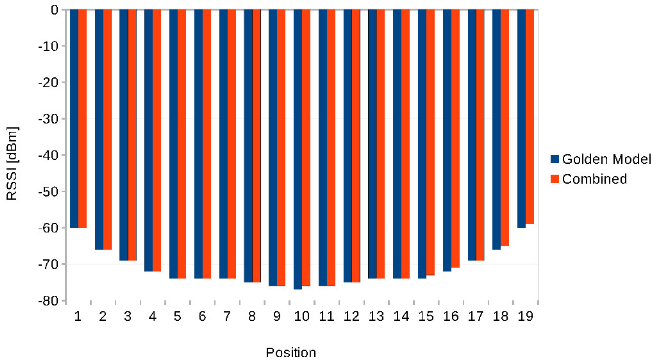
A. *Static Scenario*

In this scenario the base station was placed in the center of a $100\mathrm{x}100\,\mathrm{m}^2$ area. The nine sensor nodes were placed arbitrarily in this area. The results for the RSSI values and the consumed energy of the ”golden model“ and the combined simulation framework matched exactly.

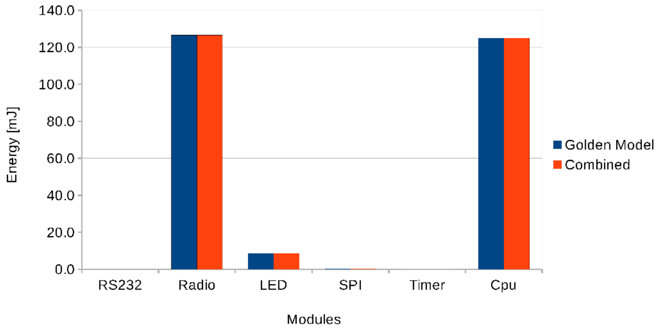
B. *Mobile Scenario*

The second evaluation scenario incorporates a mobile sensor node and therefore requires the usage of a mobility pattern. We used the rectangular movement pattern provided by MiXiM, i.e., a mobile sensor node moves with ten metres per second anti-clockwise through nineteen positions describing a rectangle. At every position it sends a packet to a fixed positioned base station. PAWiS does not provide mobility patterns, so we implemented such a movement by means of LUA scripts in the ”golden model“. Since we do not use a linear continuous movement (i.e., we set the positions discrete) the simulation results differ from each other as can be seen for position 10 in Fig. 4(a). Further, the MiXiM rectangular movement pattern introduces a random deviation from the starting position. Nevertheless, the absolute error is 1 dBm which corresponds to the resolution of the modeled hardware.

We further simulated the energy consumption of the mobile sensor node as depicted in Fig. 4(b). As can be seen, the results between the ”golden model“ and the combined framework match exactly.



(a) RSSI readout values



(b) Consumed energy of the mobile sensor node

Fig. 4. Evaluation results for the integrated framework

IV. Conclusions

In this paper we identified shortcomings in state-of-the-art simulation environments of WSNs, namely focusing on special characteristics of WSNs such as mobility and realistic channels models. We have presented a way towards an integration of well-known, established, and daily used simulation frameworks. MiXiM especially offers mobility and sophisticated channel models whereas PAWiS provides accurate modeling of the timing and energy consumption of hardware components. The STEAM-Sim simulator enables the time accurate simulation of real-life application code in an efficient way. Our system integration approach, which was verified by two simulation studies, will give the community the opportunity to establish accurate simulation of mobile WSNs and WSANs.

V. Acknowledgements

This work has been supported by the Linz Center of Mechatronics in the framework of the Austrian COMET-K2 programme.

References

[1] A. Köpke, M. Swigulski, K. Wessel, and et. al. Simulating Wireless and Mobile Networks in OMNeT++ – the MiXiM Vision. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques*, SIMUtools'08, 2008.

[2] G. Möstl, R. Hagelauer, G. Müller, and A. Springer. STEAM-Sim: Filling the Gap Between Time Accuracy and Scalability in Simulation of Wireless Sensor Networks. In *Proceedings of the 8th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N'13, 2013.

[3] A. Varga. The OMNeT++ Discrete Event Simulation System. *Proceedings of the European Simulation Multiconference (ESM)*, 2001.

[4] D. Weber, J. Glaser, and S. Mahlknecht. Discrete Event Simulation Framework for Power Aware Wireless Sensor Networks. In *5th IEEE International Conference on Industrial Informatics*, 2007.