Hy-Sim: Model based Hybrid Simulation framework for WSN application development

Zhen Yu Song
Politecnico di Torino
c.so Duca degli Abruzzi, 24
Torino, Italy
zhenyu.song@polito.it

Mohammad Mostafizur Rahman Mozumdar
UC Berkeley
545H Cory Hall, UCB
Berkeley, USA
mozumdar@eecs.berke-ley.edu

Maurizio Tranchero
Politecnico di Torino
c.so Duca degli Abruzzi, 24
Torino, Italy
maurizio.tranchero@polito.it

Luciano Lavagno
Politecnico di Torino
c.so Duca degli Abruzzi, 24
Torino, Italy
luciano.lavagno@polito.it

Riccardo Tomasi
Istituto Superiore Mario Boella
via Pier Carlo Boggio 61
Torino, Italy
tomasi@ismb.it

Stefano Olivieri
The MathWorks
via Bertola, 34
Torino, Italy
stefano.olivieri@mathworks.it

ABSTRACT

Bridging the physical world with the virtual one often broadens the possibilities of accelerating and easing embedded system design. This is even more true for WSNs, where generally the developed applications need to be tested and executed in hundreds to thousands of nodes. Often times, it is hard to manage test beds that have huge number of nodes. The most common solution is to rely on simulation frameworks that allow the developers to create virtual sensor nodes and then provides levels of abstraction to specify the applications which will be executed on the nodes. The foremost drawback of this kind of simulation is the absence of direct interfaces with the physical environment. Hence in this paper we propose a hybrid simulation framework for WSN application development that interconnects a virtual network with the physical network and then allows one to simulate the networks as a whole. Moreover, the developers model WSN applications by using high level abstractions which could be used for multi-platform automatic code generation (in TinyOS and Ember ZigBee platforms).

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Design, Language, Theory

SimulationWorks 2010 March 15–19, Torremolinos, Malaga, Spain.

Keywords

Wireless Sensor Networks, Model-Based Design, Hardware-In-the-Loop, Automatic Code Generation

1. INTRODUCTION

Wireless Sensor Networks (WSNs) are communities of cooperative heterogeneous embedded systems used for monitoring and control applications. WSNs, and, more generally, Cooperating Objects [14, 17] have been a very active research topic in last years and are now a mature technology, used more and more in actual on-the-field applications and production environments.

Most application scenarios require WSN to behave as a "disappearing technology," meaning that the final user must not perceive the presence of the operating devices, but just get the required results, without any configuration or management effort. This operative paradigm calls for a significant improvement of reliability and trustworthiness of embedded WSN nodes.

To provide such reliability, however, the WSN developer faces major difficulties especially due to the high degree of distribution of applications, the high heterogeneity of available devices and the scarce debug capabilities provided by the WSN nodes (i.e., lack of user interface for each node, scarce processing capabilities, lack of on-chip debugging,. . .).

For such reasons, it is a common practice to make a large use of simulation during the development phase. However, it is not always possible to fully take into account the behavior of the actual WSN hardware, because in some cases such behavior cannot be modeled or predicted a-priori. This issue is expected to become more critical with the growth of importance of paradigms such as cyber-physical systems [11], where the link between computation and "live" physical devices gets tighter.

Because of this tightening link, a growth of the need for simulation tools ready to be interfaced with the real-world devices is foreseen, and this can be done only through Hardware-In-the-Loop (HIL) techniques [20]. The motivation for the introduction of HIL is simple: to cope with hardware dependent, physical-related aspects while at the same

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

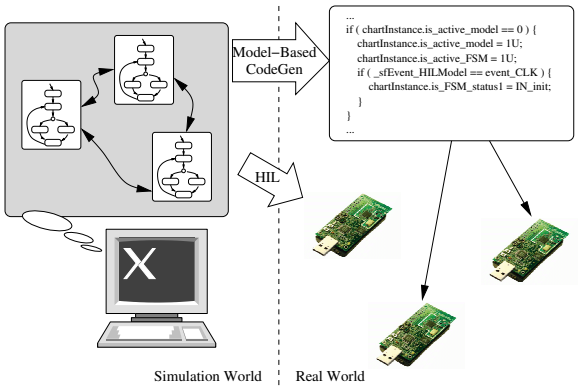


Figure 1: The Hy-Sim concept

time exploiting simulation to cope with aspects related to distributed, complex, cooperative and large-scale environments.

The introduction of HIL gives the WSN developer the possibility to make a giant step from the simulated world to the actual implementation. Such approach can provide even more advantages if integrated in a single framework with the WSN application modeling and code-generation techniques described in [15]. The result of such integration is a novel model-based hybrid simulation framework for WSN application development, named Hy-Sim. The Hy-Sim concept is outlined in Figure 1.

The engineering approach used within the framework follows a development flow consisting of four different steps. In the first phase, called "design phase," application-level embedded software can be designed and developed using only platform-independent, statechart- and block diagram-based abstractions, provided by the MathWorks tools [19]. In the second phase, called "simulation," such models can be configured to run within a fully simulated environment, provided with synthetic communication models and simulated physical sensors. In the third phase, called "hybrid simulation," exploiting HIL extensions it is possible to link some of the simulated WSN nodes with hardware depended features. In the fourth phase, the developed models can be used to automatically generate code for different WSN platforms (such as TinyOS [13], or Ember ZigBee [4]), that run on real-world WSN nodes.

The hardware dependent features used in the last two phases are the same: for such reason it is also possible to create WSN deployments consisting of any combination of simulated, partially simulated and real WSN nodes. Thus by using Hy-Sim, users can build a hybrid network consisting of virtual and real nodes and then simulate it as a whole.

The remainder of the paper is organized as follows: in section 2, a review of related work is presented; in section 3 the overall architecture of the proposed solution is outlined; in section 4, an example use-case is illustrated; section 5 presents conclusions and future research directions.

2. RELATED WORK

The use of simulation-based approaches is widespread in the area of development for WSN, because they provide means to cope with the intrinsic difficulties of the highly distributed environment and the relative lack of standard debugging tools. Both generic [3, 21] and specialized simulators [18, 12, 10, 16] can be used for rapid prototyping, repeatable evaluation of new network protocols and non-intrusive debugging. Simulation is a de-facto standard first-step in any implementation of WSN-based solution. However, such approaches are not completely reliable, for many different reasons. In the first place, the degree of realism of the simulation depends on the complexity of the underlying simulation model, which is often based on unrealistic assumptions (e.g., regarding the communication models). In the second place, code being executed within the simulation is different from the software running on actual WSN nodes, mainly because of differences in the HW platform, which introduces discrepancies regarding simulation timing, concurrency and performance. Finally, even in cases where the simulation code is very similar to the actual code (such as in [12]), the simulation necessarily behaves differently regarding access to platform-dependent components, such as radio transceivers or sensors. For such reasons, WSN developers are usually forced to engage in a time-consuming debug sessions when they move from the simulation to the actual deployment phases. The hybrid simulation approach, mixing real and simulated objects through HIL interfaces, might help reduce efforts in such intermediate phase. Hybrid hardware in loop simulation is a consolidated approach, especially in area of development of embedded systems for automation and control [8, 1, 7]. In hybrid simulation, parts of the system are simulated and some parts are real-world systems integrated through HIL. A co-simulation framework usually is in charge of controlling the interactions between the simulated and the real world. Despite complexity and technical difficulties, hybrid simulation is valuable because it can couple the realism of actual deployment with the flexibility of simulation. Hybrid simulation has already been used in WSNs, for both testing and debugging purposes. In [9] hybrid simulation is used specifically for testing embedded software for TinyOS-based applications, using a wireless-based control channel. In EmStar/EmTOS [5, 6], hybrid simulation (called "emulation mode") is used as a means to enhance simulation with real radio channels. In [22], hybrid simulation is used to extend TOSSIM capabilities, employing a time-freeze strategy to keep the simulation running at pace with the real world. In [23] the same approach is used, but with the opposite purpose, i.e., to "augment" a real network with a set of simulated nodes.

Even though research has achieved important goals, many aspects still require more work. Indeed, many of these tools and methodologies are not suited for describing complex designs or lack some modeling aspects which are essential to create a fully integrated co-design environment which can actually be used in an industrial setting. As a typical example we can cite works based on SystemC modeling [2], which are simple to use and powerful, but introduce a virtual machine on the node to execute the code, and hence are not optimal for low power and high performance applications.

Another approach introduced in [15] can lead to better implementation, since it can generate code automatically for various WSN platforms from the high level application model, instead of using a virtual machine. But the proposed framework suffers from the lack of modularity and also there is a small support for HIL interactions with the developed model. In this paper the Simulink-based frame-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

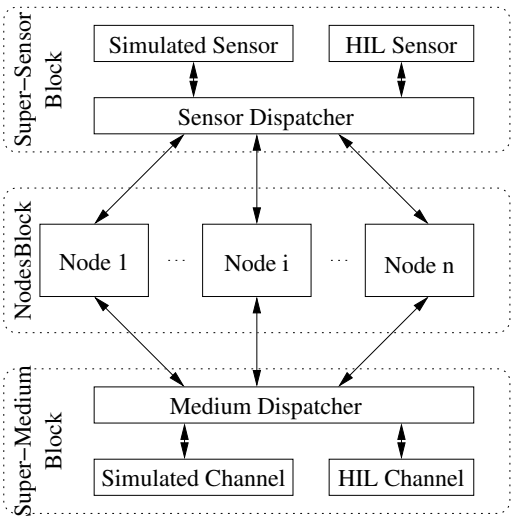


Figure 2: Hy-Sim Architecture

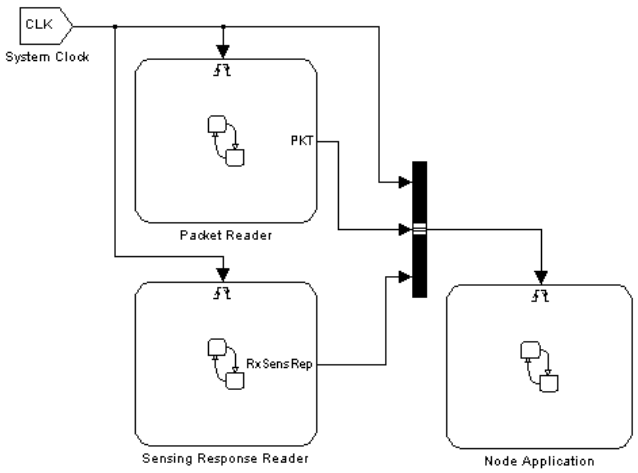


Figure 3: Node instance structure

work proposed in [15] is extended in order to achieve better modularity, portability, and to provide complete HIL interfaces. The driving goal of this extension is to provide WSN developers with tools to develop platform-independent WSN application through model-based abstractions, test the implementations through simulation, extend the realism of simulations through HIL (thus introducing hardware dependent features) and finally enrich the simulations with fully real nodes, through code-generation.

3. HY-SIM ARCHITECTURE

The architecture of the Hy-Sim framework, implemented using Simulink/Stateflow [19], is depicted in Figure 2. A brief description of the functionalities of each component in the framework is given below:

1. **Nodes Block.** This is a container block which includes a separate instance for each node included in the scenario. Node objects can be either fully simulated, partially simulated or independent. Fully simulated nodes are completely handled inside the simulator; partially simulated nodes are modeled inside the simulation framework but are able to access HIL components (such as sensors, or transceivers) which reside on-board physical devices; finally, independent nodes are actual nodes which interact with fully or partially simulated nodes (or vice versa) only through the radio.

2. **Super-Medium Block.** This is a component devoted to managing all radio communications within the scenario, and is able to dispatch packets within the simulated world and also between the simulated and actual world.

3. **Super-Sensor Block.** This is a component devoted to managing all sensing aspects within the scenario, and is able to provide simulated sensor readings and means to access actual sensors through HIL interfaces.

In the following subsections, *Nodes* (Sec. 3.1), *Super-Medium* (Sec. 3.2), and *Super-Sensor* (Sec. 3.3) blocks are illustrated more in detail.

3.1 Nodes Block

The Nodes Block is an abstraction that contains nodes which will interact with each other in the application scenario. Each node is represented by a separate instance composed as shown in Figure 3.

The *Node Application* is a block representing the platform-independent model of the WSN application running on each node, realized in Stateflow or Simulink. The *Node Application* can interact with the rest of the world through the *Packet Reader* and *Sensing Response Reader* providing direct connection to and from respectively the Super-Medium and the Super-Sensor. The *Packet Reader* behaves as a radio receiver: upon detection of a radio packet, it generates an event (PKT) towards the *Node Application* component, making the payload of the received packet available to it. Similarly, a *RxSensRep* event is used by the *Sensing Response Reader* to notify availability of a sensor reading. A *System Clock* synchronous signal, finally, provides timing support to the node components.

Although each node instance is structured according to the same scheme, its actual behavior within the simulation depends on its type. In the current version of the framework, five different node types are defined: SIM, HIL_SEN, HIL_RF, HIL_FULL and REAL. The characteristics of such types of nodes depends on how they interact with the *Super-Medium* and the *Super-Sensor* and are described in the following. Note that the application on model is totally unaware and independent of the node type, since the Hy-Sim framework handles these aspects transparently.

- SIM (SIMulated): fully simulated node, using both virtual sensors and simulated radio transceiver.

- HIL_SEN (Hardware-In-Loop SENsor): partially simulated node, using the simulated radio transceiver but collecting sensor data from one or more actual sensor nodes through the *Super-Sensor*.

- HIL_RF (Hardware-In-Loop Radio): partially simulated node collecting data only from virtual sensors, but using the actual transceiver through the *Super-Medium* for sending and receiving packets.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

- HIL_FULL (Hardware-In-Loop FULL): partially simulated node running the application within the simulation, but using both actual sensors and the actual transceiver.
- REAL: fully independent node, existing only in the physical world. Such nodes can have active radio communication with other types of node, thanks to HIL nodes connected to the *Super-Medium*. The application running on it could be coded manually (i.e., using platform-dependent programming languages and operating system, such as TinyOS), or can more easily be generated automatically from the Stateflow model of *Node Application*.

During the initial scenario definition phase, users can generate instances of any number and combinations of the aforementioned node types.

For all partially simulated nodes, the WSN developer can specify the association between any WSN node instance and HIL interfaces in the Super-Medium and in the Super-Sensors. This relationship associates partially simulated nodes with actual HIL radio transceivers or sensors. Note that multiple HIL nodes can be associated with the same physical device, but this procedure must be handled carefully, because it can generate inconsistencies (e.g., it is not possible to send two radio packets at the same time using the same radio physical radio transceiver). A simple method is adopted by now to resolve the resource contention, which is dropping the collided communication.

The main idea beneath such method is to provide a simple, yet modular and flexible approach to extend high-level, WSN application simulation with HIL, letting the WSN developer specify the hybrid simulation configuration on-the-fly.

3.2 Super-medium Block

The *Super-Medium* block manages the exchange of packets within the framework including the exchange between (fully and partially) simulated nodes, and the exchange between (fully and partially) simulated nodes and real nodes. A high-level view of the logic behind the *Super-Medium* Block is depicted in Figure 4. The *Super-Medium* Block works by performing read and write operations over *tuples* of incoming and outgoing buffers, one tuple per node in the scenario. Packets are exchanged among buffers according to different parameters, such as the type of the transmitting node, the type of the receiving node and the type of the link among the two, depending on the radio model configured by the user. For instance, a packet directed to a REAL node might be transferred directly to an actual device through the HIL interface, while a packet exchanged among two SIM nodes is first processed by a simulated radio channel model and then, in case of success (no packet error), simply transferred between the two virtual nodes.

A snapshot of the *Super-Medium* Simulink block is presented in Figure 5. It is composed of four sub-blocks, namely the Simulated Channel and the HIL Channel, plus two blocks used to handle communication from and towards nodes. The role of such blocks is to collect all packets sent by node instances, process them either via simulated radio models (Simulated Channel) or using actual radio channels (HIL channel) and then notify packet reception to all nodes which received some packets.

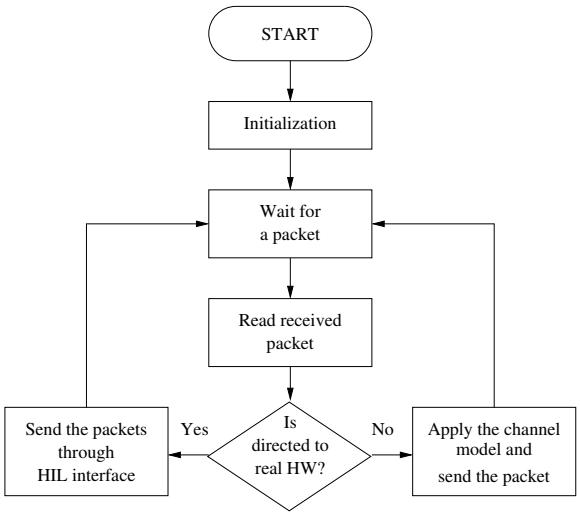


Figure 4: Super-Medium logic

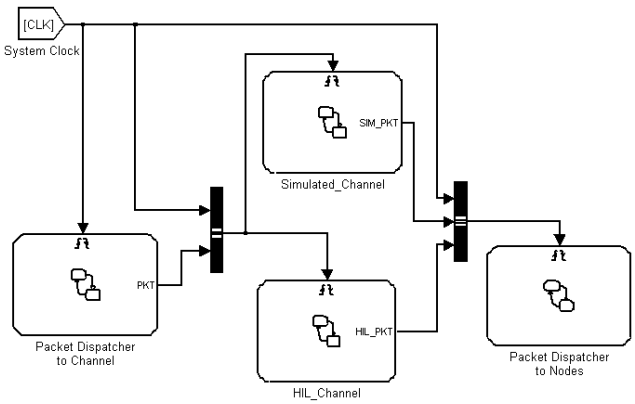


Figure 5: Structure of Super-Medium Block

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

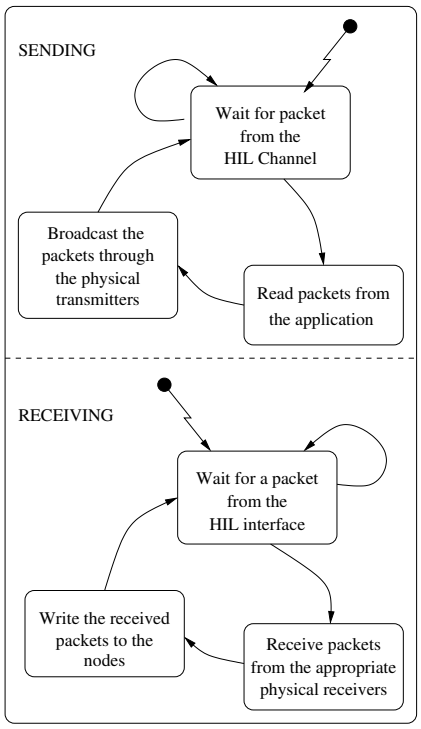


Figure 6: HIL_Channel logic

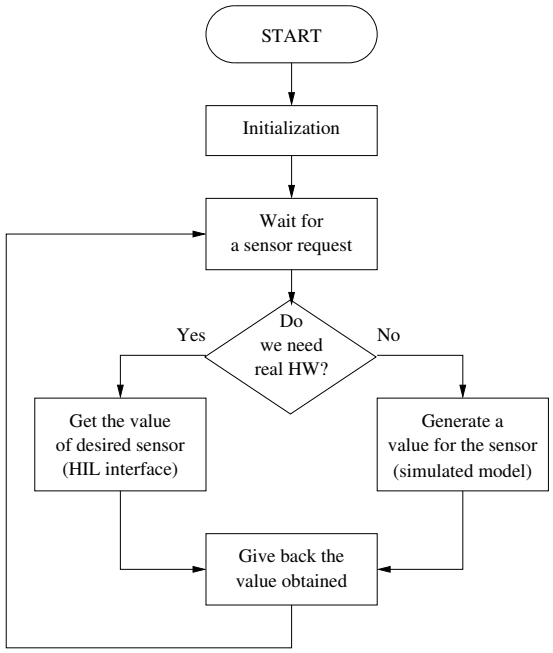


Figure 7: Workflow of Super-Sensor Block

3.2.1 Simulated Channel

Currently the Simulated Channel uses a simple implementation based on a link-quality matrix specified in the configuration phase, containing packet loss probability among any couple of nodes. Such implementation can be replaced by other simpler or more complex radio model implementations. Current implementation dose not take into account of possible interferences on transmitted packets but can be easily integrated into the model by the users.

3.2.2 HIL Channel

The HIL Channel handles radio communication from and towards actual radio nodes. The role of HIL Channel is to handle and intercept packets to be broadcast on the physical radio interface (either directed to a HIL_RF or HIL_FULL node, or sent to broadcast addresses including REAL node). The HIL Channel controls, through serial cables, the radio interfaces of one or more physical devices. The logic that controls the HIL Channel is represented in Figure 6. A software stub running on the physical devices is responsible for dispatching radio packets from the physical interface towards the framework and vice-versa.

When the HIL_Channel object receives a packet through the HIL interface, it is able to decode the packet header and dispatch the packet to the correct destination node (or nodes) according to the association between the node ID and the HIL interface.

3.3 Super-Sensor

A high-level view of the behavior of the Super-Sensor Block is sketched in Figure 7.

Since sensor interactions are usually local to a node, its design is much simpler than the Super-Medium, though they share some similarities. Each node instance is provided with an incoming sensor data buffer. When a new sensor reading is available, the *Node Application* is notified.

As described before, the Super-Sensor is able to receive sensing requests from all node instances. Upon reception of a request, the Super-Sensor generates a sensing value in different ways, according to the node type and the request type. In case of nodes with simulated sensors (SIM, HIL_RF), the sensor reading is generated according to a simulated model (e.g., a data file) defined in the initial configuration phase. In case of nodes with HIL sensors (HIL_SEN or HIL_FULL), a request is routed to a specific HIL interface connected to a physical device.

A snapshot of the Super-Sensor Simulink block is depicted in Figure 8.

The Super-Sensor is composed of three sub-blocks: Simulated Sensor, HIL Sensors and a third block handling sensing data exchange with the nodes.

4. EXAMPLE: A SIMPLE TOKEN-RING NETWORK

In order to illustrate how to design, simulate and test the WSN application based on this framework, we show a small example describing a token-ring network.

The algorithm developed implements a token ring network in which each wireless sensor node

1. awaits for the token (transmitted via the radio channel),
2. collects several acceleration sensing values, and
3. transmits the token to the following node.

Tokens are passed according to the node ID (see Figure 9).

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

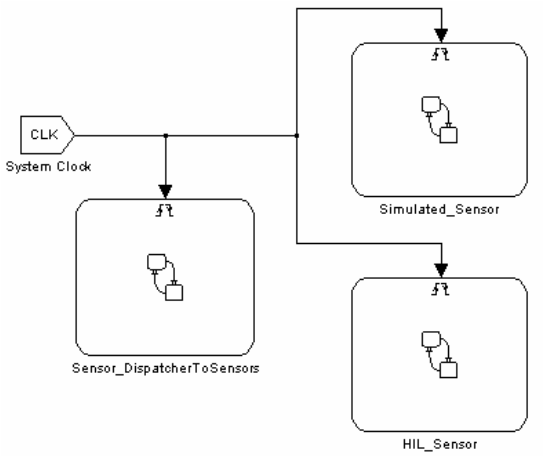
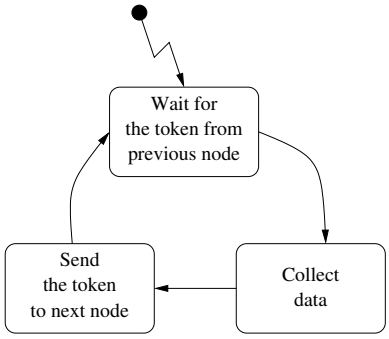
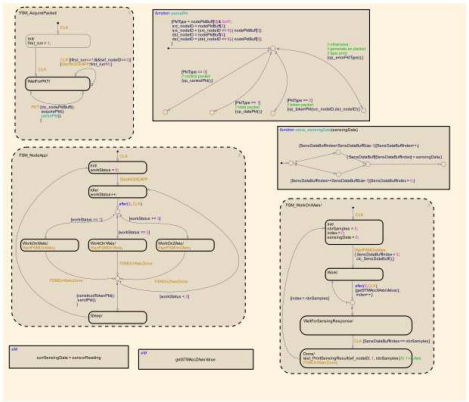


Figure 8: Structure of Super-Sensor Block



(a)



(b)

Figure 9: FSM of the simple token-ring network developed (a) and its Simulink implementation (b).

When the node receives the token, it collects 3 acceleration values on X-axis, 4 acceleration values on Y-axis and 5 acceleration values on Z-axis. Then the current active node sends out the token and enters into the sleep mode until it is woken up by receiving the token again.

After designing the WSN application, the user should set the configuration of the nodes and the network. A matlab-file is used to perform this customization: the user can easily modify it by hand, but we are also working towards the implementation of a graphical user interface able to further simplify this operation. The execution of this script automatically generates the network and sets up the environment for the simulation. In case of HIL nodes, this operation configures also the hardware interfaces to allow the communication among simulated and hardware nodes.

4.1 Results

The application was tested with a different number of nodes, to give an idea of the computational power needed to use the framework. Table 1 shows the simulation time for different network dimensions. Each setup is configured to have one fourth of fully simulated nodes and three fourths of HIL ones, one for each HIL configuration (i.e., HIL_SENS, HIL_RF, and HIL_FULL). The last three columns are showing respectively the execution time required to complete the simulation of the entire network, the execution time of a single fully simulated node, and the execution time of a HIL node (independently from their configuration, HIL nodes are requiring about the same time to be simulated).

From these value it can be seen how the network simulation time scales linearly with its dimension. HIL nodes, have a big impact on the overall simulation time, since, at the moment, the communication among the framework and the hardware modules is performed synchronously, but we are looking forward to have an asynchronous communication available soon.

After the model has been developed and simulated, we have generated application code automatically both for TinyOS [13] and for Ember ZigBee [4] platforms. Firstly, by using Stateflow Coder [19], we generated ANSI C code for the token-ring application. Then by executing TLC scripts [19] we added platform dependent code both for TinyOS and for Ember separately. Table 2 denotes the size of the automatically generated code for both platforms. In this table we also noted the size of the platform base code (without any application code) to have an idea about the code size of token-ring application. These results show how this methodology can be used with very small overhead, compared to the software infrastructures needed in the desired platform (i.e., TinyOS and ZigBee stack), making the usage of automatic code generation affordable even for low performance CPUs.

5. CONCLUSIONS AND FUTURE WORK

We described an extensible framework for platform independent application modeling and hybrid simulation for sensor network algorithms based on MathWorks tools. To the best of our knowledge, this is the first time that a framework of this sort has been developed and tested for the sensor network domain. The reason for choosing the MathWorks tools over, for example, TOSSIM, NS, OmNet, is that they are well known and already provide rich libraries for digital signal processing and control algorithm behavior simulation.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

Table 1: Simulation time for the token-ring network.

Number of Nodes	Number of Neighbours	Node Types		Simulation Time [s]		
		SIM	HIL	Global	SIM	HIL
8	5	2	6	818	0.41	136
16	8	4	12	1634	0.55	136
32	16	8	24	3295	0.87	137
64	32	16	48	6592	1.01	137

Table 2: Code size and memory usage for the token-ring application

Software System	Memory	Platform base code with libs no application code (bytes)	Automatic code generation (bytes)
TinyOS	ROM	16366	17958
	RAM	840	918
Ember ZigBee	ROM	87326	89662
	RAM	2738	2790

They also provide extensible mechanisms for efficient code generation and platform-specific re-targeting. A possible extension of this work would be to provide more library functions for sensors and extending the framework for simulating a more detailed channel model fully inside Simulink.

6. REFERENCES

[1] E. Bellei, E. Bussolino, F. Gregoretti, L. Mari, F. Renga, and L. Reyneri. Simulink-based codesign and cosimulation of a common-railtm injector test bench. *Journal of Circuits, Systems, and Computers*, 12:171–202, 2003.

[2] N. Bombieri, F. Fummi, and D. Quaglia. TLM/network design space exploration for networked embedded systems. In *Proc. of CODES+ISSS*, pages 53–56, seoul, korea, december 2006. ICM.

[3] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. In *IEEE Computer*, pages 33(5):59–67, May 2000.

[4] Ember. *Zigbee Wireless Semiconductor Solutions by Ember*. http://www.ember.com.

[5] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 24–24, Berkeley, CA, USA, 2004. USENIX Association.

[6] L. Girod, N. Ramanathan, J. Elson, T. Stathopoulos, M. Lukac, and D. Estrin. Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks. *ACM Trans. Sen. Netw.*, 3(3):13, 2007.

[7] H. Hanselmann. Hardware-in-the-loop simulation testing and its integration into a cacsd toolset. In *Computer-Aided Control System Design, 1996., Proceedings of the 1996 IEEE International Symposium on*, pages 152–156, Sep 1996.

[8] R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7(5):643 – 653, 1999.

[9] D. Jia, B. Krogh, and C. Wong. TOSHILT: middleware for hardware-in-the-loop testing of wireless sensor networks. `www.ece.cmu.edu/webk/sensor_networks/pub/ipsn05_hilt.pdf`.

[10] A. K opke, M. Swigulski, K. Wessel, D. Willkomm, P. K. Haneveld, T. Parker, O. Visser, H. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++: The mixim vision. In *First International OMNeT++ Developers Workshop*, March 2008.

[11] E. A. Lee. Cyber physical systems: Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[12] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 126–137, November 2003.

[13] P. Levis, S. Madden, and D. Gay. *TinyOS: An Operating System for Sensor Networks*. Ambient Intelligence edited by W. Weber, J.Rabaey, and E. Aarts, Springer-Verlag, 2004.

[14] P. Marwedel. *Embedded System Design*. Kluwer Academic Publishers, Boston, Massachusetts, 2003.

[15] M. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri. A framework for modeling, simulation and automatic code generation of sensor network application. In *Proc. of SECON*, pages 515–522, San Francisco, California (USA), 2008.

[16] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *In Proceedings of the First IEEE InternationalmWorkshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, pages 126–137, November 2006.

[17] Pedro José Marrón et al., CONET Consortium. *Roadmap on Cooperating Objects*. Kluwer Academic Publishers, Luxembourg, EU, 2009.

[18] L. Shu, C. Wu, Y. Zhang, J. Chen, L. Wang, and M. Hauswirth. Nettopo: Beyond simulator and visualizer for wireless sensor networks. In *Second International Conference on Future Generation Communication and Networking (FGCN 2008)*,

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662

December 2008.

[19] The MathWorks. Stateflow, simulink and real-time workshop. `http://www.mathworks.com`.

[20] R. C. Underwood, B. M. McMillin, and M. L. Crow. An open framework for highly concurrent real-time hardware-in-the-loop simulation. In *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 44–51, Washington, DC, USA, 2008. IEEE Computer Society.

[21] A. Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM)*, pages 319–324, June 2001.

[22] D. Watson and M. Nesterenko. Mule: Hybrid simulator for testing and debugging wireless sensor networks. In *Workshop on Sensor and Actor Network Protocols and Applications*, August 2006.

[23] Y. Wen, W. Zhang, R. Wolski, and N. Chohan. Simulation-based augmented reality for sensor network development. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 275–288, New York, NY, USA, 2007. ACM.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8662
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8662