Electronic Communications of the EASST
Volume 70 (2014)

Proceedings of the
14th International Workshop on
Automated Verification of Critical Systems (AVoCS 2014)

A Formal Co-Simulation Approach for Wireless Sensor Network Development

Adisak Intana, Michael R. Poppleton, Geoff V. Merrett

15 pages

Guest Editors: Marieke Huisman, Jaco van de Pol
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Formal Co-Simulation Approach for Wireless Sensor Network Development

Adisak Intana, Michael R. Poppleton, Geoff V. Merrett

ai1n10,mrp,gvm@ecs.soton.ac.uk
Electronics and Computer Science
University of Southampton, Southampton, SO17 1BJ, UK

Abstract: This paper proposes a Formal Co-simulation (*FoCoSim-WSN*) framework to provide a good software engineering practice for wireless sensor networks (WSNs) including high-level abstraction, separation of concerns, strong verification and validation (V&V) techniques. This provides an iterative interworking framework which combines the benefits of existing simulation and proof-based formal verification approaches. The complexity of software development for the sensor node controller is reduced by separating the controller model from the simulation environment. Controller algorithms from application through network and MAC layers can be formally developed and verified in a layered manner using the refinement method of the Event-B language and its RODIN toolkit. The absence of certain classes of faults in controller models which cannot be guaranteed by simulation testing techniques, can be proved by formal methods. On the other hand, the MiXiM simulation of physical environment provides full confidence about reliability and performance analysis through long running simulation via wireless channels. Our prototype development confirms the flexibility of the framework for interworking between formal, simulation and co-simulation modelling.

Keywords: formal modelling and analysis, Event-B, proof, simulation, wireless sensor network, co-simulation

1 Introduction

A Wireless Sensor Network (WSN) is a distributed system of cooperating devices that performs distributed monitoring applications in a physical environment over a self-organised wireless network topology. In traditional WSN development, WSN requirements are tackled with a "code-and-fix" process [Pie10] in which the code is implemented on the real hardware. The WSN application is developed under the constraints of display-less, low-level specific platform and low-power design. In safety-critical application domains where WSNs are increasingly being adopted - from healthcare to military - the demand on verification is high [Pie10, BRWR10, ACB10]. Functional requirements including safety/liveness properties have to be considered together with performance and reliability requirements of the network.

Simulation is usually used at an early stage of designing and testing communication protocols because it provides the higher level of abstraction [KM07, ISH10]. It abstracts away from specific operating system platforms whereas other testing techniques such as emulation and laboratory testbeds do not. In current simulation practice, protocols and algorithms are layered to create a communication networking protocol suite by a standard protocol stack scheme. These are integrated with a stochastic environment framework of wireless channel, radio and analogue models to generate the long running testing scenarios. The simulation and performance analysis such as network latency and energy consumption are conducted independently from any specific platform. However, code for simulation is developed monolithically; current practice is a long way from model-based software engineering process. The specification of the behaviours of the software controller algorithm and the behaviour of the environment are implemented in simulation at the same time. This gives significant complexity to manage during development and makes it hard to understand the code. Thus, a clear separation of concerns is required in this aspect. Furthermore, critical design errors are not guaranteed to be discovered during simulation. This technique cannot guarantee the absence of certain classes of faults as discovered in [IPM13].

Formal Methods have been considered to design and verify the WSN application and protocol. For example, formal analysis is proposed in [MRDD10] to detect critical network elements with OMNeT++. The framework proposed by [WBLS09] also indicates the translated formal specification in PVS (Prototype Verification System) from the logic-based Network Datalog language (NDlog) to guarantee the protocol behaviour. In [IPM13], the proof-based formal method gives a strong guarantee for the absence of faults. Certain functional requirements and safety properties are encoded as invariants in Event-B and it is proved that this invariant is always satisfied by the system behaviours before actual implementation. However, the complexity and scalability of applications need long running simulation behaviours to give full confidence about reliability and performance requirements in formal models.

To increase the quality of current SE practice for WSN development, this paper is implementing the vision proposed in [PM12]. We construct the infrastructure for co-simulation between formal Event-B WSN models and MiXiM environment simulation engines. This provides an integrated set of methodologies for WSNs: (S)imulation, (F)ormal and (C)o-simulation;see Figure 1. The *FoCoSim-WSN* framework is proposed which is a formal co-simualtion method for Event-B and MiXiM for WSNs.

(S) S-style development is the traditional WSN development style that layers the protocol algorithms and evaluates the network performance as mentioned earlier. Target code based on the simulation model is generated together with standard platform-specific libraries. Node level simulation or emulation takes place to test the correctness and performance of the real code before the real world deployment.

(F) F-style modelling represents the requirement specification, modelling and verification in a formal modelling language [IPM13, Abr07, ABHV06]. Each protocol algorithm is layered and verified through refinement steps at network level development. The verified network model is produced before different refinement paths are encoded with requirements for the specific dependent platform at node level. The final, verified node code is generated with standard libraries for a specific hardware platform from this verified node model.

(C) our C-style prototype framework enables the complexity of development to be reduced by separating the software controller from the environment. Formal methods provide the controller, a formal model of code in the real nodes, containing the protocol algorithms separately. An environment simulator provides stochastic sensed data and radio environment, allowing simulation scenarios to be defined as required. The verified controller model for each layer of protocol stack - ultimately, down to verified generated code - is co-simulated with the environment model to perform the performance analysis. A master co-simulation language and algorithm is required to integrate and manage the component simulators. The formal Event-B controller model simulated by ProB can co-simulate with a sensor environment provided by MiXiM via this master.

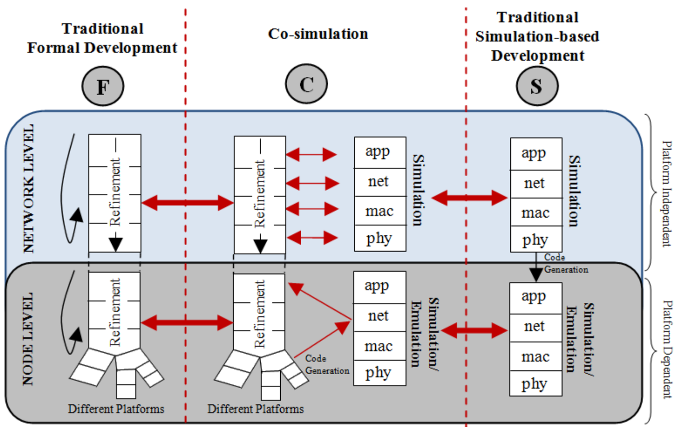


Figure 1: Vision of co-simulation approach for WSN development

In this work, our *FoCoSim-WSN* framework cosimulates between node controller models on the Event-B simulation and sensor environment models on MiXiM simulation. Each network algorithm at each protocol layer is separated from the wireless and physical environment and modelled in Event-B language. A master algorithm is developed by Groovy language to coordinate between network algorithm and environment models. As MiXiM can provide a socket interface in order to integrate its network models together with other simulation environment such as the Vehicles in Network Simulation (Veins) [SGD11] framework[1], we mock-up our own interfaces for MiXiM containing a socket interface for co-simulation. This work focuses on the network level of the development. We exercise our *FoCoSim-WSN* framework with the two abstraction layers at the network level development, application (app) and network (net). The two lower layers, MAC and physical (phy), are our work in progress. The node level development remains work for the future.

The remainder of this paper is organised as follows. Sections 2 and 3 discuss related work and introduce the running case study, an environment monitor system. In Section 4, we demonstrate the strengths and weaknesses of S-style development. Section 5 discusses the F-style approach showing the benefit of specifying and verifying the network algorithm model. We apply a shared-event decomposition to separate the controller from the environment. The main section is Section 6 which introduces our prototype C-style framework (*FoCoSim-WSN*) for co-simulation and demonstrate with the case study. Finally Section 7 gives some conclusion and future work.

[1] see Veins -http://veins.car2x.org/

2 Related Work

Recently, co-simulation frameworks have been proposed in order to co-ordinate between the software controller model and physical simulation environment. DESTECS [2] [BKL+10] is an integrative co-simulation framework for co-modelling and co-simulating between discrete-event (DE) and continuous-time (CT) of physical models via XML-RPC Interface. The formal simulation of ADVANCE[3] [ADV13] provides a framework for integrating multi cyber-physical systems using different simulation engines via Functional Mock-up Interfaces (FMIs)[4]. A master-slave algorithm to execute the co-simulation is implemented in the Groovy language of the ProB tool.

A hybrid design framework is needed for WSN development when the WSN application closely interacting physical environments has become more complex. The model-based system design (MBSD) framework for WSNs proposed by [WB12] co-simulates event-triggered components illustrating network algorithms together with continuous dynamic behaviour exhibited by physical environment. HybridSim[WB13] adopts FMI standard to co-simulate between sensor application models provided by TinyOS and simulation environment generated by Modelica. SysML (based on UML) is applied to the work described above [WB12, WB13] to express the application abstraction. Their work is similar to our work in which the model of the node is co-simulated with the environment. However, their node models do not contain formal elements that leads to lack of formal precise semantics. The closest to our co-simulation framework is NMlab [HSB10] which provides a co-simulation framework for Matlab and ns-2 simulator. The system controller implemented in Matlab co-simulates with the network models provided by ns-2 by using socket interfaces. Similar to this, HarvWSNet [DBMS13], a framework for energy harvesting WSNs, combines the strengths of two development toolkits via a standard socket interface. The power management model is implemented in Matlab to communicate to the wireless sensor network communication model provided by WSNet. However, the communication algorithms at each protocol layer are still implemented in WSNet simulator.

3 Case Study

To demonstrate the effectiveness of our approach, we have extended an environment monitoring system from our preliminary work [IPM13]. This case study is derived from deployed projects described in [BIS+08]. Each sensor node in a network senses data such as temperature and wind speed periodically. This environmental data is regularly sensed and routed wirelessly via multi-hop from the source node to a sink node. As a small number of nodes was initially deployed in *SensorScope* project to evaluate the first use of multi-hop [BIS+08], we implemented our preliminary models consisting of 7 nodes (6 sensor nodes with node 0 representing a sink in Figure 2a) to evaluate the first demonstration of a multi-hop network.

Nodes in the simulation represent the wireless devices with their protocol stacks as shown in Figure 2b. The data that has to be sent to a data sink is collected by *Application Layer* before sending down to lower layers. The task of the *Network Layer* is to manage the route tree used to

[2] see DESTECS -http://www.destecs.org/
[3] see ADVANCE -http://www.advance-ict.eu/
[4] see FMIs -https://www.fmi-standard.org/

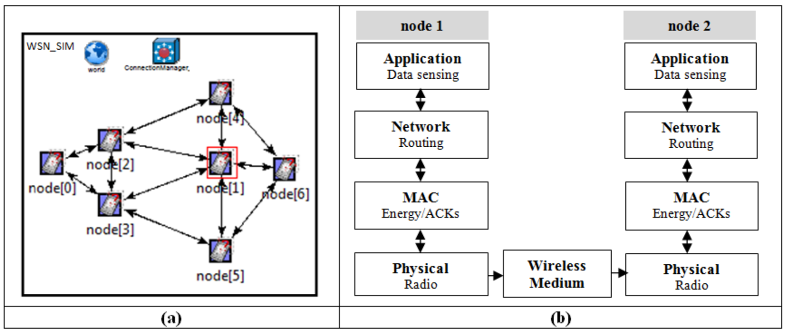


Figure 2: An example of (a) a multi-hop network topology and (b) node's structure

decide the next hop for transmission towards the sink. *MAC Layer* manages power consumption by switching radio on/off to sending/receiving packets and provides an acknowledgement (ACK) mechanism for reliable transmission. The lowest layer, the *Physical layer*, performs the radio propagation for packet sending and receiving.

In this work, we focus on the experiment at the two upper layers, application and network layers. *SensorApp*, a periodic sensing protocol in which the sensed packet is transmitted periodically down to the lower layer, is chosen to implement at the application layer. For the network layer, *MintRoute*, a link quality protocol is selected to be an efficient routing protocol.

MintRoute [WTC03] is a routing algorithm to build the route tree from every node towards a sink. The route tree is dynamically changed based on the link quality between nodes. This link quality is adjusted by the successful rate of transmitted packet delivery. *MintRoute* performs four major steps: (1) *neighbourhood discovery* - one neighbour discovers another neighbour based on broadcast beacon messages, (2) *link quality estimation* - the estimation of reception link quality ratio is calculated periodically by observing the successful rate of receiving packets, (3) *route broadcast* - the transmission link quality ratio of each neighbour is estimated periodically based on the reception link quality ratio attached in "*route update message*", and (4) *parent selection* - this is performed periodically to specify one of the neighbours for routing. The *path cost* towards a sink is calculated based on both two ratios (reception and transmission). Link which has these two ratios less than the quality threshold is not considered. A neighbour with the smallest path cost is chosen as a parent.

4 S-style modelling

To explore the benefits and drawbacks of S-style development, this section describes the simulation experiment on the case study described in the previous section.

MiXiM[5][KSW+08] provides a development framework for the simulation and performance analysis of wireless networks including WSNs. It provides generic and flexible component architecture for models based on a standard network simulation engine, OMNeT++[6]. This layers the development environment into the standard IP protocol stack as shown in Figure 2b. Each

[5] see MiXiM -http://mixim.sourceforge.net/
[6] see OMNeT++ -http://www.omnetpp.org/

layer can communicate with the adjacent layer via communication interfaces named gates. In our simulation model, we extended the base modules (the general structure) for application and network layers provided by MiXiM to implement *SensorApp* and *MintRoute* respectively. The configuration parameters were replicated from the real configuration used in *SensorScope* [BIS+08], as discussed in our previous work [IPM13].

This exercise expresses the strength and weakness of S-style development. The network algorithms can be analysed with the performance evaluation such as the load distribution of the network and the network latency as described in the introduction. However, based on our previous work [IPM13], simulation enables us to discover the loop problem occurring in the route tree but this problem cannot be revealed at all running experiments we performed. This leads us to fix and prove this problem in formal models. Thus, simulation cannot guarantee that the fault in the algorithm will be discovered.

Furthermore, this modelling style causes complexity in development. The controller representing the specific protocol algorithm for each layer in protocol stack has to be completed together with environment elements provided by the standard interface in simulation toolkit (wireless channel communication and connectivity, the library functions - implementing sending/receiving packet and packet encapsulation/decapsulation) to form a single simulation model. This leads us to encounter difficulties of managing such a complex simulation model (especially in *MintRoute* algorithm implemented at the network layer). The next section will demonstrate how to reduce this complexity by using Event-B modelling techniques. Each single controller for each protocol stack in a simulation model is separated and implemented into multiple layers by using the refinement technique.

5 F-style modelling

5.1 Overview of Event-B Modelling

Event-B [Abr07] is a proof-based formal method for specification and verification based on set theory and first order predicate logic. An Event-B model consists of two parts: *context* and *machine*. The context describing the static part contains *carrier sets*, *constants* and *axioms*. The machine represents the behavioural part which consists of three elements: *variables*, *invariants* and *events*. States are described by typed variables. Invariants that state the guaranteed properties of the model express the functional requirement and safety property. Events in Event-B give a state transition. Each event contains *guard(s)* and *action(s)*. The event guards express the necessary conditions that enable the event to successfully and usefully trigger, and actions describe the state transitions over the variables. *Proof Obligations (POs)* are used to state that invariants are satisfied by every event.

Event-B Tool: RODIN [ABHV06] is an open tool platform based on Eclipse. This extensible tool was developed by the European Union ICT Project DEPLOY[7](2008-2012). RODIN includes editors, a proof obligation generator (PO-generator), graphical front ends, theorem provers and the *ProB*[8] animator and model checker.

[7] see DEPLOY - Industrial deployment of system engineering methods providing high dependability and productivity: FP7 Project 214158 http://www.event-b.org

[8] see ProB - http://www.stups.uni-duesseldorf.de/ProB/

Refinement: refinement is a method that allows software engineers to manage the complexity of the development by layering the abstraction of the models. A simple abstract view of essential requirements is implemented first. More requirement or design detail is added at each refinement step until implementation, data structure and algorithm are added to the concrete model in order to bring the model to become close to the real implementation. Refinement POs state that concrete refining events must correctly implement their counterpart abstract refined events.

Shared-Event Model Decomposition: the complexity of large system development is managed by breaking a single machine into sub-machines [But09, SPHB11]. These sub-machines interact by exchanging messages via shared events. The shared-event decomposition Rodin plug-in[9] is applied to this work to decompose a software controller from a sensor environment.

5.2 WSN Development in Event-B

The benefit of using the F-style for WSN development described in Section 1 is presented in this section. Event-B is used to create a WSN specification and its verification. The development approach can be shown in Figure 3. The two upper layers: *Application* and *Network layers* are implemented. The Event-B refinement technique is used to layer the model which corresponds to each layer of protocol stack. We apply our *MintRoute* models proposed in [IPM13]. The refinement structure and some events are adjusted to support our co-simulation framework. Six refinement models are created. This begins with a very simple abstract model (*M0*) in which the data packet is transmitted to a sink in one atomic step. Then, the first and second refinement models (*M1-M2*) fulfill the operation for *SensorApp*. These refinement models define the neighbour node to determine multi-hop network for broadcasting mechanism before they are refined down to implement *MintRoute*. Each step of *MintRoute* protocol is layered (as in *M3-M5*, Figure 3) from *neighbourhood discovery* to *parent selection* as described in Section 3. The models for the C-style of co-simulation development are also prepared by separating the software controller from the environment for both layer protocols. Shared event decomposition technique is applied to decompose the second and fifth refinement models corresponding to application and routing layers respectively. Note that separated environment models *ENV2* and *ENV5* are used for the controller verification and validation before decomposition. In the C-style development, they are replaced with the concrete simulation environment implemented in MiXiM.

For the route tree construction verification, we create safety invariants as shown in Figure 4. This shows us the benefit of proof to indicate no-loop property as a necessary invariant proved after simulation revealed it. We apply the definition of transitive closure (tcl) and the no-loop property proposed by Abrial[Abr10] and applied in [DBA08, HKBA09]. We define the route tree in @*inv3_2*. The initialisation of variable *nodes* contains only a sink ($\{Sink\}$) and the route tree (*cRouteTree*) is initialised to be an empty set. As soon as a node is explored to discover its parent, it is recorded in *nodes* with a pair between this node and its parent put in *cRouteTree*. Flag *completedRoute* in invariant @*inv3_3* indicates the completion of route tree construction. Thus, invariants @*inv3_2* and @*inv3_3* represents that when the route tree is finished, each sensor node must have its own parent. Furthermore, safety invariant @*saf3_1* illustrates the no-loop property. This can guarantee that there are no loops in the routing tree. Theorem @*mth3_3* confirms every

[9] see shared-event decomposition plug-in - http://wiki.event-b.org/index.php/Decomposition_Plug-in_User_Guide

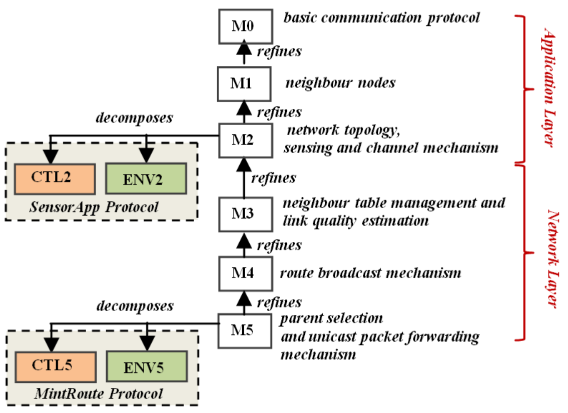

M0
basic communication protocol
refines
M1
neighbour nodes
refines
decomposes
M2
network topology, sensing and channel mechanism
CTL2
ENV2
SensorApp Protocol
refines
M3
neighbour table management and link quality estimation
refines
M4
route broadcast mechanism
refines
decomposes
M5
parent selection and unicast packet forwarding mechanism
CTL5
ENV5
MintRoute Protocol
Application Layer
Network Layer

Figure 3: Event-B Development Approach

node must have a route to a sink. Theorems @*mth3_1* and @*mth3_2* are introduced to help the proof of @*mth3_3*.

$$
\begin{aligned}
&\text{@inv3\_1 nodes} \subseteq \mathbf{ND}\\
&\text{@inv3\_2 cRouteTree} \in (\text{nodes} \setminus \{\mathbf{Sink}\}) \rightarrow \text{nodes}\\
&\text{@inv3\_3 completedRoute} = \text{TRUE} \Rightarrow \text{nodes} = \mathbf{ND}\\
&\text{@saf3\_1 } \forall S \cdot S \subseteq \text{cRouteTree}^{-1}[S] \Rightarrow S = \varnothing\\
&\textbf{theorem} \text{ @mth3\_1 } \forall T \cdot (\mathbf{Sink} \in T \land \text{cRouteTree}^{-1}[T] \subseteq T \Rightarrow \text{nodes} \subseteq T)\\
&\textbf{theorem} \text{ @mth3\_2 nodes} \subseteq \{\mathbf{Sink}\} \cup (\mathbf{tcl}(\text{cRouteTree}))^{-1}[\{\mathbf{Sink}\}]\\
&\textbf{theorem} \text{ @mth3\_3 nodes} \setminus \{\mathbf{Sink}\} \subseteq (\mathbf{tcl}(\text{cRouteTree}))^{-1}[\{\mathbf{Sink}\}]
\end{aligned}
$$

Figure 4: Safety invariants regarding the route tree in the fifth refinement model (*M5*).

5.3 Model Decomposition, Verification and Validation

As WSN is a distributed system in which each node exchanges data via a channel interface, shared event decomposition [But09, SPHB11] is used to separate *software controller* from *environment* as shown in Figure 5. The software controller contains the necessary variables and events expressing communication networking protocols and algorithms such as *SensorApp* and *MintRoute* whereas *environment* consists of variables and events regarding the connectivity, channel and sensing environment. We design these two subcomponents to exchange messages via shared events. The steps of communication can be described as follows:

(P1) The environment activates each sensor node in the controller to sense and create a data packet via a shared event (*sensing*).

(P2) Each sensor node transmits a data packet down to the channel. The shared event (*send_down*) passes the forwarder node id and packet information as channel parameters (*chnPar*).

(P3) The channel indicates the neighbour of the current forwarder.

(P4) The channel returns the neighbour list and the forwarded packet to every neighbour controller via the shared event (*send_up*) to indicate the operation transmitting a packet up to specific receivers.

(P5) Each neighbour node (including a sink, e.g. nodes 0,1,4,3 of sender node 2, Figure 5) receives a forwarded packet (*receive_pkt* and *sink_recv_pkt*) or detects a duplicated packet (*receive_dup_pkt*).

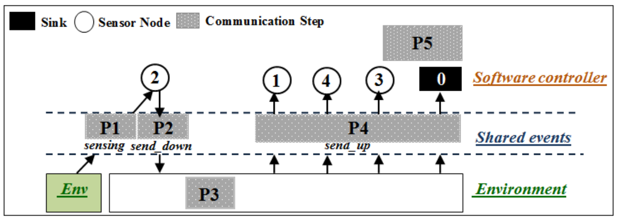


Figure 5: Steps of communication between a software controller and environment

Model Verification by POs: we verify the properties of a machine by proving that every event still satisfies invariants. 330 POs were discharged, in which 85 percent of the total number of POs were proved automatically by RODIN. This includes the invariant to guarantee that no packets are lost under the perfect network. Furthermore, the safety invariant regarding the absence of loop problem was discharged automatically. However, the remaining were proved interactively. This is because invariants and *tcl* properties include quantified predicates and graph properties.

Model Animation and Validation: we create the testing scenarios affecting the different link quality ratios that satisfy all desired requirements and strategies. These are used to animate and trace the list of operations to validate the formal model on ProB.

6 C-style Co-modelling

6.1 *FoCoSim-WSN* Framework for WSNs

We develop a prototype *FoCoSim-WSN* framework of node controller models on the Event-B simulation and sensor environment models on MiXiM simulation as shown in Figure 6. Event-B layers each communication protocol and algorithm to create and verify node controller models through refinement steps, whereas each protocol layer in MiXiM environments only contains gates (without any protocol algorithms) for communicating with the adjacent layer. In order to co-simulate these models, a master is implemented in Groovy language of the ProB tool. Here we describe the implementation of our master.

To schedule the event in the Event-B model, we implement multiple threads in the master. Each thread creates the instance of the Event-B model representing the controller of each sensor node which corresponds to each virtual node in the MiXiM Environment. MiXiM provides periodic timers such as a sensing timer and route broadcast timer, which allows the Event-B controller and MiXiM Environment models to exchange input/output periodically. TCP sockets are implemented as data exchange interfaces on both sides. We mock-up our own interfaces for MiXiM. *FMInterface* is the front-end interface which contains the synchronization event which corresponds to the shared-event defined in Event-B. This event is scheduled at fixed intervals and dedicated to maintain the data synchronization between the Event-B controller and MiXiM environment. Our protocol algorithms communicate with a sensor environment by exchanging the packet information (*chnPar*) and the receiving neighbour lists (*nbrLst*) together with the forwarded packet via the socket program implemented in the master. *SimManager* is the back-end interface where the parameters passed from *FMInterface* are transited down to/up from the channel (accessed by module *ChannelAccess*) via the protocol stack.

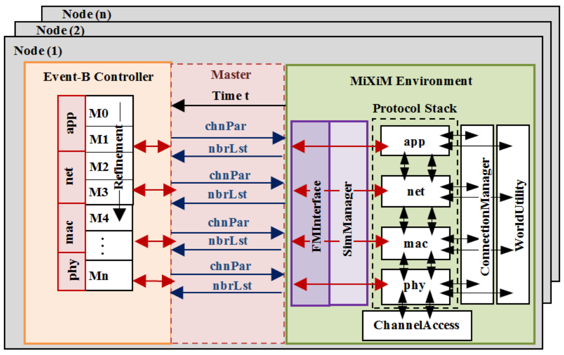


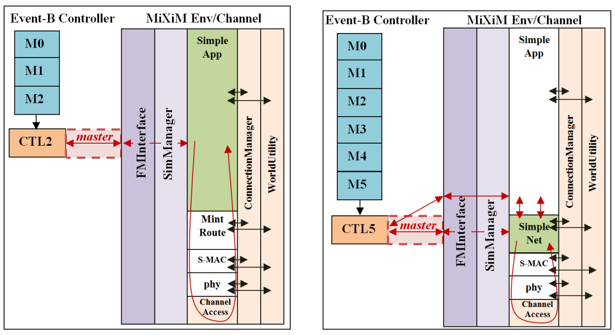
Figure 6: *FoCoSim-WSN* Co-simulation Framework

During the initialization phase, MiXiM initializes the physical environment such as interval time, ratio propagation, max transmission power and path loss coefficient alpha from the configuration file. Virtual nodes are created and placed on the simulation area (playground) generated by module *WorldUtility* in MixiM. The connection between them is established by method *updateNicConnection()* in MiXiM's *ConnectionManager*. Only nodes that are placed within the maximal interference distance of each other can be connected. This forms the network topology. Then, *FMInterface* starts requesting a connection to the Event-B controller via sockets implemented in the master. Once the connection is established, *FMInterface* sends the necessary information extracted from the configuration file such as the number of nodes (*numNodes*) and identified sink id (*SinkId*) to a master. The master creates multiple threads corresponding to the number of virtual nodes generated in the MiXiM environment before it starts creating, initialising and loading the instance of Event-B controller model into each thread. Furthermore, each thread contains a *TimerTask* which is used to schedule the sequence of the events in the Event-B model. Then, the master relays the completion of the Event-B model initialisation to *FMInterface* in MiXiM via a socket. The example of the above mentioned initialization phase can be expressed in Figure 8.

At the simulation phase, the MiXiM begins to simulate at time 0. Every periodic time, MiXiM sends the information to tell each controller node to start performing the operation such as sensing and creating a packet. MiXiM generates sense data to the required random distribution for each node controller. When each node creates a packet, the channel information (*chnPar*) including the initial source of packet, the sequence number, the forwarder and the destination node id (setting to -1 for broadcasting mechanism) is synchronized between the output shared event in Event-B controller (e.g. event *send_down*, Figure 5) and the synchronization event in *FMInterface*. The virtual packet containing this correspondent input parameter is created and transmitted to the virtual neighbour node via the protocol stack. The receiving neighbour lists in MiXiM of the forwarder node are synchronized back from the synchronization event in *FMinterface* to the input shared event in Event-B controller (e.g. event *send_up*, Figure 5) of each thread of node controller. After receiving the neighbour lists, each node controller will perform the next operation (e.g the receiving events described in step P5, Figure 5). Then, the receiving node that is not the destination forwards/rebroadcasts a received packet to its neighbours. The steps of this communication are the same as described for packet creation and transmission mechanism.

6.2 Co-simulation Case Study Modelling

This section describes how to use our general *FoCoSim-WSN* framework to implement the specific models in our case study.



(a) *SensorApp* co-models (b) *MintRoute* co-models

Figure 7: Co-simulation models.

The C-style model for mixed (Event-B/MiXiM) co-simulation is developed by reuse of F-style formal-based and S-style simulation-based developments. Our prototype co-simulation framework exercises two levels of abstraction - application and network layers as shown in Figure 7. We reuse node controller models, M2(*SensorApp*) and M5(*MintRoute*), in F-style development to implement in this mixed co-simulation. In the MiXiM environment, we develop two protocol layers, *SimpleApp* and *SimpleNet* by removing the protocol algorithms from *SensorApp* and *MintRoute* modules in S-style. These modules are a standard module extending the base modules in MiXiM containing only packet en/decapsulation functions and gates. This enables the corresponding Event-B controller representing the upper layer model to be able to communicate with the lower layer in MiXiM environment via *FMInterface* and *SimManager*.

To demonstrate the iterative co-model development for each layer protocol, we start exercising at the application layer in which only the *SensorApp* algorithm is separated from the simulation environment and implemented in Event-B. In MiXiM, module *SimpleApp* is used as a gate for communicating between *SensorApp* controller in Event-B and lower layers retaining protocol algorithms *MintRoute* and *S-MAC* as illustrated in Figure 7a. Figure 7b shows the co-simulation at the network layer, the co-models are implemented in the same way as in the application layer. Module *SimpleNet* in MiXiM is used to coordinate between the *MintRoute* controller in Event-B and the lower layer. S-MAC is still retained in MiXiM's MAC layer. As in this Event-B controller also contains the concrete model of *SensorApp* protocol containing unicast data packet forwarding mechanism via the route, this model also co-simulates with module *SimpleApp* in MiXiM.

Figure 8 shows the master algorithm interaction expanding on Figure 7a to demonstrate one sensing cycle for *SensorApp*. This interaction corresponds to communication steps in Figure 5. As we use the same Event-B controller, we design steps R1, R2 and R4 to be the same as steps P1, P2 and P4. The input/output parameters are passed from/to shared event (*send_down*/*send_up*) in order to exchange information between the controller and environment. Then, these parameters are passed by a socket sending and receiving mechanisms (methods *send* and *recv*) as shown in steps R2 and R4. Step R5 is implemented for a receiving node to receive the forwarded data

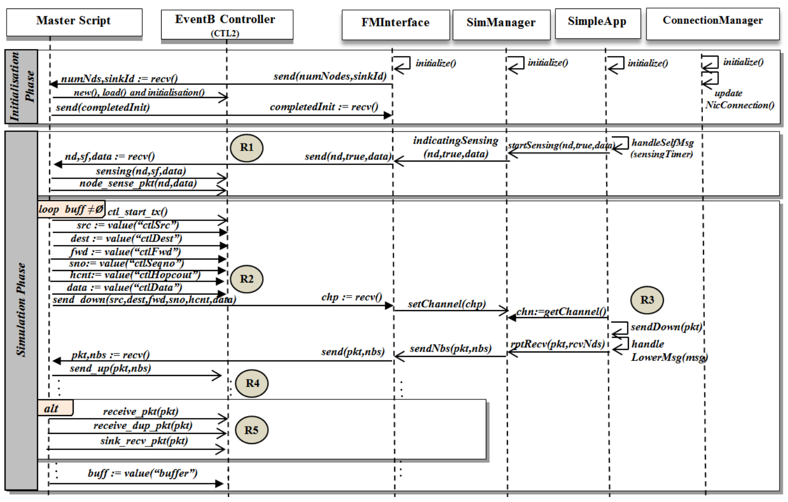
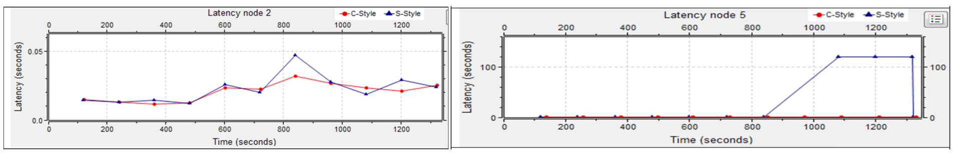


Figure 8: Master algorithm for *SensorApp* protocol co-simulation



(a) (b)

Figure 9: Example of performance analysis results of some nodes

packet, method *anyEvent* provided by ProB Groovy is used to create alternative choices among events *receive_pkt*, *receive_dup_pkt* or *sink_recv_pkt*. Note that we do not describe the master algorithm for the *MintRoute* protocol. This is because it has the co-simulation interaction step to exchange the same input/output parameters for beacon and route packet transmission.

Considering step R3 which performs concrete operations of step P3 in Figure 5, after each node in MiXiM environment receives channel information from *FMInterface* via *SimManager*, the packet is generated based on this information and transmitted down to lower layers and finally a channel. Note that virtual nodes in the MiXiM environment are generated by module *SimpleApp* (the same as *SimpleNet* for the network layer) containing only gates for transmitting/receiving a data packet to/from lower layers. The forwarding node uses method *sendDown* to relay a current packet down to the lower layer and finally to the channel. Method *handleLowerMsg* is used by receiving nodes for receiving a forwarded packet from the lower layer. *SIMManager* collects information from receiving nodes to generate neighbour list. Finally, this neighbour list is sent to *FMInterface* and relayed to the Event-B controller via the master.

6.3 Co-simulation Prototype Validation

In order to validate our C-style prototype, we ran our C-style co-models within 1400 seconds of simulation time with different network topologies varying from 1-hop to 3-hop network containing 4 to 7 nodes respectively. These simulation networks were generated from the parameters configured in the MiXiM configuration file. This performs eleven cycles of sensing periods, together with three cycles of parent selection. We compared the performance analysis results in C-style with that of S-style to identify the difference and weakness of C-style and S-style. However, we only found the randomness of the routing algorithm to discover the route tree. Figure 9, for example, shows one of our co-simulation results from the network containing the topology illustrated in Figure 2a. As shown in Figure 9a for both styles, the latency of node 2 is always low as it always chooses a sink to be a parent. However, in Figure 9b, at two-thirds of running simulation time, the latency of node 5 in S-style simulation is considerably high. This is because its parent had the highest number of forwarded data packets compared to that of C-style.

6.4 Engineering Process for C-style Development

Based on our experience, our prototype *FoCoSim-WSN* enables three flexible modes of working in which formal and simulation can be integrated easily.

This can start from the pure Event-B model (F-style) development in which the protocol algorithm can be modelled through the refinement steps. System engineers can develop either an early model for some high level or a refined model for more detailed level of functional abstraction. Then, the software controller illustrating the protocol algorithm can be separated from a sensor environment by using the decomposition technique. This separated controller is also prepared for C-style co-simulation.

On the other hand, the developed controller algorithm in pure S-style simulation model can be separated from the environment and implemented in the Event-B model in the same way. The separated environment, which will be used by C-style co-simulation, still retains only the communication gates (no algorithm) together with the standard function in MiXiM such as packet encapsulation and decapsulation. However, instead of developing the sensor environment from scratch, reusing our standard module for communicating between two upper layers (such as *SimpleApp* and *SimpleNet*) is another alternative way to prepare simulation environment.

The next step is C-style co-simulation in *FoCoSim-WSN* framework. The Event-B controller model from F-style and the separated sensor environment from S-style are co-simulated via a master. To support the development with reuse, our master is modularized into the different interfaces such as *WSNSocket*, *TimerTask* and *EventBCtl* for encapsulating the functionality of the socket program, scheduling events for multiple threads and Event-B controller model interface respectively. In order to develop a specific master algorithm, the system engineers only customize the *EventBCtl* to be compatible with their Event-B model (as we have done for our proposed co-simulation models for different layers). During implementing a master algorithm, the synchronization (shared) events for communicating between these two models are needed to be identified. However, the automatic master code generation is still left for future work. This work has accomplished the generic modules for only two upper layers: application and network. The lower standard modules for the lower layer such as MAC is still required and is work in progress.

7 Conclusion and Future Work

This paper has demonstrated the co-simulation approach to the extension of current SE for WSN development. Our prototype development shows that the framework integration for (F)ormal, (S)imulation and (C)osimulation can combine the benefits between two modelling approaches.

The complexity to manage in communication protocol and algorithm development can be reduced by the refinement approach provided by Event-B. Furthermore, Event-B offers strong V&V in which the absence of certain classes of faults such as the loop problem in the route tree can be guaranteed by POs. Whereas a stochastic environment framework of wireless channel, radio and analogue models provided by simulation can help engineers to analyse and evaluate the performance of the network such as network latency and congestion. Our prototype *FoCoSim-WSN* framework provides an iterative interworking scheme through multiple refinement levels. System engineers also can work either F- or S-style development before the separated controller and environment are combined and co-simulated into our prototype framework. This framework can be flexible and utilized to integrate between F-, S- and C-style modelling. System engineers can implement their models cross over into these three flexible modes of working easily.

In the future, we will perform the experimentation of this preliminary prototype framework through long running testing scenarios. The real network deployment problem will be addressed by this prototype framework as expressed in [IPM13]. Node failure, unreliable connection and buffer overflow scenarios will be injected into our co-simulation. The bottleneck problem will be tackled by limiting the queue size through long running simulation. “Killer” traces will be seeked to validate formal Event-B models. More dense network models will be considered in order to evaluate the reliability of this framework and explore the network congestion problem. Code generation experiments from the node controller for the real node is still promising for our future work. Open research issues are the node level co-simulation development and the extension for multi-cosimulation.

Bibliography

[ABHV06] J.-R. Abrial, M. J. Butler, S. Hallerstede, L. Voisin. An Open Extensible Tool Environment for Event-B. In *ICFEM*. Pp. 588–605. 2006.

[Abr07] J.-R. Abrial. Formal Methods : Theory Becoming Practice. *Journal of Universal Computer Science* 13(5):619–628, 2007.

[Abr10] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, NY, USA, 1st edition, 2010.

[ACB10] M. Allen, G. Challen, J. Brusey. Designing for Deployment. In Guara et al. (eds.), *Wireless Sensor Networks*. 2010.

[ADV13] ADVANCE. ADVANCE Deliverable D4.2 (Issue 2): Methods and tools for simulation and testing I. Technical report, March 2013.
http://www.advance-ict.eu/sites/www.advance-ict.eu/files/AdvanceD4.2-issue2.pdf

[BIS$^+$08] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, M. Parlange. SensorScope: Out-of-the-Box Environmental Monitoring. In *IPSN*. Pp. 332–343. 2008.

[BKL$^+$10] J. F. Broenink, C. Kleijn, P. G. Larsen, D. Jovanovic, M. Verhoef, K. Pierce. Design Support and Tooling for Dependable Embedded Control Software. In *SERENE*. Pp. 77–82. 2010.

[BRWR10] J. Beutel, K. Roemer, M. Woehrle, M. Ringwald. Deployment Techniques for Sensor Networks. In *Sensor Networks: Where Theory Meets Practice*. Pp. 219–248. 2010.

[But09] M. Butler. Decomposition Structures for Event-B. In *IFM*. Pp. 20–38. 2009.

[DBA08] K. Damchoom, M. J. Butler, J.-R. Abrial. Modelling and Proof of a Tree-Structured File System in Event-B and Rodin. In *ICFEM*. Pp. 25–44. 2008.

[DBMS13] A. Didioui, C. Bernier, D. Morche, O. Sentieys. HarvWSNet:A co-simulation framework for energy harvesting wireless sensor networks. In *ICNC*. Pp. 808–812. 2013.

[HKBA09] T. S. Hoang, H. Kuruma, D. A. Basin, J.-R. Abrial. Developing topology discovery in Event-B. *Sci. Comput. Program.* 74(11-12):879–899, 2009.

[HSB10] O. Heimlich, R. Sailer, L. Budzisz. NMLab: A Co-simulation Framework for Matlab and NS-2. *SIMUL* 0:152–157, 2010.

[IPM13] A. Intana, M. R. Poppleton, G. V. Merrett. Adding Value to WSN Simulation through Formal Modelling and Analysis. In *SESENA '13*. Pp. 24–29. IEEE, 2013.

[ISH10] M. Imran, A. M. Said, H. Hasbullah. A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks. In *ITSim*. Pp. 897–902. 2010.

[KM07] W. Kiess, M. Mauve. A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Netw.* 5(3):324–339, Apr. 2007.

[KSW$^+$08] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, S. Valentin. Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In *Simutools*. Pp. 71:1–71:8. 2008.

[MRDD10] P. Matouek, O. Ryav, G. S. De, M. Danko. Combination of Simulation and Formal Methods to Analyse Network Survivability. In *SIMUTools*. P. 6. 2010.

[Pie10] G. P. Pietro. Software engineering and wireless sensor networks: happy marriage or consensual divorce? In *FoSer*. Volume 4, pp. 283–286. 2010.

[PM12] M. Poppleton, G. Merrett. Towards a Principled and Evolvable Approach to Software Development for Future Wireless Sensor Networks. In *SESENA*. 2012.

[SGD11] C. Sommer, R. German, F. Dressler. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE TMC* 10(1):3–15, 2011.

[SPHB11] R. Silva, C. Pascal, T. S. Hoang, M. Butler. Decomposition tool for Event-B. *SPE* 41(2):199–208, 2011.

[WB12] B. Wang, J. S. Baras. Integrated Modeling and Simulation Framework for Wireless Sensor Networks. *WETICT* 0:268–273, 2012.

[WB13] B. Wang, J. S. Baras. HybridSim: A Modeling and Co-simulation Toolchain for Cyber-physical Systems. *DS-RT '13* 0:33–40, 2013.

[WBLS09] A. Wang, P. Basu, B. T. Loo, O. Sokolsky. Declarative Network Verification. In *PADL '09*. Pp. 61–75. 2009.

[WTC03] A. Woo, T. Tong, D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*. Pp. 14–27. 2003.