Design Flow for the Rapid Development of Distributed Sensor Network Applications

Alexios Lekidis, Paraskevas Bourgos, Simplice Djoko-Djoko, Marius Bozga, Saddek Bensalem

Verimag Research Report n^o TR-2014-13

October 8, 2014

Design Flow for the Rapid Development of Distributed Sensor Network Applications

Alexios Lekidis, Paraskevas Bourgos, Simplice Djoko-Djoko, Marius Bozga, Saddek Bensalem

October 8, 2014

Abstract

The exponential increase in the demands for the deployment of large-scale sensor networks, makes necessary the efficient development of functional applications. Nevertheless, the existence of scarce resources and the derived application complexity, impose significant issues and require high design expertise. Consequently, the probability of discovering design errors once the application is implemented is considerably high. To address these constraints there is a need for the availability of early-stage validation, performance evaluation and rapid prototyping techniques at design time. In this paper we present a novel approach for the co-design of mixed software/hardware applications for distributed sensor network systems. This approach uses BIP, a formal framework facilitating modeling, analysis and implementation of embedded real-time, heterogeneous, component-based systems. Our approach is illustrated through the modeling and deployment of a Wireless Multimedia Sensor Network (WMSN) application. We emphasize on its merits, notably validation of functional and non-functional requirements through statistical model-checking and automatic code generation for sensor network platforms.

Keywords: Wireless Sensor Networks, Model-based development, Multimedia Communication, Clock synchronization, Performance evaluation

Reviewers: Marius Bozga

How to cite this report:

```
@techreport {TR-2014-13,
   title = {Design Flow for the Rapid Development of Distributed Sensor Network Applications},
   author = {Alexios Lekidis, Paraskevas Bourgos, Simplice Djoko-Djoko, Marius Bozga, Saddek Bensalem},
   institution = {{Verimag} Research Report},
   number = {TR-2014-13},
   year = {}
}
```

Contents

1 Introduction

The introduction of sensor networks in various application fields nowadays has been a significant technological advance. Such fields include health-care, transportation, agriculture, environmental monitoring, security systems, high-energy physics, industrial process control, factory and building automation and more. The applications of distributed sensor networks are broad due to the unique characteristics of the sensor devices, from which they are composed. Each sensor is a tiny, low-cost, low-power, energy harvesting, multifunctional device. Being usually deployed in a large-scale distributed environment, it needs to configure itself automatically, in order to collect, process and send information to a central processing unit, called base station or sink. The transmission is handled by the underlying network, which can be either wired or wireless. The use of wireless networks is often preferred over wired, due to the derived limitations from the cost of wiring.

The development of functional applications, ensuring the several benefits of sensor networks, is however extremely challenging. This is due to their scarce resources, imposing constraints such as the limitations in the communication cost, the energy consumption, the memory usage and the achievable network bandwidth. These limitations are enhanced as they are usually deployed in inaccessible or distant areas (e.g mountains, forests) and thus cannot be frequently changed in case of a failure. In addition, specific applications have strict timing constraints for data handling, which may not be guaranteed due to the influence of the communication and data processing latencies. Equally important is to consider that design errors in the final application development stage are highly probable, even if there is detailed knowledge of the application area and the hardware platforms. Moreover, if an error is observed at that stage, the debugging is extremely hard and time-consuming.

To address these challenges we propose a model-based design approach, in order to express the behavior and functionality of such applications. A model-based framework improves the quality, the modularity and reusability of the developed software artifacts. It can further allow separation of concerns, in order to describe software and hardware architecture at a certain level of abstraction. Thus, any change within the application results only in the modification of the software architecture. Furthermore, validation and verification are enabled in every development stage. The overall contribution of this work is the construction of a full-fledged design flow, based on a single semantic framework (BIP [2]), facilitating the rapid development of correct and functional sensor network applications. This flow supports application and system modeling, validation of functional correctness and performance analysis on system models. It also permits automatic code generation in distributed sensor network platforms, leading to a significant reduction in the development time and errors of a manual implementation.

The paper is organized as follows. Section 2 provides a brief introduction to the area and the current challenges of distributed sensor network applications. Section 3 presents the proposed design flow and details on its key steps. Section 4 illustrates its use in a concrete WMSN application and Section 5 provides conclusions and perspectives for future work.

2 Sensor network applications

A major design factor in the development of sensor network applications is the communication, in order to exchange sensed data. As each network node is a resource-constrained device, the developed applications should have low bandwidth demands and tolerance to the communication latencies. Recently, the significant size reduction of inexpensive hardware, such as microphones and cameras, made possible the addition of audio and video capabilities for multimedia applications on a sensor network environment [16]. The development of such applications is mainly based on the increasingly popular lightweight versions of Linux, often referred to as embedded Linux [11]. This is due to their open-source environment and the support of several off-the-shelf platforms. Multimedia sensor network applications have strict timing constraints for data delivery and are extremely demanding in terms of memory and storage. The latter make necessary the usage of compression algorithms. An example of such an application deployed over a wireless network for audio streaming and synchronization of the local sensors clocks is provided in Figure 1.

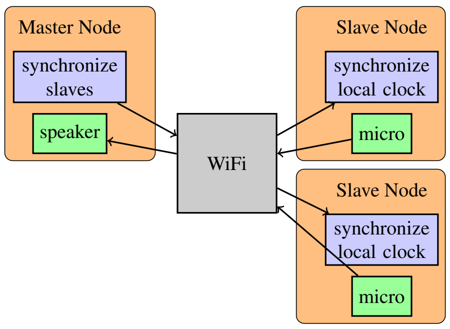


Figure 1: WMSN Application example

The main arising challenge in the successful development of correct and functional distributed sensor network applications is to provide productive and efficient design solutions ensuring the three following goals:

The addressing of functional and non-functional requirements. This goal focuses in the ability to identify and the methods to evaluate these requirements at design time ([6]). On the one hand, non-functional requirements concern the optimal exploitation of the available hardware resources. This is accomplished by limiting the communication cost, memory usage and energy consumption as well as reducing the resource failure rate. A first example for such a requirement are the delays imposed by the processing time or the communication latency, which may lead to the reception of outdated sensor data. As an outcome, adverse actions may be triggered in the network. Secondly, is the network connectivity, determining the packet delivery ratio, that is, the percentage of successfully received packets by the total packets transmitted in the network. On the other hand, functional requirements concern the correctness and performance of the application. More specifically, they aim on managing buffer utilization, improving the efficiency of the compression algorithms for the multimedia and providing strict time guarantees for data handling. It should be also noted that in some situations non-functional can affect the functional requirements, as depicted by the strong influence of the communication latency and the packet delivery ratio to the buffer utilization.

The synchronization of the local sensors clocks (clock synchronization). In many applications, the exchanged data need to be accurately timestamped, in order to be further processed. Nonetheless, this poses a serious application development problem, as the construction of a common time reference in a distributed system is hard to achieve. The common time reference can be also used to measure the duration between two events occurring in different nodes, whose clocks can drift or become desynchronized over time. Several solutions to this problem were proposed, in order to obtain a global time reference in the system. The commonly obtained synchronization accuracy is considered to be in the microsecond scale. A traditionally adopted solution is the Network Time Protocol (NTP) [23], which nevertheless requires increased computational power and storage memory, since it uses extra messages to calculate the Round Trip Delay (RTD). Additionally, the use of several trials to compute the average RTD results in less accuracy, further overhead and thus is suitable only for applications with low precision demands. A better protocol, also relying on the RTD calculation, which achieves high synchronization accuracy in both wired and wireless sensor networks, is the Precision Time Protocol (PTP) [14]. However, the derived hardware enhancements (as in [15]) introduced to achieve microsecond accuracy may not be available in lightweight and resource-constrained environments. A new family of protocols for software-based clock synchronization is derived from the application of the Kalman filter algorithm [8]. Compared to the other synchronization protocols, this family does not require the interaction or the development of dedicated drivers to access the hardware, since it is operating in the application level. The underlying Kalman filter algorithm relies on tracking the advance of a reference clock and automatically adapting to it. The synchronization method used by this family is different from the above protocols, since it does not rely on the RTD calculation.

Tools for application development and code debugging. As multimedia sensor network applications require the dense deployment of the small-scaled sensors, the communication latencies and the conflicts occurring in the protocol stack are unpredictable. Therefore, the probability of having design errors in the final development stage is extremely high. This situation may arise even if the developer has complete knowledge of the application as well as the underlying hardware architecture. Moreover, the debugging techniques at that stage are extremely hard and time consuming, even for experts. Consequently, an error may possibly lead to a new system implementation. This happens due to the absence of separation of concerns, such that the application is developed independently from the hardware architecture. In this scope, a developer has to specify and build separate artifacts for the software and the hardware architecture, which could also be reused in latter applications. Then, he should be able to define the optimal methodology for the deployment of the application on the given architecture, such that it functions properly. This procedure is called as mapping [5].

Meeting all the aforementioned goals, is extremely demanding. A starting point to this challenge would be the availability of simulation and validation tools in the early development stage, such that the system is validated beforehand and the design goals are ensured. Previous work in this scope is mainly divided in three categories. The first category uses the Mathwork's tools for modeling, simulation and automatic code generation targeting specific sensor network operating systems [17] [18]. These tools are well known due to their vast variety of libraries, however they are not able to address functional and non-functional system requirements. Secondly, the metamodeling frameworks addressing such requirements use the UML tools to model and the Eclipse platform to generate code for sensor network applications [21]. Though certain developed frameworks ([1]) are also able to validate them, they do not focus on clock synchronization and the generated code is usually not complete. Finally, formal modeling approaches for such applications provide validation support for functional and non-functional requirements [22] [25] [12] as well as clock synchronization [9], but do not implement tools for automatic code generation. Therefore, as far knowledge is concerned, the existing work is not considering all the above design goals simultaneously. To this extent, in the following section we propose a novel method for the systematic development of distributed sensor network applications, enabling separation of concerns and targeting all their design goals.

3 Design Flow

In this section, we propose a novel approach for building sensor network applications. This approach is based on a design flow, which leads to a framework for 1) the construction of a faithful sensor network system model for analysis as well as performance evaluation and 2) the generation of deployable code for applications in the domain of sensor networks. The design flow is based on the BIP framework described below.

The BIP – Behavior / Interaction / Priority – framework [2] is aiming at design and analysis of complex, heterogeneous embedded applications. BIP is a highly expressive, component-based framework with rigorous semantic basis. It allows the construction of complex, hierarchically structured models from atomic components characterized by their behavior and interfaces. Such components are transition systems enriched with data. Transitions are used to move from a source to a destination location. Each time a transition is taken, component data (variables) may be assigned new values, computed by user-defined functions (in C/C++). Atomic components are composed by layered application of interactions and priorities. Interactions express synchronization constraints and define the transfer of data between the interacting components. Priorities are used to filter amongst possible interactions and to steer system evolution so as to meet performance requirements e.g., to express scheduling policies. A set of atomic components can be composed into a generic compound component by the successive application of connectors and priorities.

BIP is supported by a rich toolset [1] which includes tools for checking correctness, for source-to-source transformations and for code generation.

Example 1 *Figure 2 shows a graphical representation of one atomic component in BIP, which models the behavior of the PLL process (presented in Section 3.1). The behavior of PLL is described as a transition system with control locations* idle, recvMsg, process *and* sndRes. *It is responsible for the reception of synchronization frames through the* CLK_RECV *port. It subsequently moves from the* idle *to the* recvMsg *state. After an interaction through the port* LOCAL_CLK*, it calculates a software clock through the internal port* update *and returns to the initial (*idle*) state.* CLK_REQ *port is used to receive requests for calculating the local clock. The value of the local clock is calculated at the internal transition* prepare *and is exported through port* CLK_RES.

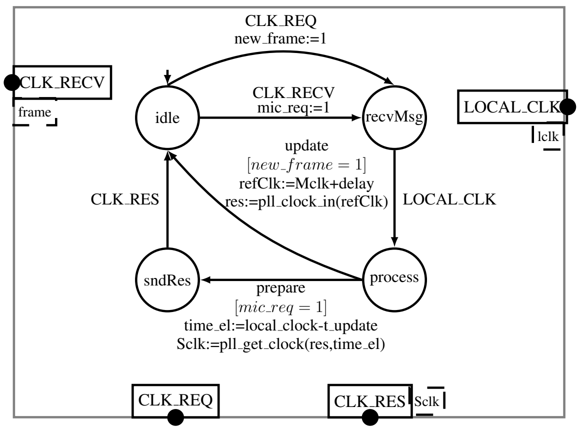


Figure 2: PLL component

A statistical method was recently proposed to handle scalability issues present in numerical methods that are classically used to check stochastic systems. This novel technique is called Statistical Model Checking (SMC) [26] [10]. It requires, as in classical model checking, building an operational formal model of the system to verify and to provide a formal specification of the property to check, generally using temporal logic. The BIP framework is extended to allow stochastic modeling and statistical verification [4]. On the one hand, the method relies on BIP expressiveness to handle heterogeneous and complex component-based systems. On the other hand it uses statistical model checking techniques to perform quantitative verification targeting non-functional properties.

The BIP design flow, illustrated in Figure 3, uses PPM specifications, thoroughly described in Section 3.1, as a re-targetable input model to: (1) automatically generate a sensor network system model in BIP and (2) automatically generate the code for execution on the target distributed sensor network platform. The proposed flow is used to evaluate both functional, non-functional and clock synchronization requirements of sensor network applications. To achieve that, on the one hand, we apply SMC on the *system*

[1]http://www-verimag.imag.fr/tools

model in BIP and on the other hand, we execute the generated code on the target sensor network platform. It is important to mention that the two paths, meaning the construction of the *system model in BIP* and the *generation of executable code* are consistent between each other. This is accomplished because, first, both approaches integrally preserve the behavior of the input application software and, second, the *Sensor Network Components in BIP* faithfully model the target sensor network.

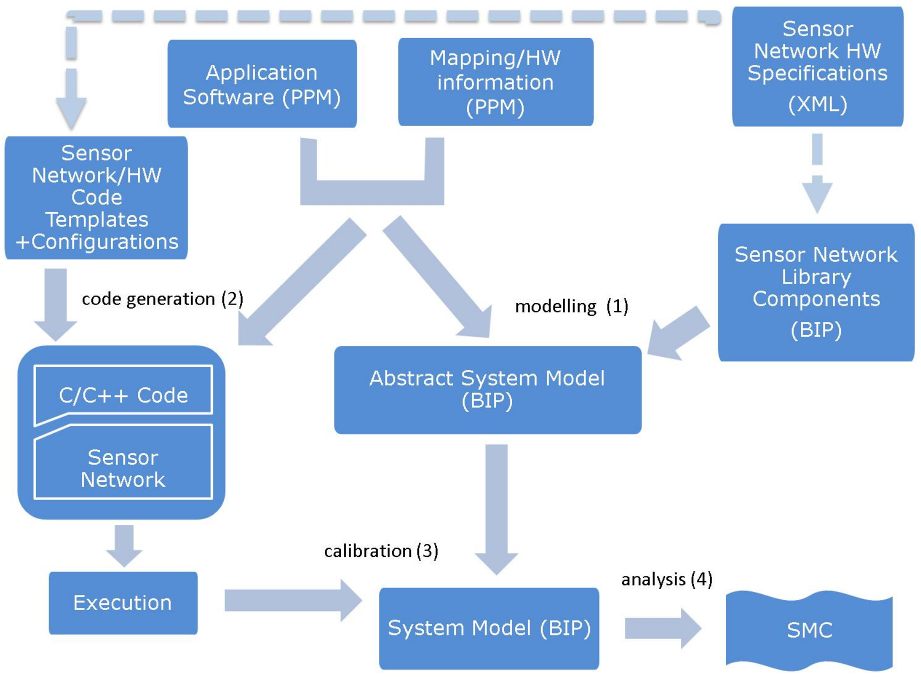

Application Software (PPM)
Mapping/HW information (PPM)
Sensor Network HW Specifications (XML)
Sensor Network/HW Code Templates +Configurations
code generation (2)
modelling (1)
Sensor Network Library Components (BIP)
C/C++ Code
Sensor Network
Abstract System Model (BIP)
calibration (3)
analysis (4)
Execution
System Model (BIP)
SMC

Figure 3: Overview of the proposed Design Flow

The proposed design flow proceeds in four main steps:

1. The construction of an *abstract system model*. This model represents the behavior of the application software running on the hardware platform according to the mapping, but without including all hardware dependent (e.g. execution times, data processing delays) and network-specific information (e.g. packet delivery ratio, end-to-end delays).

2. The *generation of executable code* that is deployed on the physical hardware platform. This is performed by initially transforming the input hardware specifications into code templates. Once these templates are fully constructed by the user, they can be reused for any sensor network application. They are accordingly parametrized, using node configuration files, in order to automatically generate the executable code.

3. The construction of the *system model in BIP* by injecting all the missing hardware dependent information to the previously generated *abstract system model*.

4. The performance analysis on the calibrated *system model in BIP* with the use of Statistical Model Checking (SMC) that performs quantitative verification targeting functional and non-functional requirements. The results are used as a feedback to the user to propose enhancements in the design.

3.1 Pragmatic Programming Model

The Pragmatic Programming Model (PPM) is a description language developed to provide a simple and convenient way for describing highly-parallel applications expressed as networks of communicating processes. The language has been inspired by DOL (Distributed Operation Layer) [24], which is a framework

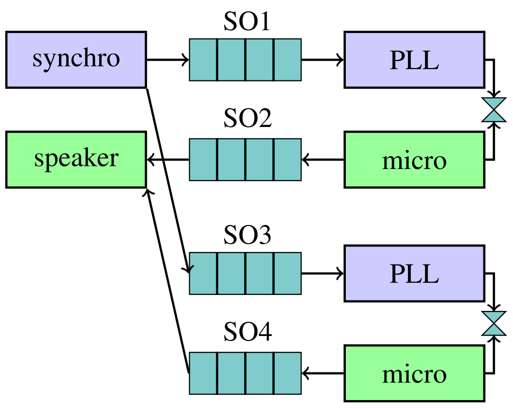


Figure 4: WMSN Application PPM Model

devoted to the specification as well as the analysis of mixed software/hardware systems and provides a Kahn Process Network (KPN) model of the application.

In PPM, application software is defined using a process network model. It consists of a set of deterministic, sequential processes communicating asynchronously through shared objects, such as FIFOs, shared memories and mutexed locations. The mapping associates application software components to devices of the hardware platform, that is, processes to processors and shared objects to remote communication media. Specifications of the latter, including communication interface and protocols, are also described in the mapping to provide all the necessary details for the code generation and the construction of the system model.

WMSN Application

In Figure 4 we present an WMSN application in PPM, referring the application described in Section 4. It consists of 1) one clock synchronization process *synchro*, sending out synchronization data through the FIFOs (*SO1*, *SO3*), and 2) two audio capturing processes *micro*, sending out audio data, through the FIFOs (*SO2*, *SO4*). The synchronization data are received by two processes *PLL* (implementing the clock synchronization protocol) and the audio data by an audio reproduction process *speaker*.

Application Software in PPM

The application software in PPM consists of three basic entities: *Processes*, *Shared Objects*, and *Connections*. The network structure is described in XML. Each *Process* has input, output ports and sequential behavior. Processes communicate by using *shared objects*. Each *shared objects* has input and output ports, uniquely associated with ports of processes.

In Figure 5, we present a fragment of the XML specification of the WMSN application described below. It consists of processes, shared objects and connections. In Figure 5, we depict the *PLL* process. For each process, we specify the name of the process, the number of input and output ports, the names of the ports, the respective types and the location of the source C code describing the process behavior. For each shared object (i.e FIFO) we specify the name, the type the maximum capacity of data and the input and output port. Finally, we define the connections between the processes and the shared objects by specifying the input and output ports which contribute in each connection.

Process behavior is described using sequential C programs with a particular structure (see Figure 6 for a concrete example). For a cyclic process as *P*, its state is defined as an arbitrary C data structure named *P_state* and its behavior as the program:

$$P\_init(); while(true) P\_fire();$$

where *P_init(), P_fire()* are arbitrary functions operating on the process state. The initial call of the *P_init()* function is followed by an endless loop calling the *P_fire()* function. Communication is realized by using two particular primitives, namely *write* and *read* for respectively sending and receiving data to shared

```
<header lang="c" file="global.h"/>

<process name="pll" process-class="WhileFire">
<port name="out" peer-class="FIFO" peer-name="in"/>
<header lang="c" file="pll_state.h" x-state="true"/>
<header lang="c" file="pll.h"/>
<source lang="c" file="pll.c"/>
<source lang="c" file="SPM_clock.c" libs="-lblas -lm -lrt"/>
</process>
...
<shared-object name="SO1" object-class="FIFO" size="4" item-size="64">
<port name="in"/>
<port name="out"/>
</shared-object>
...
<connection>
<port-ref node="SO1" port="out"/>
<port-ref node="pll" port="in"/>
</connection>
...
```

Figure 5: WMSN Application XML Description

objects. A *read* operation reads data from an input port, and a *write* operation writes data to an output port. Moreover, the *P_fire()* method may invoke a *detach* primitive in order to terminate the execution of the process.

Example 2 *The description of a PLL process is shown in Figure 6. It defines the function pll_init() to initialize the process state and the function pll_fire() to describe the cyclic behavior of the process. PLL process receives data from the process network using the $FIFO\_read()$ function and the rest of the code implements the synchronization algorithm ($pll\_clock\_in()$ function).*

```
#include "pll_process.h"
void pll_init(pll_process *p) {
   (p->local->pll).stream_size = 1;
   (p->local->pll).block_size = (unsigned int) sizeof(clockOut_t);
   (p->local->pll).data_in = malloc((p->local->pll).block_size);
    p->local->data_size = (p->local->pll).block_size;
}
int pll_fire(pll_process *p) {
   FIFO_read(p->in, (p->local->pll).data_in, (p->local->pll).block_size);
   gettimeofday ( &(p->local->slave_time), NULL );
   uint64_t slave_clock = ( ( uint64_t ) p->local->slave_time.tv_sec * \
     ( uint64_t ) 1000000 ) + ( uint64_t ) p->local->slave_time.tv_usec;
   clockOut_t* master_frameClock = ( clockOut_t* ) (p->local->pll).data_in;

   master_clock = master_frameClock->time;
   pll_clock_in ( slave_clock, master_clock, p->local->argument);

   return 0;
}
```

Figure 6: PLL Process Code Description

Application Mapping on the Platform

The deployment of the use case applications on the target platform is specified with the use of a mapping XML description file, as presented in Figure 7. The application processes ("app-node" in XML) are bound to a hardware platform node ("hw-element" in XML). The binding ("deployment" in XML) includes additional information, concerning the hardware platform ("hw-property"), that are necessary for the configuration for establishing communication between the network nodes. This information includes the network interface name, the IP addresses of the destination network node, the port specification and the type of communication used (unicast, multicast and broadcast). The "communication protocol" globally used and "extra" process properties ("app-property") are specified in separate XML elements.

Example 3 *The description of the mapping XML file of the WMSN application is shown in Figure 7. The first "deployment" element specifies that the PLL process is deployed on the "udoo" hardware node using "wlan0" as network interface, "10.0.0.14" as destination IP address and 375, 250 as origin and target port respectively. The second "deployment" binds the synchro process to a second "udoo" hardware node. The use of the UDP communication protocol is defined next, followed by extra application properties such as the clock synchronization periods.*

```xml
<deployment>
  <app-node name="pll"/>
  <hw-element name="node" hw-class="udoo" index="0"/>
  <hw-property name="networkInterface" hw-class="node-inter" value="wlan0"/>
  <hw-property name="srcPort" hw-class="node-srcPort" value="375"/>
  <hw-property name="dstPort" hw-class="node-dstPort" value="250"/>
  <hw-property name="dstIP" hw-class="node-dstIP" value="10.0.0.14"/>
</deployment>
<deployment>
  <app-node name="synchro"/>
  <hw-element name="node" hw-class="udoo" index="1"/>
  <hw-property name="networkInterface" hw-class="node-networkInterface" value="wlan0"/>
  <hw-property name="srcPort" hw-class="node-srcPort" value="250"/>
  <hw-property name="multiIP" hw-class="node-multiIP" value="10.0.0.255"/>
  <hw-property name="broadcast" hw-class="node-broadcast" value="0"/>
</deployment>
...
<communication protocol="udp"/>
...
<extra>
   <app-property app-name="synchro" property-name="period" value="1"/>
</extra>
```

Figure 7: WMSN Application Mapping XML Description

3.2 System model in BIP

In our design flow, we construct the *system model in BIP* to faithfully represent the behavior of the application running on the underlying hardware and network. The construction proceeds in two steps, as presented in the design flow. The first step is the construction of the intermediate *abstract system model in BIP* and the second step is the construction of the complete *system model in BIP*.

The *abstract system model in BIP* is constructed in several steps. Firstly, the *application software model in BIP* is constructed by performing transformations to the application software. These transformations and proven correct-by-construction [5] by preserving all the functional properties of the application software. Secondly, HW specific components are constructed systematically from the characteristics of the sensor network platforms as well as the entities and communication mechanisms of the network protocols. As an example, the model of the wireless network includes specific details as the collision detection and avoidance techniques of the MAC layer, the out-of-order delivery and the packet losses due to possible collisions or reduction of the network bandwidth. Finally, the derived application software model is progressively enriched with the HW specific components, given a specified mapping.

The generation of the *application software model in BIP*, presented in [5], receives as input an application software model described in PPM and produces the equivalent representation in a BIP model. The construction is fully automated and preserves the behavior of the software application. Thus, the generated BIP models inherit all the merits of PPM models which enable separate analysis of computation and communication, expose functional parallelism and separate the functionality of the application from the target hardware platform.

The derived *abstract system model in BIP* is parametrized and allows flexible integration of specific target hardware features, such as communication protocols, scheduling policy etc. However, the *abstract system model in BIP* does not include all the hardware-dependent (e.g. execution times, data processing delays) and network-specific information (e.g. packet delivery ratios, end-to-end delays). The above information are injected to the model in the form of probabilistic distributions which are obtained by profiling techniques and execution of the generated code on the physical hardware platform. To compute these probabilistic distributions, we analyze the debugging traces from the execution of the generated code on the hardware platform and produce stochastic independent data [19] [13]. This technique is called calibration and results in obtaining the complete *system model in BIP*.

3.3 Code Generation

In this section, we describe the method and the associated tool for automatic generation of deployable code, targeting distributed sensor networks. The method is based on an infrastructure for generating code from PPM specifications. The generated code is portable and can be eventually deployed and run on different hardware including sensor networks. The generated code consists of the functional code and the glue code.

The functional code is generated from the application software in PPM consisting of processes and shared objects. In the case of sensor networks, processes are implemented as threads, and shared objects are implemented according to the underlying communication protocols. The implementation in C contains the thread local data and the routine implementing the specific thread functionality. The latter is a sequential program consisting of plain C used as a controller, wrapping the process C code described in PPM. The communication function calls are implemented by substituting the *read* and *write* primitives by *read* and *write* API calls on the respective communication protocol.

The glue code implements the deployment of the application to the sensor network platforms, i.e., allocation of threads to the sensors. The glue code is essentially obtained from the mapping. Threads are created and allocated to network nodes according to the process mapping, which also specifies configuration parameters for the underlying communication protocols. In particular, for User Datagram Protocol (UDP), each process is assigned a source port (srcPort), a destination (dstPort) port and a destination node IP (dstIP). The glue code is linked with sensor network hardware library to produce the binary executables for execution on the sensor network nodes.

The generated code is described in C language. Both functional and glue code are implemented using re-targetable template files and sensor network hardware specific files. The tool is implemented in C++ and it consists of approximately 35 files and 11235 lines of code.

4 Case Study: Industrial WMSN Application

We illustrate our approach using a case study provided by an industrial partner (Cyberio [2]). It targets on audio capturing and reproduction over a WiFi wireless network with the addition of local clock synchronization. In this case we focus on a sender-to-receiver synchronization, where the base station broadcasts periodically a frame containing the hardware clock value (*synchro* process of Figure 8) to all the nodes through the wireless network. Each node applies a Phase Locked Loop (PLL [20]) synchronization technique, to construct a software clock. The PLL system takes as input the broadcasted clock and keeps the local clock synchronized to it. The construction is based on the Kalman filter algorithm (Appendix A). The

[2]www.cyberio-dsi.com/

expected synchronization accuracy, defined as the difference between the input and output clock, for the particular case study is specified as 1μs. The resulting clock is used by the *micro* process to timestamp the audio frames. Subsequently, the base station is able to reproduce the received audio frames in the correct chronological order.

Sensor Network Platform Description

We target as platform a Wireless Sensor Network (WSN) of spatially distributed autonomous sensors. They are responsible of monitoring sound, referred as slave nodes, and cooperatively pass their data through the network to a base station, referred as the master node.

The wireless network (WLAN) provides the ability of bidirectional communication between all the network nodes for audio handling and clock synchronization. Thus, the choice of the master node is completely arbitrary. In addition, the WLAN is based on the IEEE 802.11 standards (WiFi).

Each network node is a hardware platform, which consists of the computational core, the WiFi and the sound card. The computational core is responsible for the node's processing operations, the WiFi card supports the wireless communication of the network, and the sound card is dedicated to capture or reproduce sound.

In the specific case study, we use a WSN that consists of three network nodes, as represented graphically in the lower part of Figure 8. As network nodes we use 3 UDOO platforms [3] and as Access Point (AP) we use a Snowball SDK platform [4]. To capture and reproduce audio samples, we used the API provided by the Advanced Linux Sound Architecture (ALSA) [5]. This API supplies structures and functions to communicate with the node's sound card through the ALSA library.

In the following subsection we present the mapping that is used for the deployment of a WMSN application to different hardware nodes.

4.1 Code Generation on Distributed Sensor Network Platform

As depicted by the deployment of Figure 8 the clock synchronization protocol runs in parallel with an audio application. The *synchro* and *speaker* processes are mapped to the Master UDOO node, whereas the *PLL* and *micro* processes to the Slave UDOO nodes. The shared objects are mapped to the WiFi cards, which are managing the communication through the Snowball SDK AP. The sensor network nodes can communicate through various modes, such as unicast, broadcast and multicast. They also support additional communication protocols, apart from UDP, such as the raw Socket protocol.

We hereby present some experimental results obtained from the generated code for the case study. The results focus on the clock synchronization accuracy of a slave node. Specifically, in Figure 9 we plot the time difference between the Master and the software clock computed in the PLL of the Slave. The software clock follows the advance of the Master clock and maintains a relative offset from it (here around 100μs) with a resulting accuracy of 76μs. As illustrated in [20], in a PLL-based approach this offset depends on the synchronization frequency of the application. Although an increase of this frequency results in better synchronization, it is simultaneously increasing the number of transmitted packets in the network. This leads to higher energy consumption, thus shortening the network lifetime.

The execution of the generated code also provided debugging traces, which we analyzed, in order to compute probabilistic distributions for specific case study parameters. These parameters concerned the computation of each local hardware clock, the packet delivery ratio and the end-to-end delays. The debugging traces were used to calibrate the BIP abstract system model and produce the BIP system model (design flow step 3), thoroughly described in the following subsection.

4.2 BIP System Model

This section presents the system model constructed for the WMSN case study. It consists of the Master component and two instances of the Slave component, using the same interfaces and interactions with the

[3]http://www.udoo.org/features/

[4]http://www.calao-systems.com/articles.php?pg=6186

[5]http://www.alsa-project.org/main/index.php/Main_Page

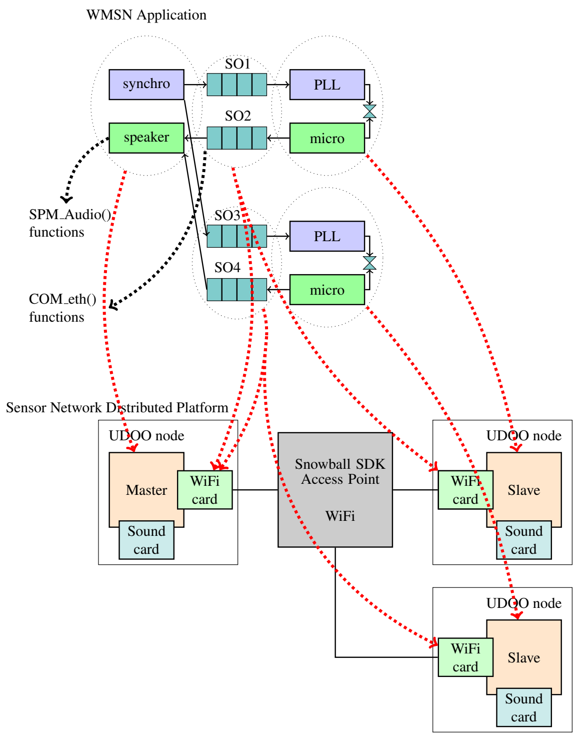


Figure 8: Mapping of the WMSN Application on the distributed network

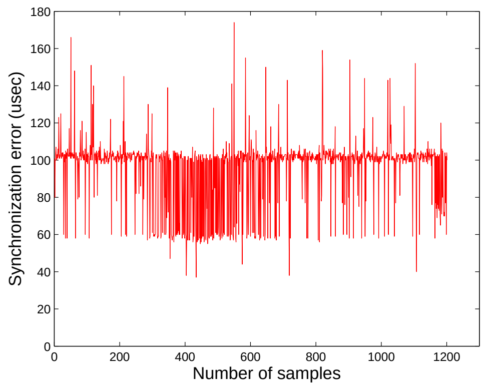


Figure 9: Synchronization accuracy (in μs) observed from the generated code

other system components. For comprehension purposes, Figure 10 illustrates a simpler system containing only one instance of the Slave component. The Master is responsible for periodical transmission of synchronization packets containing its hardware clock value through the port *CLK_SEND*. This value, as well as the Slave's hardware clock value, are obtained using probabilistic distributions for the Gaussian random variables of the discrete clock model (see Appendix A). The timing model is as a discrete time step advance and associated with the interaction *TICK*. This interaction is used as a strong synchronization among all the system components, implementing a timing model. The transmitted and received packets are stored in a buffer component (Mbuffer and Sbuffer instances of Figure 10), which follows a FIFO queuing policy. The processing and transmission of the data is handled by the WiFi component, modeling the wireless network (WiFi unit of Figure 8), and responsible for the packet transmission to every Slave component in the model. This component is using probabilistic distributions for network-specific characteristics, such as the packet delivery rate and the end-to-end delays. Whenever a synchronization packet is received by the Slave component (*CLK_RECV* port), it computes the synchronized clock of the Kalman algorithm (see Appendix A). Each audio packet is transmitted through the *AUDIO_SEND* port and timestamped with the latest computed value of the synchronized clock.

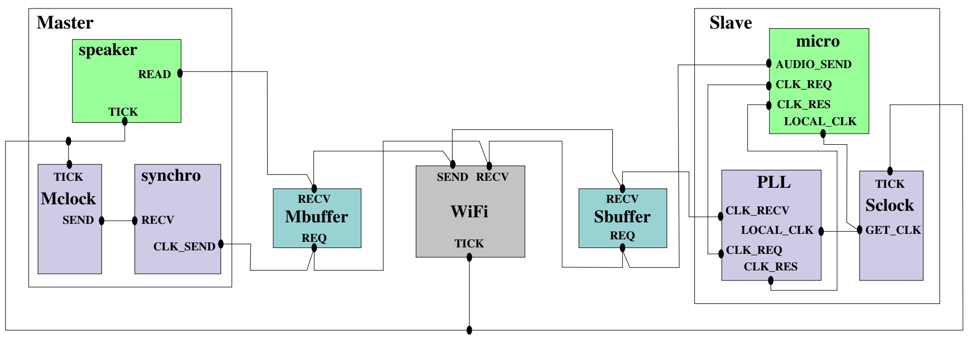


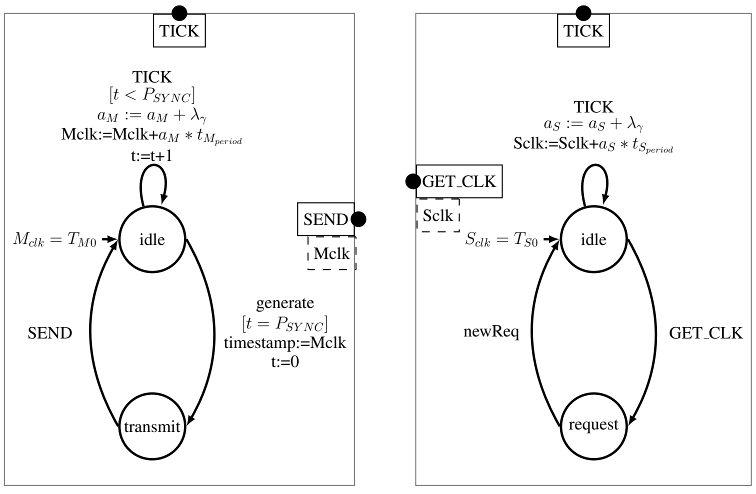
Figure 10: BIP abstract system model of the case study

Thereafter, we provide a detailed description of the representative BIP system components, depicted by finite-state automata and extended by the data and functions used in the real application. As an abbreviation we consider that the ports used in the interactions between the system components are presented in capitals, hence all the remaining are internal ports.

Component behavior

The transmission of synchronization packets is initiated by the Master compound component in the model, formed by the *Mclock*, the *synchro* and the *speaker* atomic components. The *Mclock* (Figure 11a) models the behavior of the Master's hardware clock. The *synchro* component is responsible for the periodical transmission of synchronization packets and the *speaker* component for the consumption and playout of the received audio packets. The *Mclock* component (Figure 11a) consists of the initial state *idle* and the *transmit* state. It periodically triggers the transmission of packets through an interaction with the *synchro* component. The time needed for the generation of packets (P_{SYNC}) is fixed and thus considered as a model parameter. An interaction through the port *TICK* will result in a time progress equal to one (tick) unit. When the time is equal to P_{SYNC}, the control moves from *idle* to the *transmit* state due to the corresponding guard. Following the interaction involving its *SEND* port, the current hardware clock value is forwarded to the *synchro* component. This value is computed using probabilistic distributions for the discrete clock model of the Master. The *speaker* component starts to reproduce the received audio samples periodically (P_P period) through the port *READ* after an initial playout delay p_1.

The WiFi component (Figure 12) is formed by two parts. The first concerns the reception of the transmitted frame by the Master component and the second, the response time calculation as well as the transmission of a frame to the Sbuffer component. We accordingly consider packets that lost or delivered out-of-order as failed transmissions. Consequently, in the model every frame received through the *RECV* port, is either successfully transmitted (*success* state) or discarded if delayed or lost (*degraded* state). The number of consecutive successful or failed packet transmissions is chosen by corresponding probabilistic distributions (λ_{ok} and λ_{fail} respectively). If a frame is received in the *success* state through the *RECV* port, it is stored in a FIFO queue and the value of successful packet transmissions is decreased. The frame's transmission time is accordingly chosen by the end-to-end delay distribution (λ_{delay}). Afterwards, the control moves to the second part, where the time advances through the *TICK* port. Whenever the transmission time of a frame in the queue is reached, it is forwarded to the Sbuffer component through the *SEND* port. In the meantime, if the chosen number of consecutive successful transmissions is equal to zero, the component moves from *success* to the *degraded* state. Accordingly, a value from the distribution of failed transmissions is chosen. This value indicates the number of subsequent frames, received through the *RECV* port, that are discarded. The WiFi component only returns to the *success* state, when it becomes equal to zero again.



(a) Mclock component (b) Sclock component

Figure 11: Hardware clock components of the Master and the Slave

The Slave compound component consists of three atomic components: The *Micro*, the *Sclock* and the *PLL*. The *Micro* component is responsible of capturing and transmitting periodically audio samples. Additionally, the *Sclock* component is modeling the hardware clock of the Slave and the *PLL* component (previously presented in Figure 2) computes the synchronized clock of the Kalman filter algorithm. In order to model the *Sclock* component (Figure 11b) we use the same method with the Mclock component constructing a probabilistic distribution for the discrete clock model of the Slave. Furthermore, the *PLL* component receives the transmitted synchronization packets from the Master and updates the synchronized clock. To accomplish that, it needs to interact with the *Sclock* component receiving its local clock (*LOCAL_CLK* port), in order to apply the PLL functions of the real application. It is also polled periodically by the *Micro* component (*CLK_REQ* port), in order to add a hardware clock value to each audio packet scheduled for transmission. The corresponding reply (*CLK_RES* port) contains the latest computed synchronized clock value augmented by the time elapsed between the last reception of a packet and the received request. Both are measured through an interaction with the *Sclock* component (*LOCAL_CLK* port). The *Micro* component generates each audio packet periodically (P_M period).

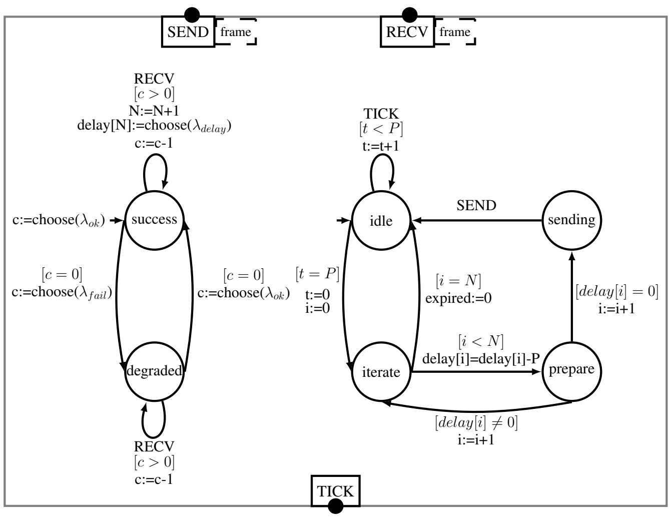


Figure 12: WiFi component

In the following subsection we report on the experimental results from the analysis of the BIP system model (step 4 of the design flow), obtained by the simulations and the use of SMC.

4.3 Analysis and experimental results

We conducted two sets of experiments, focusing on equally important requirements in the development of multimedia sensor networks. The first analyzed the utilization of the buffer components concerning only the audio capturing and reproduction in the system. Thus, this experiment focused a functional requirement, influenced by non-functional such as the packet delivery ratio and the end-to-end delays. In the second we focused on the synchronization of the device clocks. Therefore, we observed the difference between the Master clock (θ_M) and the synchronized clock computed in every Slave (θ_S) without the impact of the audio capturing and reproduction. In order to evaluate these requirements we describe them with stochastic temporal properties using the Probabilistic Bounded Linear Temporal Logic (PBLTL) formalism [4] and detail on their probabilistic results using the SMC tool of the BIP framework.

Experiment 1: Buffer utilization. We evaluated the property of avoiding overflow or underflow in each buffer component by considering the following properties: $\phi_1 = (S_{Sbuffer} < MAX)$, as well as $\phi_2 = (S_{Mbuffer} > 0)$, where $S_{Sbuffer}$ and $S_{Mbuffer}$ indicate the size of the Slave and Master buffer components accordingly. The value of MAX is considered as fixed and equal to 400. As illustrated by Figure 13 $P(\phi_1) = 1$, meaning overflow in the SBuffer is avoided, for the considered value of MAX. Furthermore, the probability of underflow avoidance in the Mbuffer depends on the initial playout delay (p_1). Specifically, in Figure 14 we can observe that for delays greater than 1430 ms $P(\phi_2) = 1$, meaning that the Master component should start the consumption of audio packets when this time duration has elapsed.

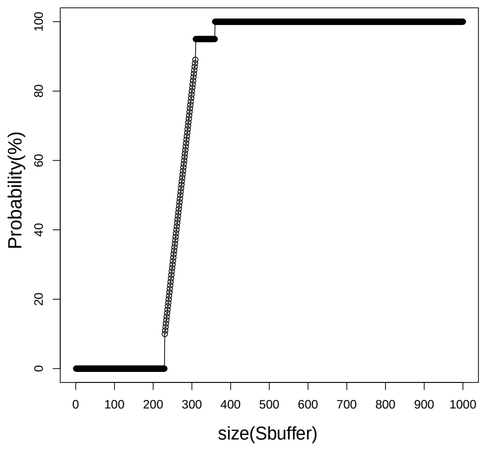


Figure 13: Probability of satisfying overflow avoidance in the Sbuffer

Experiment 2: Synchronization accuracy. The property of maintaining a bounded synchronization accuracy is defined as: $\phi_3 = (|(\theta_M - \theta_S) - A| < \Delta)$, where A indicates a fixed offset between the Master and each computed software clock and Δ is a fixed non-negative number, denoting the resulting bound. In the first step we used several probabilistic distributions from the execution results of the application to test if the expected bound $\Delta = 1\mu$s is achieved. However, as it can be depicted by Figure 15 the achieved bound by the simulations was always above the defined bound of 1μs for $A = 100\mu$s. As a second step we repeated the previous experiments, in order to estimate the best bound. Thus, we tried to estimate the smallest bound, which ensures synchronization with probability $P(\phi_3) = 1$, by repeating the previous experiment for a variety of Δ between 10μs and 80 μs. The simulations have depicted that the synchronization bound was 76 μs, as it is also observed by the execution results of the generated code in Section 4.1.

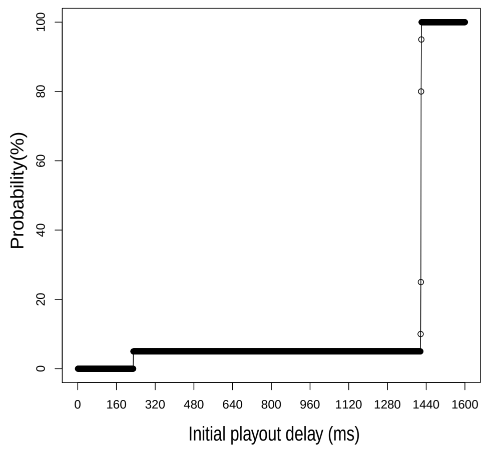


Figure 14: Probability of satisfying underflow avoidance in the Mbuffer

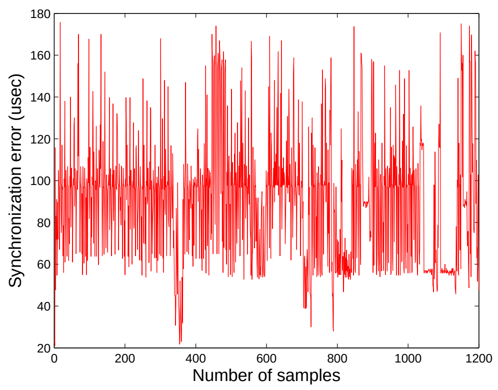


Figure 15: Synchronization accuracy (in μs) observed from the BIP model simulations

5 Conclusions

We have presented a novel approach, based on a design flow, facilitating the development of correct and operational applications for sensor network systems. It takes as input the application software and the hardware specification (communication protocol and sensor network platforms) as well as the mapping between them and constructs a system model in BIP. This model is stochastic, meaning that it can be tested, simulated and validated using the statistical model checking tool of the BIP toolset. Moreover, through the use of rapid prototyping, our approach supports the automatic code generation for the target distributed sensor network platform.

We illustrate our method through a multimedia sensor network application targetting in two paths: 1) the construction of a sensor network system model and 2) the automatic generation of correct C code for execution in the target platforms. The method is used to evaluate functional and non-functional requirements for such applications, through statistical model checking. It also exploits the advantages of code generation for deployment on the target platform and for debugging purposes. The conducted experiments focus on the buffer utilization and the synchronization accuracy of local clocks according to a common time reference in the system.

As a future work, we are considering improvements in order to decrease the relative offset between the software clock, computed in each device, according to a reference clock. Thus we are experimenting with various clock synchronization frequencies, whilst trying to keep the amount of packets in the network as low as possible. This may as well result in a possible alternation of the clock synchronization protocol. Additionally, we focus on multimedia applications for environments supporting lower resource platforms than Linux. In this scope, Basu et al. introduced in [3] formal models for TinyOS, an evenly popular environment for the development of such applications. Although supporting communication with lower resource consumption, such systems allow the transmission of a small amount of data in each packet. Therefore, in the target multimedia applications data are often transmitted in several packets. Consequently, the network is more frequently occupied, resulting in a higher probability of collision occurrence and packet loss. In order to analyze the impact of the additional latencies in the available resources, we plan to develop a similar design flow for such systems.

References

[1] Bahar Akbal-Delibas, Pruet Boonma, and Junichi Suzuki. Extensible and precise modeling for wireless sensor networks. In *Information Systems: Modeling, Development, and Integration*, pages 551–562. Springer, 2009. 2

[2] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.H. Nguyen, and J. Sifakis. Rigorous component-based design using the BIP framework. *IEEE Software, Special Edition from Routines to Services 28 (3)*, pages 41–48, 2011. 1, 3

[3] Ananda Basu, Laurent Mounier, Marc Poulhies, Jacques Pulou, and Joseph Sifakis. Using BIP for Modeling and Verification of Networked Systems–A Case Study on TinyOS-based Networks. In *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*, pages 257–260. IEEE, 2007. 5

[4] Saddek Bensalem, Marius Bozga, Benoit Delahaye, Cyrille Jegourel, Axel Legay, and Ayoub Nouri. Statistical Model Checking QoS properties of Systems with SBIP. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 327–341. Springer, 2012. 3, 4.3

[5] Paraskevas Bourgos. *Rigorous Design Flow for Programming Manycore Platforms*. PhD thesis, Université Joseph Fourier, 2013. 2, 3.2

[6] Dazhi Chen and Pramod K Varshney. QoS Support in Wireless Sensor Networks: A Survey. In *International Conference on Wireless Networks*, volume 233, pages 1–7, 2004. 2

[7] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002. A

[8] Benjamin R. Hamilton, Xiaoli Ma, Qi Zhao, and Jun Xu. ACES: adaptive clock estimation and synchronization using Kalman filtering. In *Mobile Computing and Networking*, page 152–162, 2008. 2, A, A

[9] Faranak Heidarian, Julien Schmaltz, and Frits Vaandrager. Analysis of a clock synchronization protocol for wireless sensor networks. *Theoretical Computer Science*, 413(1):87–105, 2012. 2

[10] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 73–84. Springer, 2004. 3

[11] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. The platforms enabling wireless sensor networks. *Communications of the ACM*, 47(6):41–46, 2004. 2

[12] S Jager, Tino Jungebloud, Ralph Maschotta, and Armin Zimmermann. Model-Based QoS Evaluation and Validation for Embedded Wireless Sensor Networks. 2

[13] Jean-Yves Le Boudec. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010. 3.2

[14] K Lee, John C Eidson, Hans Weibel, and Dirk Mohl. IEEE 1588-Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In *Conference on IEEE*, volume 1588, 2005. 2

[15] Aneeq Mahmood, Georg Gaderer, Henning Trsek, Stefan Schwalowsky, and N Kero. Towards high accuracy in IEEE 802.11 based clock synchronization using PTP. In *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*, pages 13–18. IEEE, 2011. 2

[16] Satyajayant Misra, Martin Reisslein, and Guoliang Xue. A survey of multimedia streaming in wireless sensor networks. *Communications Surveys & Tutorials, IEEE*, 10(4):18–39, 2008. 2

[17] Mohammad Mostafizur Rahman Mozumdar, Francesco Gregoretti, Luciano Lavagno, Laura Vanzago, and Stefano Olivieri. A framework for modeling, simulation and automatic code generation of sensor network application. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON'08. 5th Annual IEEE Communications Society Conference on*, pages 515–522. IEEE, 2008. 2

[18] Mohammad Mostafizur Rahman Mozumdar, Luciano Lavagno, Laura Vanzago, and Alberto L Sangiovanni-Vincentelli. Hilac: A framework for hardware in the loop simulation and multi-platform automatic code generation of WSN applications. In *Industrial Embedded Systems (SIES), 2010 International Symposium on*, pages 88–97. IEEE, 2010. 2

[19] Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, and Saddek Bensalem. Building Faithful High-level Models and Performance Evaluation of Manycore Embedded Systems. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 10th IEEE/ACM International Conference on*. IEEE, 2014. 3.2

[20] Fengyuan Ren, Chuang Lin, and Feng Liu. Self-correcting time synchronization using reference broadcast in wireless sensor network. *IEEE Wireless Commun.*, 15(4):79–85, 2008. 4, 4.1

[21] Taniro Rodrigues, Priscilla Dantas, Flávia Coimbra Delicato, Paulo F Pires, Luci Pirmez, Thais Batista, Claudio Miceli, and Albert Zomaya. Model-driven development of wireless sensor network applications. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pages 11–18. IEEE, 2011. 2

[22] Ludovic Samper, Florence Maraninchi, Laurent Mounier, and Louis Mandel. GLONEMO: Global and accurate formal models for the analysis of ad-hoc sensor networks. In *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 3. ACM, 2006. 2

[23] Bharath Sundararaman, Ugo Buy, and Ajay D Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323, 2005. 2, A

[24] Lothar Thiele, Iuliana Bacivarov, Wolfgang Haid, and Kai Huang. Mapping Applications to Tiled Multiprocessor Embedded Systems. In *Proceedings of the Seventh International Conference on Application of Concurrency to System Design*, ACSD '07, pages 29–40, Washington, DC, USA, 2007. IEEE Computer Society. 3.1

[25] Simon Tschirner, Liang Xuedong, and Wang Yi. Model-based validation of QoS properties of biomedical sensor networks. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 69–78. ACM, 2008. 2

[26] Håkan LS Younes. Ymer: A statistical model checker. In *Computer Aided Verification*, pages 429–433. Springer, 2005. 3

Appendices

A Kalman filter algorithm

This clock synchronization algorithm (proposed in [8]) continuously corrects the local clock reducing its offset from the master clock. A clock is defined by a discrete model as follows:

$$\theta[n] \quad = \quad \sum_{k=1}^{n} \alpha[k]\tau[k] + \theta_0 + \omega[n] \tag{1}$$

, where α is the clock skew, $\tau[k]$ the sampling period at the k^{th} sample, θ_0 the initial clock offset, and $w[n]$ the random measurement as well as other types of additive noise. In a sender-to-receiver synchronization, this noise consists of four factors [23]:

- the time for message construction and sender's system overhead,
- the time to access the transmit channel,
- propagation delay,
- the time spent by the receiver to process the message.

Since $\tau[k]$ can be different, the above clock model covers uniform and non-uniform sampling. Equation (1) can be rewritten recursively as follows:

$$\theta[n] \quad = \quad \theta[n-1] + \alpha[n]\tau[n] + \vartheta[n] \tag{2}$$

, where $\vartheta[n] = \omega[n] - \omega[n-1]$ is considered as a Gaussian random variable with mean 0 and variance σ_{ϑ}^2, as described in [7]. We assume that the clock skew $\alpha[n]$ is time-varying, that is, it can change completely from one sample to another with the optimal estimator being:

$$\hat{\alpha}[n] \quad = \quad \frac{\theta[n] - \theta[n-1]}{\tau[n]} \tag{3}$$

This variation can be modeled as a random process defined by the Equation (4):

$$\alpha[n] \quad = \quad \alpha[n-1] + \gamma[n] \tag{4}$$

, where γ is considered as a Gaussian random variable with mean 0 and variance σ_{γ}^2 indicating the noise model,as described in [8]. As the above equations are used to define the Kalman Filter algorithm, we accordingly illustrate its vector-matrix form, previously introduced in [8].

Let θ denote the master timestamp in which we add the noise delays (see Equation (1)), and $\tilde{\theta}$ the value of the synchronized clock.

$$\tilde{\theta}\,[n] = \sum_{k=1}^{n} \alpha\,[k]\,\tau\,[k] + \ \theta_0 \Rightarrow$$

$$\tilde{\theta}\ \ [n] = \ \tilde{\theta}\ \ [n-1] + \ \alpha\,[n]\,\tau[n] \tag{5}$$

Based on the Equation (4), the Kalman Filter state of the synchronized clock is defined by the Equation (6).

$$x\,[n] = Ax\,[n-1] + \ u\,[n] \tag{6}$$

, where $x\left[n\right] = \ [\tilde{\theta}\ \left[n\right] \quad \alpha[n]]^T$, $A = \begin{bmatrix} 1 & \tau \\ 0 & 1 \end{bmatrix}$, $u\left[n\right] = [0 \quad \gamma[n]]^T$ and τ is the sampling period. The Kalman Filter observation Equation is the noisy observation of the reference clock (Equation (7)).

$$\theta\left[n\right] = \ \tilde{\theta}\left[n\right] + \ v\left[n\right] = \ b^T x\left[n\right] + \ v\left[n\right] \tag{7}$$

, where $b^T = [1 \quad 0]$. Then, the Kalman Filter vector-matrix form is defined by the following Equations:

$$\hat{x}[n] = A\hat{x}[n-1] + \ G\left[n\right]\left(\theta\left[n\right] - \ b^T \ A\hat{x}[n-1]\right) \tag{8}$$

$$S\left[n\right] = AM\left[n-1\right]A^T + \ C_u \tag{9}$$

$$M\left[n\right] = \left(I - G\left[n\right]b^T\right)S\left[n\right] \tag{10}$$

$$G\left[n\right] = S\left[n\right]\ b\left(\sigma_v^2 \ + \ b^T\ S\left[n\right]\ b\right)^{-1} \tag{11}$$