DISSERTATION

Energy Profiling of Networked Embedded Systems

Submitted at the Faculty of Electrical Engineering, Vienna University of Technology in partial fulfillment of the requirements for the degree of Doctor of Technical Sciences

under supervision of

Univ. Prof. Dr. habil. Christoph Grimm
Institut number: 384
Institute of Computer Technology

and

Univ. Prof. Dr. habil. Ian O'Connor
Electronic Electrical and Control Engineering Department
Ecole Centrale de Lyon

by

Javier Moreno Molina
Matr.Nr. 0627190
Peter-Bardens-Str. 9, 67661, Kaiserslautern, Germany

Date _____

Kurzfassung

In dieser Arbeit werden Methoden diskutiert und entwickelt, die es ermöglichen, den Energieverbrauch in vernetzten, eingebetteten Systeme über mehrere Ebenen eines Systems hinweg zu simulieren und so erstmals eine ebenenübergreifende Optimierung der Stromaufnahme zu ermöglichen.
Das Problem ist, dass alle Ebenen einen Beitrag zum Gesamt-Energieverbrauch leisten. Obwohl die Optimierung des Energieverbrauchs damit ein ebenenübergreifendes Problem ist, findet eine Energie-Optimierung bislang primär auf den Hardware- und Hardwarenahen Ebenen statt. Dies vor allem, weil man Energieverbrauch sehr einfach einer Hardwarekomponente zuordnen kann. Um âĂđPower-AwarenessâĂIJ auch auf höheren Abstraktionsebenen - wie die der Software-Entwicklung oder Netzwerkprotokolle - zu schaffen, ist daher zunächst eine Cross-Layer und Cross-Domain Modellierung des gesamten Systems erforderlich.
Die Offenheit heutiger, drahtlos vernetzter und verteilter Cyber-Physikalischer Systeme erfordert domänenübergreifende Modellierung auch von Umwelt-Interaktionen und Netzwerkereignissen, deren Auswirkungen auf den Energieverbrauch es zu berücksichtigen gilt.
Daher schlägt die vorliegende Arbeit zunächst eine Simulationsumgebung vor, die in der Lage ist, Modelle unterschiedlicher Aspekte zu nitrieren und eine Gesamt-Systemsimulation zu ermöglichen. Das vorgeschlagene Framework basiert auf SystemC TLM und C++, die sich als de-facto Standard für den Co-Design von Hardware und Software etabliert haben. Ausserdem stellt das Framework ein TLM basiertes Framework bereit, mit dem sich Inter- und Intra-Knoten-Kommunikation einschlieÃ§lich der Funkwellen-Ausbreitung modellieren lassen. Um die physikalische Umgebung zu simulieren, werden die Systeme AMS Erweiterungen verwendet.
Kern der Arbeit ist dann ein Konzept zum Profiling des Energieverbrauchs. Das Konzept ist insbesondere in der Lage, die semantische Lücke zwischen der Hardware-Ebene und der Welt der Software- und Netzwerkentwicklung zu schlieÃ§en. Der Energieverbrauch der Hardware wird zu Energie-Profilen aggregiert, die aussagekräftige Informationen in Bezug auf die Energieeffizienz von Software und Netzwerkprotokollen und Topologie darstellen.
Das so entstandene Framework wird mit anderen Ansätzen zur Simulation verglichen. Hierzu wird ein Cyber-Physikalisches System modelliert. Dann wird die Energieeffizienz unterschiedlicher Designalternativen in Software und Netzwerk verglichen und mögliche Energie-Ineffizienzen identifiziert.

Abstract

In this work, strategies to simulate and account energy and power consumption in networked embedded systems are studied, proposed and discussed, in order to enable energy consumption optimization of the system across all levels.
All design levels have their contribution and responsibility in overall system energy consumption. However, although energy consumption optimization is a cross-level problem, energy awareness is almost exclusive of hardware design. This is mainly because energy consumption estimation requires accurate power models of hardware subsystems. To bring up energy awareness in other design levels, such as software or network, a holistic cross-level and cross-domain modelling approach is required.
Furthermore, openness of wireless, distributed and cyber-physical systems relapses into the cross-domain problem, as it requires considering the significant effect of environment interactions and network events in overall energy consumption. These features are nowadays present in the vast majority of embedded systems networks.
Hence, this thesis proposes a simulation framework which is capable of integrating models of all aspects of cyber-physical and distributed embedded systems. The framework is based on SystemC/TLM and C++, which is already an industry standard for hardware and software co-design. In addition, the framework provides a TLM based, inter- and intra-node communication model, including a wireless radio propagation model which enables the integration of the networking aspects. Furthermore, the framework integrates simulation of some physical processes through SystemC-AMS.
This thesis also proposes an energy profiling approach to close the semantic gap between the energy consumption estimation, performed at hardware level, and software and network designers. Hardware level data is processed and aggregated in high-level energy profiles that can provide meaningful information that enables energy aware design at software and network levels.
The framework is evaluated and contrasted with a state-of-the-art simulator. Framework capabilities are assessed by modelling a real Cyber-Physical System application. The energy profiling approach is implemented using the proposed framework and demonstrated by exploring and comparing different software and network design alternatives, identifying possible energy leakages.

In memory of my dear sister María.

Acknowledgements

First I would like to thank Prof. Christoph Grimm, who gave me the opportunity to complete this research work. First in Vienna University of Technology and later in University of Kaiserslautern. His confidence and support have been determining. I must also thank Dr. Jan Haase and Dr. Stefan Mahlknecht, who led the projects I have worked in, as well as my colleagues during my time in Vienna: Markus Damm, Josef Wenninger, Yaseen Zaidi and, specially, Sumit Adhikari, who shared his knowledge and passion, not only in science and engineering, but also in life. I would like to express my gratitude also to Prof. Dietmar Dietrich, whose inspiring vision has been probably the major influence to get involved in science and research.

In Kaiserslautern, I want to thank Manuela Burkart, who helped me establishing myself in the new city and working place and the rest of my colleagues who have been like a new family for me along the last two years: Ralf Grünwald, Xiao Pan, Carna Radojicic, Thiyagarajan Purusothaman, Frank Wawrzik and George Adrian Ciolacu.

My stay in Vienna would not have been the same without all the good friends I have left there. So thank you, Josep Colom, for so many dinners, parties, squash death matches, etc. Thank you, Elena Recas and Miriam del Río, my two female friends, that were there from the very beginning and have endured and grown up with me. Thanks also to my festival and sports-watching buddy David Fernández. Thank you Thomas Herbst, for those Australian Hot Dogs. Thank you, Manuel Pascual, Emilio Muñoz, Miriam Kim, Nacho García, Raúl Ramos, Germán Meyer...

Eight years after leaving Madrid, I still have some good friends that deserve their own space in these words. Thank you, Manuel Achúcarro and Javier Sauras, for so many visits and so many good times in so many different countries. Thank you, Ivan Blanco, for never giving up talking. Thank you Miguel Muñoz, for never talking but being always ready in those nights in Madrid.

I owe some words also to my family, to my parents Natividad and Jose Miguel, for your unvaluable love and support, and to my sister María, who always gave me much more than I could ever give back. I miss you.

And last but not least, to Lorena Mardones, who has accompanied me in this long journey and who has given me the courage and emotional support to accomplish all my goals. And he is still not conscious of all this, but thank you too, Daniel.

Table of Contents

Abbreviations

AMS	Analogue Mixed Signal
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BER	Bit Error Rate
CPS	Cyber-Physical System
DE	Discrete-Event
EMU	Energy Management Unit
ESL	Electronic System-Level
FMI	Functional Mock-Up Interface
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
HDL	Hardware Description Language
IoT	Internet of Things
ISS	Instruction Set Simulator
MAC	Medium Access Control
MANET	Mobile ad hoc network
MBD	Model-Based Design
MEMS	Microelectromechanical Systems
MoC	Model of Computation
M2M	Machine to Machine
PCB	Printed Circuit Board
RF	Radio Frequency
SDF	Synchronous Data-Flow
SNR	Signal-to-Noise Ratio
SoC	System on Chip
TDF	Timed Data-Flow
TLM	Transaction-Level Modeling
TTL	Time-To-Live
WSN	Wireless Sensor Network
XML	Extensible Markup Language

1 Introduction

The underlying motivation for this research work is, in essence, to achieve a better exploitation of the available energy resources, which is indeed a universal problem humanity has to continuously confront in all kind of activities and domains. In spite of the universality of the problem addressed, this research is focused on the specific field of networked embedded systems, with special focus on those that are connected through wireless interfaces.

Today's embedded systems are autonomous devices that can typically interact with their environment and communicate with other devices and which are produced at very low cost. In the case of wireless devices, not only production, but also deployment and installation costs can be very inexpensive. As a result, these embedded systems can be easily networked in order to support more sophisticated distributed applications, which are capable of obtaining information from very different sources in order to take some actions.

Such high flexibility and inexpensiveness has opened a huge potential market for this kind of devices and distributed applications. This is how new ideas and concepts have emerged and are gradually acquiring relevance.

A first example of these ideas is the concept of **Wireless Sensor Networks (WSN)**. Sensors are capable of detecting physical nature. Current technology permits the integration of microcontrollers, Radio Frequency (RF) transceivers and Microelectromechanical Systems (MEMS) and sensors in a single System-on-Chip (SoC). A wireless sensor network is a distributed collection of autonomous sensor nodes that communicates through radiofrequency with a central unit. This way, WSNs are a bridge between the physical world and information networks [RSZ04].

Over the years, develpment of wireless sensor nodes has enable not only the detection of physical nature, but real interaction with it. These distributed networks of embedded systems have therefore become **Cyber-Physical Systems (CPS)**, with a closed loop between both the physical and the computational worlds [LS11].

In parallel, wireless technology has also been improved. The size, cost and energy consumption of RF transceivers has been reduced and they are now being included in all kind of devices, contributing significantly to the development of **Machine to Machine (M2M)**. This trend is also enlightening the way to the ambitious concept of the **Internet of Things (IoT)**, that advocates for the interconnection of potentially all kind of things or objects through an embedded system that turns them into so called smart objects [Kop11].

Although wireless typically refers to the nature of the communication among these devices, in many cases a wired power supply will not be available either. Power must therefore be supplied

by batteries or energy harvesters and, in consequence, energy becomes a very limited resource which constrains the operation and feasibility of this kind of systems.

Furthermore, energy considerations become also of major importance with the ubiquitous computing concept. Even though embedded systems energy consumption is typically low, if this kind of systems are to be included in most objects around as, like in the Internet of Things concept, the resultant total energy consumption might have a significant impact in household electricity bills and national energy sustainability plans.

In the context of this thesis, **Networked Embedded Systems**, refers to networks of autonomous and heterogeneous computational systems, which are able to communicate among them and interact with the physical environment, that serve as a platform for all kind of distributed applications, especially those for monitor and control of physical processes.

1.1 Simulation in Embedded Systems Design

The first step to create to create optimum embedded systems designs is to be able to acquire information about how the system will perform. With this information, not only the feasibility of the system can be evaluated, but some important design decisions can be explored before implementation and deployment, leading to early stage optimization.

Building prototypes to gather this information is not always feasible. In particular, in the case of embedded systems, prototyping introduces a big overhead in cost and time-to-market and the use of virtual prototypes is already a common practice.

In wireless distributed embedded systems, deployment conditions are often unpredictable, and therefore, validation and verification of the system might require the evaluation of multiple scenarios and environmental conditions. Besides, those scenarios and conditions might be very expensive to reproduce, such as a sensor network on an airplane for in-flight monitoring. A common approach to this problems are Model-Based Design methodologies. Simulated models can be used to feasibly recreate test scenarios to validate and verify the system. With the appropriate interfaces, it is even possible to perform hardware-in-the-loop simulations, where the real hardware can be evaluated in simulated environments.

Furthermore, the first aspect encountered when facing optimization of such complex systems, is that it is a very interdisciplinary problem. Although every design area (e.g. software, hardware, network) can optimize their designs independently, major improvements are achieved when exploring the synergies among them. Models provided through the Model-Based Design methodologies foster the find of these synergies.

1.1.1 The Energy Optimization Problem

The use of Model-Based Design, and, in particular, of simulated models for energy optimization of wireless distributed embedded systems requires several considerations:

- Unlike in other simpler systems or scenarios, energy consumption estimations cannot be extracted solely from hardware models. Hardware operation is determined by the software, the environment and other devices in the same network. Therefore, although power is physically consumed in hardware, the hardware usage is driven by the application, which in this case is distributed over a network and includes the physical environment.

- Obtaining the energy consumption of a specific node is not sufficient to evaluate the impact on the whole system. Distributed systems are usually redundant fault-tolerant systems which are able to operate normally even when several nodes run out of energy. Energy consumption estimations must therefore be obtained for the whole distributed system. Likewise, optimization must be performed with the whole system in mind, as extending the lifetime of a node might diverge from improving the lifetime of the whole system.
- Optimizing software and network for energy consumption requires a deep and precise knowledge of how the hardware is used in order to estimate which design decisions actually improve energy consumption. Unfortunately, the required high-level of abstraction used in software and network design, also involves hardware usage transparency, creating a semantic gap between the high-level designer and the use of resources, in this case, the energy.

1.2 Challenges

Examining the requirements for a simulation approach of the energy optimization problem in systems with cyber-physical interaction and distributed over a wireless network, several difficulties become apparent.

- **Multi-Domain Simulation:** Energy consumption simulation requires the combination of very different simulation models, coming from different domains. Only through this combination, a consistent model of the distributed system can be achieved. However, this is a big challenge from the simulation point of view.
- **Multi-Level Simulation:** Accuracy-efficiency trade-off is a constant in Model-Based Design and simulation. In this particular case, energy consumption estimation requires very accurate models of some of the system hardware components. On contrast, high level optimization requires a comprehensive model of the distributed system, which includes modelling numerous complex and heterogeneous devices as well as the interactions among them (network) and with their surrounding environment. Furthermore, in order to estimate the lifetime of the system, simulation of even several years of operation might be required. With such a demanding scenario, an optimal handling of different abstraction levels becomes imperative.
- **Energy Consumption Semantic Gap:** Abstraction required for simulation performance has to be done with energy consumption in mind. Otherwise, it might result in the aforementioned semantic gap that typically stands between the energy consumption data and the high level designers. Abstraction has to improve performance and provide transparency where needed, but must expose energy consumption data. Furthermore, energy consumption data is meaningless if it is not provided together with its relevant context, which permits not only estimating how much energy was consumed but also associating energy consumption with its corresponding high level task, which is crucial for energy awareness and, consequently, for optimization.

1.3 Contribution

During the state-of-the-art analysis included in Chapter 3, a lack of energy optimization at high levels of abstraction has been observed. Most research works study the energy optimization

at the hardware level or, in best cases, at the operating system level. However, the problem analysis performed in Chapter 2, reveals that higher levels have huge responsibility in energy consumption and therefore offer very good chances for optimization. Nevertheless, high level optimization examples in literature are insufficient for software, there are some methods and simulators to account energy consumption of executed software [GNMO12], but very little about how to account this data to archetypical software tasks, i.e. create energy consumption profiles. In the case of energy consumption profiling for communication, examples in literature are practically non-existent.

The main reason for the almost entire absence of high level energy optimization research is most likely the increasing lack of energy consumption awareness on every higher abstraction level. Together with other implementation details, the information about power consumption is lost during the abstraction process. Once this information is lost, reconstructing it becomes an arduous task. As a consequence, the designer has no tangible information available to support the benefits, in terms of energy, of his design decisions.

Hence, the contribution of this work is twofold:

1. The proposal and implementation of a comprehensive simulation framework which permits modelling hardware, software, network and physical aspects of the system in order to enable the simulation of cross-level features, like energy consumption. This framework, named SICYPHOS, is open-source and available in `http://sourceforge.net/projects/sicyphos/`.

2. The proposal of an energy profiling approach to provide meaning and context to energy consumption simulated data, in order to improve energy awareness at high levels of abstraction and enable high level energy optimization of the overall system. The novelty of the approach lies, in particular, in the network profiles, which enable the optimization of the distributed system resource partition and intelligence distribution as well as optimizing the network topology and the communication protocols to achieve the longest lifetime of the distributed system. The energy profiling approach is implemented and integrated in SICYPHOS.

1.4 Scope

This thesis aims to facilitate energy optimization by proposing new modelling techniques and approaches that provide the designers with the best possible infrastructure to optimize their designs. However, optimization itself depends on the designer, the models and the specific application. The specific models of the different subsystems are application dependent and different decisions concerning accuracy-performance trade-off might be possible depending on the specific system requirements.

Although the modelling approach has been implemented for evaluation using a specific modelling language, the energy profiling approach itself is language- agnostic and independent and could be implemented using other modelling platforms.

In addition, although ultra-low power design may benefit from some of the results of this research, the main focus is on energy-aware design. The goal is to obtain distributed systems with best possible energy performance, but not to create optimal ultra-low power designs.

2 Energy-Aware Networked Embedded Systems Design

Power and energy constraints are recurring issues in embedded systems design and therefore they are a problem that has already been addressed and is already well-known and characterized. However, optimizing distributed embedded systems is more complex than just optimizing the embedded systems that compose them. Unfortunately, research to study the specifics of the distributed system energy optimization is almost non-existent.

This chapter presents fundamental ideas and concepts for power and energy aware design and analyses and extends them for the case of distributed systems. Moreover, it studies the effect that the cyber-physical interaction, through sensors and actuators, has on the energy optimization problem. This analysis is the basis for the optimization approach presented in next chapters.

First of all, the archetypical architecture of a sensor node will be presented. It is necessary to know the different subsystems and their contribution to energy consumption. Besides, energy consumption simulation will require modelling those subsystems, and therefore the difficulties have to be considered.

After the sensor nodes are introduced, the system model is completed with the network and cyber-physical interactions, which are part of the overall system concept. The implications of both network and cyber-physical components in energy consumption are presented and analysed.

Finally, power and energy consumption characteristics are dissected. The energy aware problem and the ultra-low power problem are distinguished and clarified. Power consumption is explained from the purely hardware perspective and high-level power and energy consumption problems are then presented.

2.1 Architecture Analysis

The ecosystem of embedded systems used in distributed cyber-physical applications offers a huge variety. Depending on the application and the distribution approach, the hardware requirements for these systems can be very different.

However, when we talk about wireless sensing devices, there are some common elements which are always present, even though their characteristics might vary significantly. A generic scheme containing these common parts is provided in Figure 2.1.

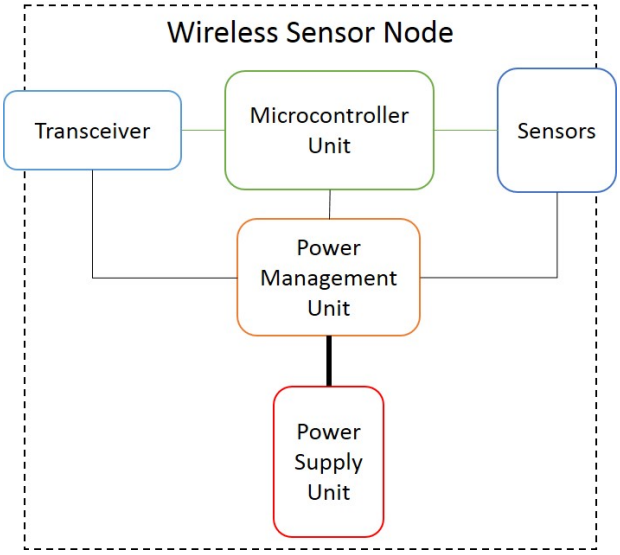

Figure 2.1: Block diagram of a generic embedded system architecture for a wireless sensor node.

All these common elements have similar characteristics from system to system and have similar requirements for their modelling approach.

2.1.1 Sensors

They are the interfaces that enable communication from the physical world, the environment, to the cyber-world, the embedded system. Sensing devices are as diverse as the nature of the quantities they sense.

A temperature sensor such as Texas Instruments LM73 can consume 1 μW while an automotive camera sensor chip like Omnivision OV7949 consumes 168 μW, which is a difference of two orders of magnitude. Therefore, while in some cases their contribution to overall energy consumption might be negligible, in other cases it might even become the limiting factor.

Furthermore, some physical processes variate smoothly along time and just require some periodic monitoring, while other physical processes require continuous measurements, with the consequent implications in energy consumption.

2.1.2 Transceiver

Wireless connectivity is provided by the transceiver. The contribution of the transceiver to power and energy consumption is typically a dominant factor.

Most transceivers have different operation modes established. Hence, the microcontroller through application or protocol stacks, can control whether the transceiver listens, sends a message or just stays idle or in an ultra-low power sleep mode.

Correct selection and tuning of a transceiver must take into account the network topology and the environment. Transmission power is directly related to the electrical power consumed when sending a message. Therefore, choosing an appropriate transceiver with appropriate configuration

and antenna has a great impact on power consumption. An excessive transmission power implies a waste of energy, but also a very scarce transmission power has negative consequences for energy consumption, as it might lead to retransmissions or additional hops for the message to reach the destination.

2.1.3 Microcontroller

Microcontrollers are the other dominant factor in energy consumption. The election of a suitable microcontroller must consider power consumption but also capability and performance, which depending on the application might greatly differ.

In recent years, some multi-processor architectures have been proposed for energy awareness. The most powerful processor is only used for most demanding tasks. If tasks are very simple, the powerful and resource demanding processor can remain in sleep mode and the tasks can be executed in an ASIC, FPGA or even another small processor with lower power consumption.

2.1.4 Energy Sources

The first step on the way to energy optimization is to analyse the energy restriction problems. Energy restrictions manifest in different ways depending on the nature of the energy sources. Therefore, although our energy optimization is focused strictly on the consumption side, it is necessary to introduce the different energy sources used in embedded systems in order to appropriately optimize the energy behaviour of the consumer parts.

There are mainly three forms of energy sources that typically supply power to the embedded devices: power grid, batteries and energy harvesters.

2.1.4.1 Power Grid

In some cases, the embedded systems have a power supply directly connected to the power grid. In these cases there is energy availability all the time, except from exceptional blackouts. Energy constraints are not so critical for the correct operation of the devices. However, the energy consumed from the power grid involves a cost all along the lifetime of the system. This energy cost cannot be neglected.

On one hand, the energy cost is a competitive advantage that may determine the success of the product.

On the other hand, embedded systems tend to be included in every object around as, as the Internet-of-Things concept suggests. In this context, in which every single object is a potential energy consumer, the energy consumption of embedded systems, even in stand-by state, becomes a real sustainability problem.

2.1.4.2 Batteries

Battery-powered nodes have a limited energy budget. If batteries can be replaced, battery lifetime affects the maintenance costs. However, in many cases, a battery replacement is not even possible and the lifetime of the whole system depends on the time until batteries are depleted.

Batteries are continuously improving. However, in many cases their efficiency is not sufficient and the cost is too high. Many wireless devices require small size and light weight. In those devices, the battery is the subsystem that most severely contributes to both size and weight. And in spite of that fact, the energy supplied is often not enough.

Moreover, batteries have a limited lifetime, and a non-linear response. They provide a voltage level, which becomes lower along both time and usage, until depletion, which occurs when the provided level is not enough to supply the system.

Furthermore, environmental conditions, such as temperature, severely affect the performance of the batteries.

2.1.4.3 Energy Harvesters

The use of energy scavengers is conditioned by the application and environment. On one hand, there is the energy that can potentially be gathered by the scavenger. Nodes in direct sun exposure, subjected to movement, vibrations, etc. or to temperature gradient are able to gather energy from their environment. On the other hand, power requirements shall be low, without peaks which may be very difficult to fulfil. For energy scavengers, it is relatively easy to fulfil a continuous demand of energy, rather than a sudden power peak.

Unlike batteries, there is no lifetime limit for energy scavenging, which is possible as long as the system remains undamaged. However, the energy available is not usually enough tu fulfil system demands.

2.2 The Impact of Network and Cyber-Physical Interaction

This dissertation is framed within networked embedded systems. Nowadays, significant applications of those are, for instance, Wireless Sensor Networks (WSN) and Cyber-Physical Systems (CPS).

Wireless Sensor Networks (WSN), are spatially distributed sensors which capture some physical information from the environment and transmit it to a main location which further processes this data.

Cyber-Physical Systems are integration of computational and physical processes [Lee08a]. The interfaces between the physical environment and computational devices are both sensors and actuators. While the main goal in WSNs is to gather information about the environment, cyber-physical systems involve more intelligence in the network so that the system itself reacts to environment changes and is able to perform some actuation based on those changes, closing the feedback loop between the physical environment and the computational system.

Both application paradigms add new levels of complexity to the already complex energy estimation problem. In both cases, the network and the environment become an actual part of the system.

Figure 2.2 illustrates this interaction between nodes, environment and network. The behaviour of the system without taking into account the physical behaviour of the network and the environment, becomes unpredictable.

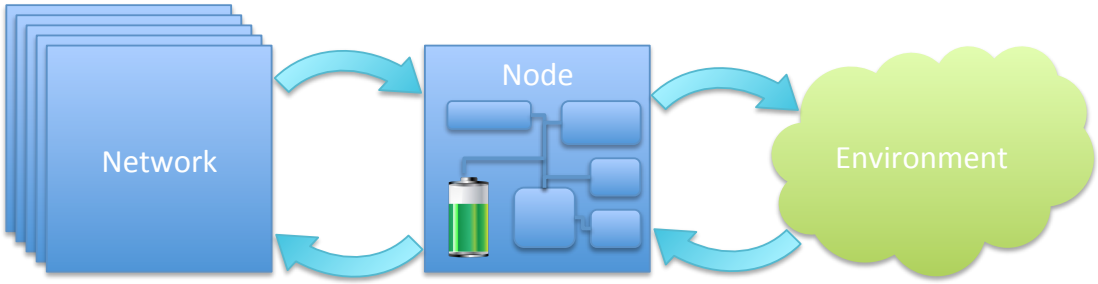

Figure 2.2: The network and the environment are part of the system and influence the energy consumption.

For instance, a wireless sensor node might have to report more frequently the sensed values if the physical quantity they sense is more variable and volatile. Sometimes the amount of information that has to be gathered by the sensor generates a lot of data that has to be transmitted, which is a very costly task. With energy simulation, the specific scenario can be considered, taking into account the physical process variability, the energy required for communication and the energy required for hardware and software operation. With all these elements, different solutions can be explored, and the best system architecture can be selected. Sometimes, some intelligence in the nodes can reduce the data to be transmitted. A more powerful intermediate node could also improve the efficiency. However, the early evaluation of the real advantages of this design alternatives is really necessary in order to avoid wasting too much resources in some designs that ultimately will bring no real advantage.

The network also has a tremendous impact in energy consumption. The distance between the sensor node and the sink node or the destination node that collects the measurements, determines the necessary transmitting power of the transceiver, which is directly related with the required power consumption. The communication protocol has influence on energy consumption as well. A contention based protocol reduces latency and does not require synchronization, however they are prompt to collisions. A scheduled based MAC protocol could avoid collisions but requires some overhead to synchronize the nodes. The best option can only be selected after a careful scenario analysis which includes a realistic network scenario and an accurate energy consumption model of all the nodes. Besides, the best option for a particular node is not necessarily the best option for all the nodes in the network.

Furthermore, sometimes the nodes are not close enough to the sink node and multi-hop routing is required. The complexity of energy optimization for a multi-hop routing scenario is extremely complex. Listening for incoming communication is a very costly operation in terms of energy consumption. Therefore, sophisticated solutions have to be evaluated in order to wisely select when to listen and when to go to sleep mode.

Therefore, for an accurate estimation of the energy consumption, not only the hardware/software platform has to be considered, but the network and the environmental conditions as well.

In Model-Based Design methodology accurate physical environment models are required for design and optimization and also for validation and verification through Hardware-in-the-Loop techniques.

As this work focuses on energy estimation, there is no need for very accurate physical models. Physical models are required as long as they condition the system behaviour and, consequently, its energy consumption. However, validating the cyber-physical aspect of the system is out of scope.

2.3 Ultra-Low Power and Energy Aware Embedded Systems Design

Power and energy are terms that are very often interchangeable. Energy is the power consumed along time. Therefore, when power consumption remains stable, with no significant fluctuation, energy reduction can be inferred from power reduction. However, in many cases, and specially when using power management techniques, power consumption may change drastically along time. In these cases, interchanging power and energy terms would be a serious mistake. Using the appropriate term is therefore very important to avoid confusion.

In Figure 2.3, the power consumption of two different devices is shown. The device represented in red reaches higher power consumption values than the device depicted in blue. Nonetheless, it is also visually observable that the device represented by the red power function has lower energy consumption.

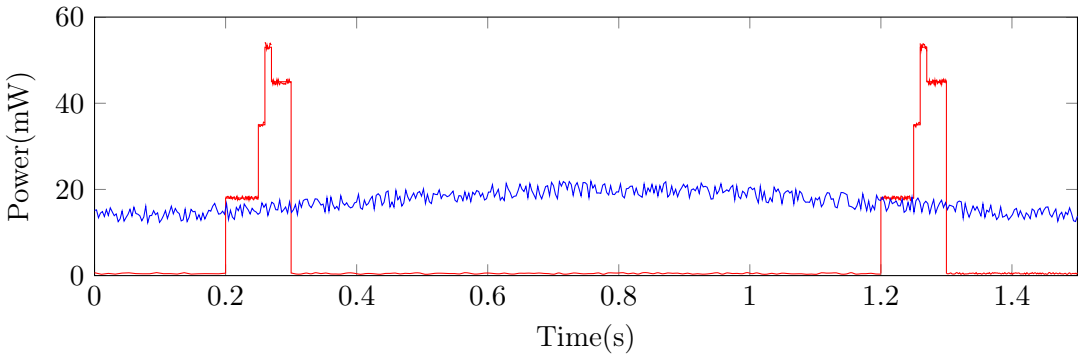

Figure 2.3: Power consumption of two different systems

In embedded systems, power restrictions have to be distinguished from energy restrictions. Although low power and low energy consumption are always desirable, they do not always have the same relevance. For instance, energy efficiency may be a dominant hard requirement while achieving the required power efficiency may not need a significant effort.

Thus, although energy optimization and power optimization are usually connected, they are separated problems that require different solutions. In general, this distinction leads to two different concepts: energy awareness and ultra-low power design. Both concepts must be clarified before going into further details [PR02].

2.3.1 Ultra-low Power Design

In every electronic system, there is always a maximum power that can be drawn by its power supply. The power supplies of the embedded systems that typically integrates Wireless Sensor Networks are very restricted, and therefore the maximum power drawn is very low.

For the system to work properly, overall power consumption must be kept under the maximum. This may include using a power management system which controls that not all subdevices are active at the same time. The strategies to keep power consumption below a threshold are part of so called ultra-low power design. Failing at ultra-low power design may lead to system lack of reliability and malfunction, e.g. brown-out resets.

In distributed systems, there is an additional complexity level that must be handled by designers. When partitioning the distributed system into different machines, it is crucial to distribute processing workload and system intelligence so that the power supply units of the corresponding embedded systems are sufficient to carry out the assigned tasks.

2.3.2 Energy Aware Design

Energy aware design consists on trying to achieve the best possible management of a limited energy budget. Limitations on energy budget are typically given by the lifetime of a battery.

Energy aware design becomes critical in battery powered devices which have to run for long periods of time. This is typical in mobile or ubiquitous systems, where connection to the grid is not available.

In addition, although energy harvesting permits gathering energy from the environment, which can be theoretically done for indefinite time, the flow of energy consumption must not be greater than the flow of energy scavenging, and therefore, energy awareness is required too, in order to meet the performance requirements.

Furthermore, energy awareness concept has to be extended for distributed systems. Distributed systems are typically fault-tolerant and redundant. Therefore, a node with depleted batteries might have no impact in the overall distributed system operation and performance. On the other hand, the behaviour that extends the lifetime of a single node, might reduce the lifetime of the distributed system. As a result, a node must not only be aware of the energy it consumes during its own activity, but also of the impact its own activity has in other nodes of the distributed system. For instance, in a multi-hop network, the frequency with which a node reports a sensed value not only conditions its lifetime as a sender node, but also the lifetime of the transit nodes that forwards the message to the sink node.

Consequently, the problem of energy awareness acquires a new dimension when applied to distributed systems. Local system optimization is insufficient and might even be detrimental when trying to optimize a distributed system. New techniques have to be developed to address this new difficulties.

2.4 Power Consumption Components

Power consumed by electronic devices can be divided into a dynamic power component dissipated during switching due to load capacitance, and a static power consumption component, which is consumed by leakage currents:

$$P = P_{dyn} + P_{sta} \tag{2.1}$$

Equation 2.1 shows these two components: static or P_{sta} and dynamic P_{dyn}. The fundamental differences between both components and the different implications they have in energy efficiency deserves some further explanation and analysis.

2.4.1 Dynamic Power Consumption

Dynamic power consumption is defined by Equation 2.2, where A is the activity factor, C is the capacitance loaded when switching, V is the supply voltage and f is the clock frequency. The activity factor (A) represents the fraction of gates that actually switch.

$$P_{dyn} = ACV^2 f \tag{2.2}$$

Dynamic power consumption is absolutely related to circuit activity. Examining the equation, dynamic power is reduced by:

- Technology improvements, with lower capacitances and lower supply voltages.
- Voltage and frequency scaling, with the consequent impact in system performance.
- Optimization to reduce unnecessary switching, which works for specific logic but becomes infeasible with complexity and unpredictability.

Although there are means to improve efficiency of dynamic power consumption which, eventually, will improve energy efficiency, dynamic power is directly related with system performance, i.e. the energy spent in dynamic power is energy spent for some functionality. Whether that functionality is really needed or not has to be decided by the high level designer.

2.4.2 Static Power Consumption

Static power consumption is calculated according to Equation 2.3, where V is the supply voltage and I_{leak} is the sum of all leakage currents in the circuit.

$$P_{sta} = V I_{leak} \tag{2.3}$$

Unlike dynamic power consumption, static power is not related to system performance. Static power is consumed even when the circuit performs no activity at all. In these cases, energy consumption can be theoretically reduced to zero by switching off the voltage supply to these circuits. However, while switching off the supply voltage, the state of this circuits is also lost, which might be undesired. State can however be retained by using additional flip-flops.

Static power consumption used to be negligible in comparison with dynamic power consumption. However, technology advancements have reverted this tendency [KAB+03] and static power consumption component is becoming a major contribution, as it can be seen in Figure 2.4.

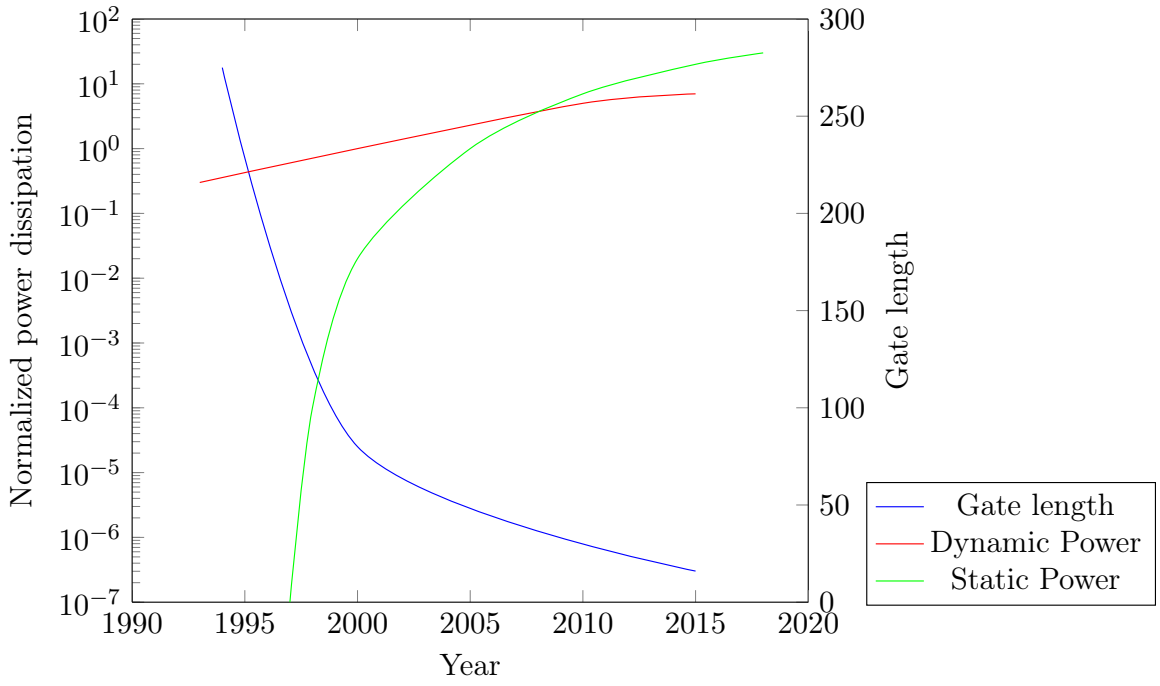

Figure 2.4: Static and dynamic power dissipation along years and gate length [KAB+03]

2.4.3 Static vs Dynamic Considerations

If we consider that a system has to perform, sooner or later, some specific tasks, reducing power consumption at the expense of performance will not reduce energy consumption.

According to ultra-low power and energy aware design concepts, discussed in Section 2.3, reducing power consumption at the expense of system performance, is a valid approach for ultra-low power systems design, as it will keep power consumption levels below the maximum. However, this would not be acceptable from an energy aware point of view. Reducing performance would increase the time the circuit is supplied with voltage, and because of static power consumption, the energy consumption would increase as well.

Hence, in these circumstances, optimization in an energy aware scenario, requires reducing the time in which independently voltage supplied circuit parts are switched on. This can be improved by using the so called "Run Fast then Stop" power management policy, and by increasing the control over the voltage supply of different parts of the system. Within the same System-on- Chip, this is currently done by power gating different defined power domains.

2.4.4 Power Consumption Proportionality

The first strategy that arises when looking for energy awareness is to use the most power efficient components. However, in most cases energy inefficiency does not lie on the energy spent while performing some task.

Most devices reach their maximum energy efficiency while they are fully utilized. On contrast, most energy inefficiencies can be found when the device is partially utilized or idle. Therefore, rather than optimizing an already ultra-low power system, energy awareness is mostly better

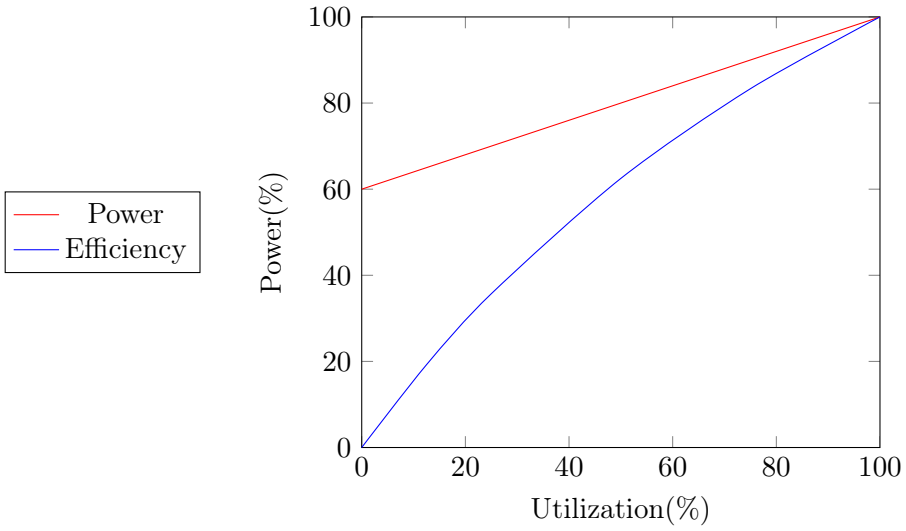

Figure 2.5: Power efficiency and relation between power consumption and system utilization.

achieved by improving the power-proportionality of the system and by combining this with intelligent power management policies, so that the system consumes energy while performing some tasks, but sleeps in very low power modes when there is no task to be performed.

Figure 2.5 shows the power/utilization ratio of a system (in red) and its power efficiency (in blue). High efficiency is only achieved when the system is almost at full capacity. However, when the system utilization is very low, power consumption is still over 60% of its maximum power.

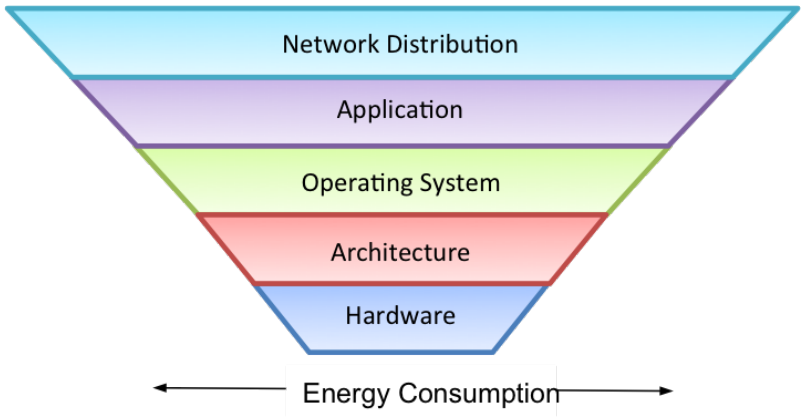

Figure 2.6: System energy efficiency can be represented as an inverted pyramid.

To achieve power proportionality, the system must integrate a power management system with enough granularity to switch off all the subsystems that are not being used. System architecture is directly involved here. Having different subsystems for different tasks increases the granularity, and therefore, even when the sum of all these subsystems might be more power consuming, being able to switch them off becomes more energy efficient.

It can be observed, that this behavior is analogous to static power consumption of electronic circuits, discussed in Section 2.4.2, but at a different level of abstraction. This evaluation of unused resources has therefore to be done at all possible levels.

2.5 High Level Energy Optimization

Energy efficiency is a cross layer system requirement. Every system design layer has its piece of responsibility in the global energy consumption of the system. The lower layers provide the resources to the higher layers that can then be as efficient as the layers below permit them to be. The whole schema can be seen as an inverted pyramid, shown in Figure 2.6.

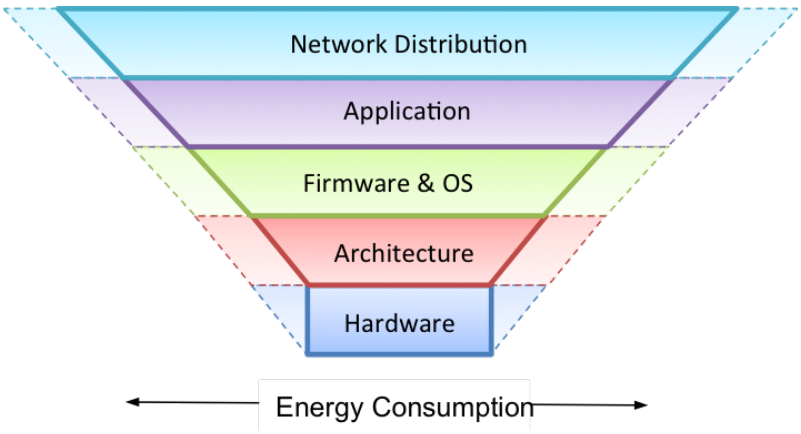

Figure 2.7: Resulting inverted pyramid after ideal hardware energy optimization.

In the state-of-the-art, as the analysis performed in Chapter 3, will reveal that most efforts in improving energy efficiency correspond to power optimization at the hardware level. However, the hardware layer is only partially responsible of the overall energy consumption, as it is represented in Figure 2.7. Even with the optimum hardware, if the design layers above it are not efficient, the overall efficiency of the system would be very poor. For instance, a optimum hardware design is useless if the software on top of it makes use of resources with no judgement. And the same happens with the network and distribution layer. A node might be very efficiently designed, but if a poor arrangement and decisions in routing algorithms lead to routing loops, batteries will deplete very fast and the system lifetime will end sooner than expected.

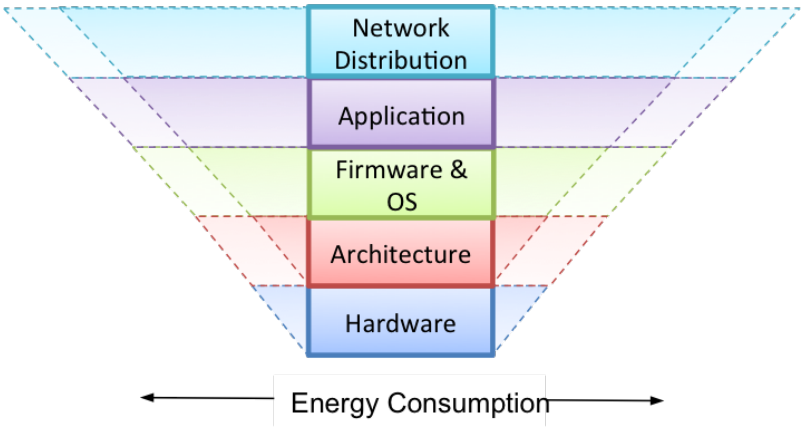

Figure 2.8: Ideal cross-layer energy optimization.

In recent years, specially with new multi-processor technologies and the arrival of smart-phones and other technologies that require energy awareness but also very high system performance,

many improvements have been made at the architectural level. However, in a distributed system integrated by spatially distributed embedded systems, the room for improvements in design layers above the hardware layers, either at the component level or the architectural level, is still considerable. Figure 2.8 depicts the ideal energy optimization of these systems.

The state-of-the-art analysis also revealed that there is a lack of metrics and semantics to provide energy consumption information at higher levels of abstraction. This could be the reason why high level energy optimization is still mostly unexplored and unexploited.

In next chapters, the approach and implementation of semantic infrastructure to provide high level energy awareness will be presented and thoroughly described.

3 Related Work

Relevance of embedded systems has substantially grown in recent years. The development of wireless technologies, the wide range of potential applications and the challenges to overcome have attracted the attention of researchers in the last decade. Research has been performed within different application paradigms, such as Wireless Sensor Networks, Internet of Things, Ambient Intelligence or Cyber-Physical Systems.

Together with all these application paradigms, plenty of design methodologies and modelling approaches have been proposed, in an attempt to overcome all the new constraints encountered by this kind of systems.

This chapter aims to present the situation and currently acknowledged challenges of design and modelling of cyber-physical and wireless embedded systems. As already discussed in Chapter 1, energy simulation in cyber-physical distributed systems requires considering all system levels, from hardware to network, and therefore the very different approaches to holistic system modelling are introduced. From them, most significant simulators are highlighted and discussed, pointing out their strengths and weaknesses, which typically vary depending on their use cases.

Hence, the first question to be answered is what kind of modelling approach might be more suitable to undertake the simulation of energy in distributed cyber-physical embedded systems fulfilling the required multi-domain and multi-level modelling.

After finding out which modelling approach and implementation option best fits the purpose of this research, it is necessary to present and evaluate the state-of-the-art in power and energy simulation, taking into account not only the different mechanisms to account power consumption, but also considering how that information can be processed or appropriately encapsulated in order to provide energy awareness at all possible optimization levels. This analysis will enable a better understanding of the current status and specific challenges of the high level energy consumption optimization problem.

3.1 Design and Modelling Challenges

The development of wireless communications transformed the connectivity of embedded systems. Embedded systems started communicating forming networks. Moreover, the advent of highly interactive embedded systems leads to a completely new approach to the design problem.

First wireless sensor networks had no real interaction with the environment. They were not real cyber-physical systems. Although sensors provided a communication channel between the physical world and the computational system, the system did not have the capability of affecting the environment. However, these systems already supposed a challenge from the modelling point of view, because the environmental conditions already had an impact in the system performance.

In Cyber-Physical Systems the interaction fully determines the system behaviour, and therefore, the environmental conditions have to be considered as part of the system itself. As a result, the most basic assumptions have to be revisited for these highly interactive cases, starting from the definition of system itself. The convergence of hardware, software, network and physical processes into a single system settles a completely new field of study.

Figure 3.1 represents a generic cyber-physical system consisting of different embedded systems devices. The figure shows the heterogeneity of the hardware/software platforms, which might differ in hardware, but also in software, as they have to be highly optimized in order to achieve optimum energy consumption. Furthermore, high interactivity can also be observed, as every node has wireless connectivity with other nodes in the network and every node has inputs and outputs to its physical environment.

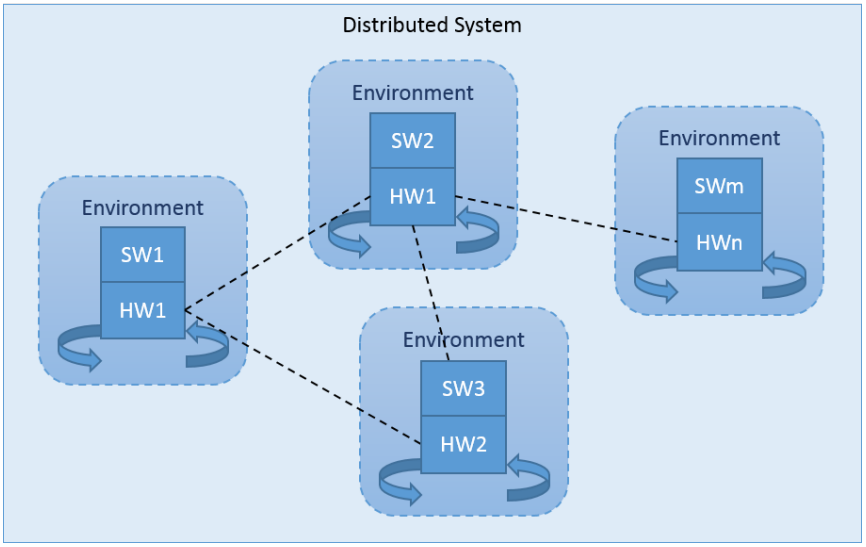

Figure 3.1: Distributed cyber-physical embedded systems run over different optimized hardware/software platforms and are highly interactive with both their physical environment and neighbour nodes.

Design and modelling challenges have been enumerated and discussed in several articles. However, there are still no widely accepted solutions.

3.1.1 Design Challenges

Design challenges start with the new requirements for cyber-physical embedded systems, which are significantly different from those of traditional embedded systems. Edward A. Lee already discusses the need of improving reliability and predictability when dealing with the physical world, as well as the problem of having timing deadlines [Lee08b].

Another crucial challenge, which has been addressed has already been described for Mobile Ad hoc Networks (MANETs), is the need to perform cross-layer optimization, in order to fulfil some requirements, such as those related to security or energy management [CMTG04].

Hence, main design challenges can be summarized in the following key points:

- **Predictability:** Embedded systems have been traditionally computer systems included in larger systems for a dedicated and specific task. The specificity and the restricted interaction possibilities of this kind of systems have made them traditionally very predictable and reliable. On contrast, in Cyber- Physical Systems, a lot of external actors are interacting with the system and, as a consequence, the aforementioned predictability is lost.
- **Real Time Constraints:** The different actors that converge in Cyber- Physical Systems interact among them in a concurrent way, which in practice provides the system with real-time computing constraints and characteristics. The system must react to external stimuli within a certain deadline, otherwise the response might become useless.
- **Dependability and Resilience:** Complexity of this kind of systems, with many spatially distributed components, some of them even inaccessible, that might need to operate along several years, makes of dependability and resilience a major challenge.
- **Cross-Layer Requirements:** Some specific requirements cannot be addressed within one single abstraction layer. However, considering all abstraction layers at once in such complex systems becomes a challenging task. Major examples are:
  - **Security:** Security in such open systems, affected by external stimuli, is a significant challenge itself. Security must be planned considering all layers of abstraction. It would be useless, for instance, to have very secure wireless communication, with the consequent resources requirements, if the hardware is exposed and easy to access.
  - **Energy Consumption:** An energy efficient system would require efficient hardware, software and communication. If any of this fails, the whole energy efficiency of the system will be compromised. This is the central topic of this thesis and therefore this work will focus on the energy consumption cross-layer problem characterization and solution. However, the energy problem is strongly related to other challenges, such as dependability or real-time constraints.

3.1.2 The Modelling Challenge

Cyber-Physical Systems do not only present new challenges from a design perspective. New design challenges are also translated into a challenging Cyber- Physical Systems modelling approach, which has also been already extensively discussed.

For instance, the importance of model-based design and model driven development has been pointed out, as well as how cyber-physical embedded systems modelling demands much more than current modelling languages and frameworks can offer [DLV12].

According to the tools used for developing the simulations, distributed cyber-physical embedded systems are the convergence of several modelling areas: digital and analogue hardware, software, network and communication, and physical processes.

There are plenty of tools available for all areas. For the first cyber-physical applications, in which the function of additional domains are very rudimentary, their effect could even be modelled using the available tools. However, complex systems with great interaction among the different domains require the combination of different modelling paradigms.

A combined model could be achieved by using different tools independently. Nevertheless, to evaluate combined effects synchronization and simulation coupling is required. Moreover, for a real optimization of the whole system, the interdependencies between all areas have to be explored, and an integrated solution would be desirable.

The need of simulating models of very different natures concurrently is a major challenge from the modelling point of view, not only in the practical implementation of simulators but also from a modelling theory perspective.

In spite of the complexity of modelling these kind of systems, models are still crucial for cross-layer optimization, like energy optimization, which is the focus of this thesis.

3.2 Modelling and Simulation of Networked Embedded Systems

In the last decade, plenty of simulators for networked embedded systems have been developed. The use of simulation in this field has been a constant due to the uncertainty faced by designers regarding system's feasibility or recreation of the environmental conditions the system will encounter once after deployment.

Networked embedded systems are the convergence of different technologies. As a consequence, simulation has been approached from different perspectives. Most of them can be encompassed in the following three main groups:

1. **Network simulators:** The first steps of networked embedded systems were driven by the development of the network protocols that would make possible the communication among the devices. For this reason, some of the first networked embedded systems simulators were extensions of network simulators.

2. **Virtual Prototyping:** Virtual prototyping is common practice in today's embedded systems development. Using modelling and simulation, a virtual platform is created that can be used to enable hardware and software co-design, so that embedded software development process can start before the hardware is manufactured. For networked embedded systems, the virtual prototyping approach has to be enhanced to consider network communication and physical processes that play a crucial role in system behaviour.

3. **Multi-Domain Simulation:** Finally, the third group of simulators are those motivated by the necessity of coupling different modelling paradigms in order to simulate heterogeneous and cyber-physical systems.

All three approaches will be analysed, through some examples, in the following subsections. Figure 3.2 summarizes the capabilities of the different simulators and frameworks mentioned next.

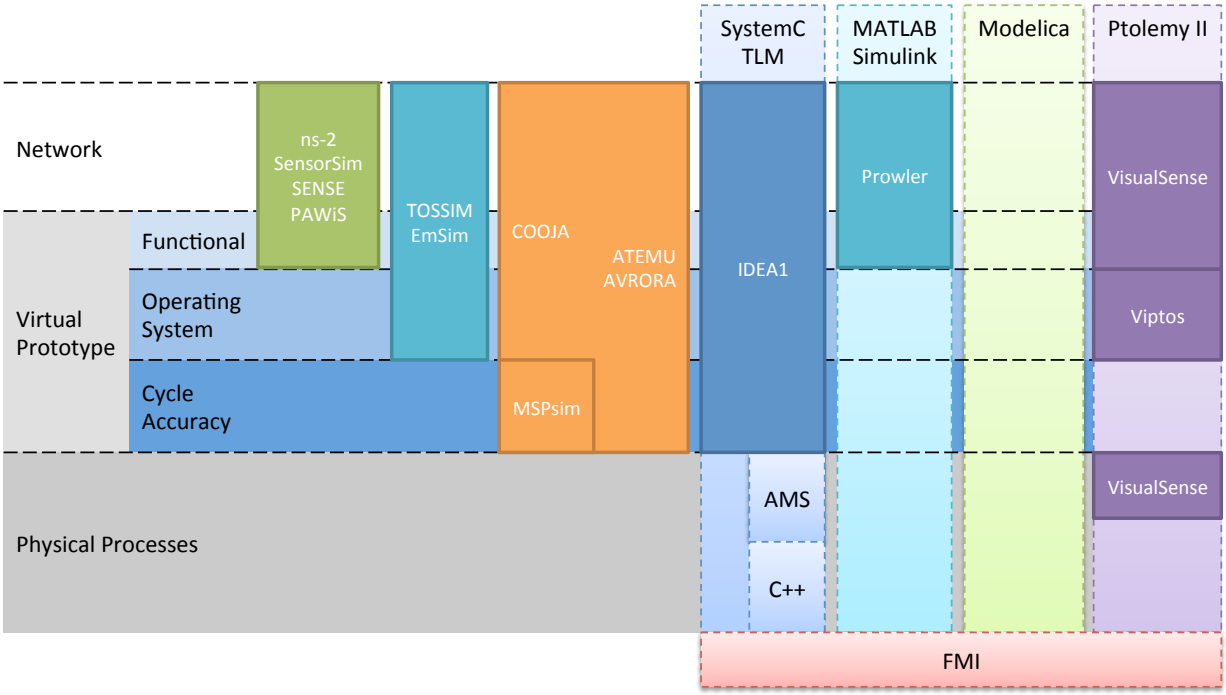

Figure 3.2: Summary of simulators and frameworks for networked embedded systems

3.2.1 Network Simulators

In networked embedded systems, the network domain is one of the most relevant parts of the simulation. Network domain is also one of the domains that have been traditionally modelled and simulated separately from hardware/software because of the wide repertoire of already existing network simulators. However, in the last decade, plenty of simulators for networked embedded systems have been developed.

The need for an integrated simulation solution, for instance in Wireless Sensor Networks (WSN), contributed to the proliferation of simulators which combined network and hardware/software models. However, the approaches where varied, depending on what kind of simulation served as the starting point. Haase et al. [HMD11] provide a full overview of different wireless sensor networks simulators, as well as some extensions focused on power estimation.

Some of the first distributed embedded systems simulators were motivated by the need of a network model to explore new applications and algorithms. For that purpose, network simulators where used, following a top-down approach, starting from the network high-level model, which gradually incorporated more and more detailed models which provided the required information.

The first approach to embedded systems networks simulation were extensions to model ad-hoc mobile networks, being among the most representatives those developed in Monarch project [Joh99] or GloMoSim [ZBG98] and its commercial version QualNet [Teca]. These extensions evolved into more specific tools, designed, for instance, for WSN applications.

These simulators started mainly as an evolution of general purpose network simulators, such as ns-2, although specific wireless sensor network simulators were developed later.

3.2.1.1 General Network Simulators

ns-2 [Ins] was a very popular free discrete event network simulator based on REAL simulator [Kes88]. However, ns-2 was not optimized for wireless ad-hoc networks. It suffered from severe scalability problems, as WSNs are usually large networks, reaching even more than thousand nodes. For instance, whenever a node sent a message, all the other simulated nodes received the signal, even when the signal strength was so low that the influence in communication was negligible: neither can these signals be received by those nodes nor contribute to the received noise.

In order to overcome some of the ns-2 constraints, some improvements started to appear. For instance, a truncation algorithm was proposed by Naoumov et al. [NG03], which consisted in preventing the simulator from modelling signal reception in those nodes that were too far away, making the ns-2 simulation much more scalable.

The first documented sensor networks simulator, **SensorSim** [PSS00], was based on the ns-2 simulation core. Due to the high level abstraction of ns-2, SensorSim added specific models required for sensor networks:

- **Power models:** SensorSim was already concerned about power consumption as a restricting factor for WSNs deployment. Therefore, it added power consumption models for the hardware components.
- **Software- and Hardware-in-the-Loop:** ns-2 applications are mostly generic traffic generators. SensorSim permitted simulating real applications as well as interacting with real nodes (hardware-in-the-loop).
- **Graphical User Interface (GUI)**

Unfortunately SensorSim was soon discontinued and is not available any more.

While independent from ns-2 and not compatible, there is a new simulator intended to be a new improved version and eventual replacement of ns-2, named **ns-3** [HRFR06]. In addition, the most important commercial general network simulator is **OPNET** [Tecb], which also includes a wireless library.

The best advantage of ns-2, ns-3 and OPNET tools is the amount of protocol implementations which already exist. Whole protocol stacks can be reused in new projects.

However, in distributed embedded systems, network simulation is only a small part of the whole simulation efforts. Infrastructure for easy, fast and accurate modelling of hardware components is crucial too. The lack of such an infrastructure restricts these simulators to those applications that require only high functional or behavioural level models of the hardware platform.

Difficulty to simulate detailed hardware models also thwarts accurate power and energy simulation.

3.2.1.2 Specific Mobile Ad hoc Networks Simulators

General purpose network simulators presented scalability problems for large wireless and mobile ad-hoc networks. Moreover, they lacked system models for simulating sensor nodes. To overcome these deficiencies, specific ad-hoc network simulators were developed, which typically included some specific models for radio propagation, sensors and hardware systems, as well as interfaces for hardware-in-the-loop simulation.

Most of them followed a similar approach. They were built on top of component- based frameworks and made use of event-driven simulation. Component-based design is more suitable for network simulations, as all interaction is captured by interfaces, so that interdependence is restricted. As a result, components are more reusable and extensible than ordinary objects.

The first example of this kind of simulators is **J-Sim**, previously known as JavaSim, which is still a general purpose network simulator and closely related to ns-2, but it is already component-based and has specific extensions to model radio propagation, hardware systems and interfaces for network emulation [SHK+06]. It is developed in Java and based on the SensorSim simulation framework described before (see Section 3.2.1.1).

The sensor simulator part of J-Sim differentiates between three types of nodes:

1. **Sensor nodes:** Gather the information from the environment.
2. **Target nodes:** Pre-process and encapsulate the information into a message.
3. **Sink Nodes:** Process and consume the information.

Power models are provided for both energy sources (e.g. batteries) and energy-consuming components (e.g. transceiver). Energy consuming components are modelled as finite state machines based on some standard operation modes, which are the same for all subsystems. Another key feature of J-Sim is that it allows simulating mobile nodes. It includes several propagation models to estimate signal power loss. Nevertheless, it does not model noise.

J-Sim supports network emulation, where protocols can be tested in a real environment, while other components are executed in a virtual environment, i.e. using a testbed with real nodes in laboratory to execute some tasks while the rest is executed in the simulation. This hardware-in-the-loop is done in both top-down and bottom-up directions. In the former case, a socket layer is provided which replicates the interfaces of the actual operating system, so that the application communicates with the virtual environment in the same way it communicates with the operating system. In the latter case, real packets are intercepted and translated into J-Sim simulated packets.

J-Sim, is still too generic and difficult to use. As an attempt to compensate this difficulty, a module library has been developed. However, in spite of having components for power and sensor modelling, the only wireless MAC protocol distributed with J-Sim is IEEE 802.11, while this is not even the main option in WSNs, where protocols with less power requirements, like IEEE 802.15.4, are of great importance.

SENSE (SEnsor Network Simulator and Emulator) [CBP+05], is built on top of COST (Component Oriented Simulation Toolkit) [CS02]. COST is a component-based general purpose discrete event simulator.

In order to improve performance, SENSE exchanges the pointer to a message instead of the actual message. This way, all nodes receive the same message instead of one copy, and much less message allocation is required. The message is not supposed to be modified during a transmission (except from noise, delay and distortion, which are modelled separately), so that sharing the same message is safe. The only problem is deallocation, which is achieved through a reference counter which tracks the number of components referencing the message. If the counter reaches 0, the message is deleted.

SENSE includes a rudimentary propagation model that considers two options:

1. A message is fully transmitted (without errors) to all the nodes within the stipulated range.
2. A range is defined in which the 100% of the messages are always fully received. Outside this range, the probability of receiving the message decreases linearly with distance.

There are also predefined components for battery models, applications, routing algorithms and MAC protocols [PCN]. However, as in J-Sim, the MAC protocol included is IEEE 802.11, which is not the most suitable standard for power aware sensor networks.

While more focused in WSN than other general purpose simulators, and in spite of being easier to use, SENSE still has some of the most significant deficiencies found in J-Sim.

- Radio propagation models are very rudimentary and do not include models for noise and interference.
- Module library does not include implementations of common protocol and standards in WSN, such as IEEE 802.15.4.

These specific problems are addressed in **PAWiS (Power Aware Wireless Sensors)** [MGH05]. OMNeT++ is a component-based discrete event simulation library for building network simulators. PAWiS is one of those simulators, focused in power aware WSNs.

The PAWiS project consists of several elements: a framework, a module library and a visualization tool. The PAWiS Framework models the physical layer and includes a propagation model. This propagation model simulates attenuation due to distance as well as interferences and noise. Dynamics of the network can be simulated at runtime through Lua scripting language [IFF96].

The module library consists of implementations of specific devices and protocols using the framework. It includes not only simple examples but specific power aware algorithms, such as S-MAC [YHE02], CSMA-MPS [MB04], and hardware models, like Chipcon low power transceiver CC2420 [Chi07].

Simulation's graphical user interface (GUI) is the one provided by OMNeT++, which permits the inspection of the different elements at runtime, but also makes the simulation very slow. Anyhow, GUI can be simplified or even deactivated for higher speed simulations.

PAWiS includes a visualization tool that renders power consumption of the different hardware subsystems. Power consumption models are based on finite state machines. CPU execution time estimation is left in hands of the system designer, which provides some parameters about his algorithms, which then are computed to get the estimation of the number of CPU cycles required

to execute them. However, there is also an extension which provides time annotation, increasing the accuracy of the application simulation [MHSM10].

PAWiS Framework is not an extension of OMNeT++, as a general purpose simulator, but takes advantage of OMNeT++ core classes, such as modules, message passing infrastructure and discrete event simulation kernel, to build a specific WSN simulator. The module library includes simple implementations with basic functionality. This makes PAWiS easy to use, as the user can always start with a basic working simulation and progressively extend it.

However, PAWiS had some inefficiencies that restricted its usage for accurate networked embedded systems simulation:

- Unlike SENSE, message transmission is not modeled as a pointer. Messages need to be replicated for each peer to peer transmission, which in dense networks leads to huge memory allocation.
- OMNeT++ core is designed for network modelling and runs into granularity problems when using it for hardware modelling.

In general, network simulators could be used efficiently to design and evaluate the network aspects of networked embedded systems. However, accurate simulation of hardware components is extremely difficult without specific semantics and support of hardware modelling languages. This infrastructure is provided by virtual prototyping tools, which are evaluated next.

3.2.2 Virtual Prototyping

Embedded systems are conceived, by definition, with a specific application in mind. Considering the application already at design time enables a level of optimization which is not possible otherwise. Therefore, embedded systems are typically used for those applications that demand more optimization.

Software has to be developed for its correspondent hardware platform. However, as the hardware platform is designed and optimized for a specific purpose, it is not available until the whole hardware design and manufacturing process is finished.

However, the time-to-market requirements do not permit developing software and hardware sequentially. Both hardware and software must be developed in parallel. In order to improve time-to-market and productivity and reduce the risks, the use of virtual prototypes is a common practice in embedded systems design. As software and hardware are typically developed by different teams, simulated prototypes are intended to provide the awareness required to increase optimization and to uncover any failure that would otherwise arise at the end of the design process, during hardware and software integration phase.

As depicted in Figure 3.3, hardware and software design processes can remain closer by using virtual prototypes, which facilitates the integration phase. Although creating the virtual models requires a big effort, these virtual models limit the uncertainties in the integration process, which might cost even more time and effort.

Therefore, virtual prototypes and model-based design methodologies are common practice in embedded systems design, and, subsequently, they are widely covered in literature [Ern98] [VNPJ96] [SRMB98].

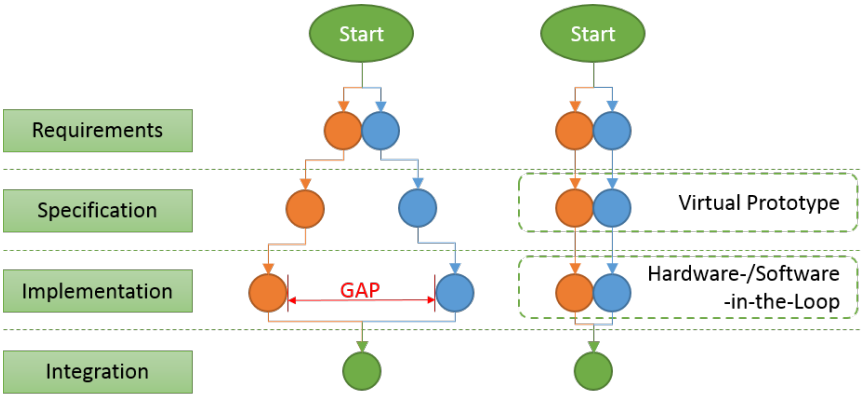

Figure 3.3: Using virtual prototypes, permits closing the gap between hardware and software design processes.

In Networked Embedded Systems, the specific embedded systems to be used are chosen or even designed at the same time as the whole distributed system. This fact restricts the usability of network simulators, as discussed in previous section. However, virtual prototypes of isolated nodes are not sufficient to design and optimize Networked Embedded Systems. In order to test the software appropriately and to validate the hardware requirements, the network and the influence of the physical environment have to be modelled as well.

Thus, there are different options for virtual prototyping of sensor networks, with some additional extensions, i.e. simulation models, specially network. There are mainly two approaches: some are conceived as operating systems simulation platforms, while others are hardware emulators.

3.2.2.1 Operating Systems Emulation

With the development of different hardware platforms and operating systems, it became necessary to test applications. However, migrating the code from simulations to the real node requires reprogramming and adapting the code to the final target hardware platform.

When specific embedded operating systems started to appear, new simulators associated to them were developed. The main focus of these simulators was to emulate those operating systems in an ordinary PC, so that final real-code applications could be tested with no need of the real hardware platform.

In order to test distributed applications, some of these emulators were extended so that other aspects, such as network, but also power consumption, could also be estimated using them.

TinyOS

One of the most renowned operating systems for wireless sensor nodes is TinyOS [HSW+00]. **TOSSIM** [LLWC03] is a simulator for it, and it is included in the TinyOS distribution package.

The main purpose of TOSSIM is to develop and test TinyOS applications in a simulation environment without installing them in the actual node hardware. Those applications are written in nesC, an extension of C programming language specially designed for sensor networks [GLB+03]. TOSSIM permits simulating a network of thousand of nodes running the same application.

In order to be able to simulate many nodes efficiently, they are all simulated in the same process. During compilation, variables are stored in arrays, where each element belongs to a specific node. Simulations of many nodes are therefore feasible without having to execute many virtual machines. However, there is the drawback of not being able to simulate heterogeneous networks, with different node architectures, operating systems or applications.

Radio propagation environment is modelled as a statistical process. TOSSIM includes an ideal environment model and a directed graph of bit error probabilities. The simplicity of the radio model increases scalability, but details about network behaviour are lost.

TOSSIM does not capture energy consumption itself. However, there are extensions for that purpose, such as PowerTOSSIM for TinyOS v1.x [SHC+04] and PowerTOSSIM 2 or PowerTOSSIMz for TinyOS v2.x [PVS+08] [PCC+08].

It is very common in distributed cyber-physical systems, to have different applications in different nodes. It is part of the system optimization to reduce the application as much as possible, and therefore, specificity is usually preferred over flexibility. As a result, not being able to simulate different applications running concurrently on different nodes is a significant restriction for using TOSSIM for modelling the whole networked system. **TinyOS Scalable Simulation Framework (TOSSF)** [PN02], is a TinyOS simulator, that broadens TOSSIM scope beyond application verification, increasing its scalability and improving the environmental models.

TOSSF is based on two previous mobile ad-hoc networks simulation tools: Dartmouth Scalable Simulation Framework (DaSSF) [LN01] and Simulation of Wireless Ad-hoc Networks (SWAN) [LPN+01].

Unlike TOSSIM, it permits simulating heterogeneous networks, with different nodes, which may differ in architecture, operating systems and applications.

It includes also more sophisticated radio propagation and environmental models. Environmental models recreate the physics and metrics that stimulate the sensors. Therefore, a more realistic network and application behaviour is achieved, in comparison with TOSSIM.

Unlike TOSSIM, it does not have specific extensions to estimate power consumption.

Linux

Many embedded applications run over Linux operating system. **EmSim** is a pure simulation environment which is provided as part of Emstar software environment [GEC+04]. Emstar is a framework to develop WSNs applications that runs over Linux. It provides a set of useful interfaces at very different levels to make the development of applications easier. It can understand even some TinyOS applications through EmTOS [GSR+04], which translates the TinyOS system API into the Emstar API.

As TOSSIM, EmSim enables real-code simulation [GRE+07]. When used in pure simulation mode, EmSim provides a radio channel simulator and a sensor simulator.

An advantage of EmSim is that it supports an emulation mode, which allows hardware-in-the-loop simulations, i.e. using real hardware, such as radio transceivers or sensors, while running the other elements in a simulation machine.

The EmCee variant provides interfaces for using real RF channels rather than simulated radio propagation models, which are very useful when the environment of the application scenario is

easy to recreate and propagation and sensed physical magnitudes can be tested with realistic values.

However, in general, in cyber-physical systems, environmental conditions will be unknown and therefore hard to reproduce in laboratory. As a result, real hardware values may not be closer to deployment values than simulated ones. In fact, hardware-in-the-loop is typically used in the opposite direction, due to the impossibility of testing the system in the final deployment conditions: the hardware/software platform is tested with interfaces to simulated environmental conditions, including radio propagation models.

Another difference regarding TOSSIM is that EmSim offers interfaces for heterogeneous simulation, which means that it has the ability to simulate nodes with different hardware architectures or operating systems, in the latter case by wrapping the Mote code in the already mentioned EmTOS tool.

Contiki

Another operating system for sensor nodes is Contiki [Ad]. There is a simulator for it, called **COOJA** [ODE+06]. It is implemented in Java. Although COOJA is really flexible and permits cross-level and heterogeneous simulation, it is primarily a Contiki emulator.

Unlike TOSSIM, the best advantage of COOJA is its flexibility, which permits executing simultaneous simulations of nodes at all the network, operating system or hardware levels.

Apart from operating system emulation, where real Contiki code can be tested, COOJA permits high level Java node models, with network functionality and coarse granularity.

A general problem of operating system emulators is that they are not able to capture low level events that are very necessary for energy consumption estimation. To overcome this, COOJA includes MSPsim, an instruction set simulator, to simulate cycle-accurate models [EOF+09]. Through this hardware emulation, simulation of nodes using operating systems other than Contiki, such as TinyOS, becomes also possible.

Communication between COOJA and the compiled event-driven Contiki kernel is done through Java Native Interface (JNI) calls.

Concerning the propagation models, custom models can be easily plugged in in COOJA. In the distribution, a unit disk graph model is included. There are also extensions for ray-tracing based radio medium model [Ö06], in order to model obstacles, and a radio interference simulation model [BROV11].

All this flexibility makes of COOJA a very powerful simulation environment, with the possibility of modelling heterogeneous networks and creating cross-level simulations, in order to evaluate WSNs in all network, operating system and hardware levels. The main drawback of such flexibility is the effect of maintaining so many interfaces in scalability, in terms of the number of nodes that can be simulated, and in the overall simulation efficiency.

Although theoretically flexible, COOJA and Contiki lack versions for some embedded platforms and architectures. This restricts the options for the designer, who has to go for Contiki operating system and the architectures supported by both the operating system and the COOJA simulator. ARM Cortex microcontrollers, for example, are not supported.

This problem can actually be extended to all operating system emulators, as all of them are restricted to the list of platforms they support, which in most cases is far from covering the wide spectrum of embedded microcontrollers.

Operating systems emulators have several constraints when used for networked embedded systems design and optimization:

- Different operating systems may have different performance and should therefore be evaluated as part of the optimization process.
- Optimized ultra-low power embedded systems might not even use a real operating system, but a small library or framework with a simple scheduler.
- A theoretical advantage of operating system emulators is that it can run on any platform running the operating system. However, in practice it becomes a constraint, as the number of supported platforms is very restricted, and specially it does not typically include the most recent hardware architectures.
- Operating system emulators are not capable of capturing low level events that are required for accurate energy and power consumption estimation, specially on the microcontroller. A microcontroller emulator has to be used in combination with the operating system emulator, as it is done in COOJA, in order to get this data.

3.2.2.2 Hardware Emulation

Hardware emulators are used to improve hardware and software co-design, so that the software design task can be started before the hardware platform is manufactured, reducing time-to-market. They are capable of executing machine code in a simulated environment, so that the simulated software can be directly used in the target hardware platform.

Hardware simulators offer a wide variety of approaches with different accuracies, ranging from complete cycle-accurate emulation to transaction level models. Selecting the appropriate approach would depend on the application requirements.

Unlike network simulators, which follow a top-down approach, in this case, the process is reversed toward bottom-up system modelling, starting from a detailed hardware model, and extending it with communication and propagation models in order to simulate the network. These opposite approaches are shown in Figure 3.4.

Hardware emulators can address some of the deficiencies of operating system emulators:

- A microcontroller emulator permits executing whatever operating system which is compatible or even bare-metal software applications.
- It provides better granularity in order to obtain the energy consumption of the microcontroller, which is a major contributor in overall energy consumption.

An early example of this kind of simulators is **Embra** [WR96], based on SimOS [RHWG95], which was capable of emulating processors and caches. Similarly to what happened with some network

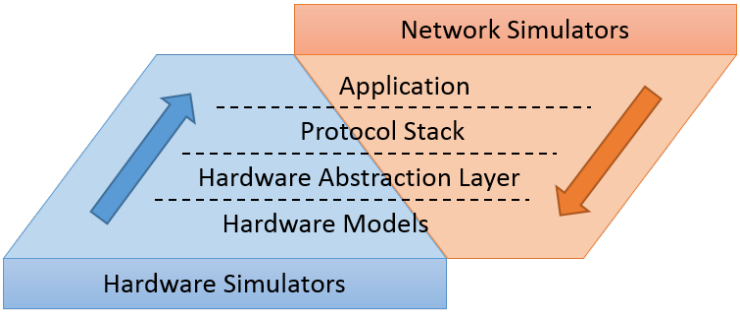

Figure 3.4: Bottom-up simulation approach based on hardware simulators versus top-down simulation approach based on network simulators.

simulators, some of these simulators were extended and became capable of simulating networked embedded systems, adding wireless communication and network models.

The first simulator to provide a low level cycle-accurate CPU model in the context of sensor networks was **ATEMU** [PBM+04]. While there were already emulators for processors, ATEMU was the first emulator tool actually focused on WSNs, and able of emulating several nodes composing a network.

Unlike operating system emulators, ATEMU depends only on the hardware it emulates, and therefore is not restricted to a specific operating system. It could even help to develop an operating system itself.

ATEMU includes an AVR emulation core, and is suited therefore to emulate nodes based on AVR micro-controllers, such as the MICA2.

ATEMU also includes a radio propagation model based on distance attenuation which takes into account interferences from other nodes, making the algorithm that keeps track of received power a n-squared algorithm.

Cycle-accuracy, together with the interference radio model, makes of ATEMU a very accurate model. However, this accuracy comes at the expense of performance, which has been estimated as 30 times slower than TOSSIM [TLP05]. Simulating such a detailed model is very costly and presents very serious scalability problems. Scalability can be crucial in Networked Embedded Systems, where networks might be conformed by hundreds or even thousands of nodes.

Motivated by the performance and scalability issues of ATEMU, another simulator based on an AVR emulator, named **AVRORA** was developed [UCL11]. AVRORA aimed to overtake ATEMU in simulation performance while keeping the cycle-accurate granularity [TLP05]. Therefore, it provides a cycle- accurate model of MICA and MICA2 motes. The last official release is from 2005 and supports ATMega128, ATMega32 and ATMega16 micro-controller models.

AVRORA radio model does not include noise simulation, but calculates interference by doing an arithmetic OR with bytes received (correctly synchronized). However, this approach is very inaccurate, and should take into account the modulation of the received signals. However, AVRORA, as most simulators, is extensible, and some successful extensions have been developed, like AVRORAz [PAP08], which allows emulation of the Crossbow MICAz mote and includes an indoor radio model.

In addition AVRORAz extends AVRORA to create IEEE 802.15.4 Standard [IEE06] compliant simulations [Cen11]. These extensions include the address recognition algorithm, frame acknowledgement, Link Quality Indicator (LQI) and Clear Channel Assessment (CCA), as well as the already mentioned indoor radio model.

Although the objective to create a cycle-accurate and scalable WSN simulator was accomplished in AVRORA, it is still 50% slower than TOSSIM [TLP05].

Apart from the effect on performance of cycle-accurate modelling, the main drawback of these emulators is their lack of flexibility. Such a degree of accuracy prevents them from being generic, and they have to be created for a specific hardware architecture. Hence, although these models are the only to provide fine granularity about hardware models, they are not practical enough for hardware architecture design, due to their lack of flexibility which prevents from easily replacing models in the simulation. Before using a cycle- accurate model, at least the micro-controller family must have been already decided.The Networked Embedded Systems simulation features are then built as an extension to these specific emulators and will therefore not support replacing the microcontroller emulation part.

However, Networked Embedded Systems are frequently heterogeneous networks where different nodes may have different architectures. A simulation platform that is not flexible enough to support simulation of heterogenous nodes is too restrictive.

Furthermore, microcontrollers are not the only energy consuming hardware subsystems within a network node. There are more hardware elements that contribute to power and energy consumption and have to be modelled in the same simulation framework.

In this context, hardware and software co-design languages, such as SystemC [Acca] or SpecC [GZD+00] can provide the required features to create networked embedded systems simulators:

- SystemC is a hardware modelling language and an industry standard.
- It is based on C/C++, that not only permits hardware and software co- desing, but also the development of extensions in a high-level powerful language.
- It provides an implementation which includes a discrete-event simulation kernel.

Therefore, hardware elements, apart from the microncontroller, can be easily modelled in a language that is known and standard. Network models can be developed in C/C++. **SCNSL (SystemC Network Simulation Library)** [DQ] is a good example of this, which uses SystemC to simulate networks, so that both system and network design can be modelled in a single tool [FQS08].

The propagation model is not very detailed. However, it is capable of calculating attenuation as a distance function and to detect collisions. SCNSL Framework has been extended in **IDEA1**, adding some protocol and hardware implementations and a graphical user interface (GUI) [DMN10a].

Estimating energy consumption required cycle-accurate models to be simulated in the SystemC-based environment. IDEA1 has been extended with an instruction set simulator that supports different microcontrollers, at least AVR and MSP430 [GNMO12]. This is a significant improvement regarding simulation of heterogeneous network with hardware level accuracy.

However, accurate models require disclosure of details that are not always provided by microcontroller designers. Instead, they provide their own models which are closed and might have problems to integrate into a broader simulation platform. Thus, it is complicated to obtain or develop accurate models of state-of-the-art microcontrollers.

3.2.3 Multi-Domain and Multi-Level Modelling

The difficulty of modelling complex heterogeneous systems has been a persistent problem in engineering. There is profuse literature on the field based on very different applications, control systems, electromechanics, fluid systems, etc.

The main problem is the synchronization between different models. Depending on the system to be modelled, there are several ways to model time. Most of them are based on two main methodologies:

- **Discrete-event:** Some systems only change their behaviour as a response to a certain event. If there is no event, the state of the system will remain static.
- **Continuous:** Other systems change continuously, and therefore, the simulation must continuously track them over time, in order to update their state.

Combination of digital systems and network models required extending the simulation tools and semantics, but the basic underlying simulation paradigm was in both cases a discrete-event simulation. However, some parts of a Cyber-Physical System will require in most cases continuous simulation.

Hence, modelling heterogeneous systems usually requires the combination of different modelling approaches. Different modelling paradigms can be combined by using different simulators. However, to have a concurrent model, interfaces have to be created and simulation coupling leads to bottlenecks and, consequently, to severe simulation inefficiency.

3.2.3.1 Models of Computation and Computational Models

In electronic systems design, the necessity of using different modelling paradigms has also been already addressed. Apart from the two main methodologies already described, there are more specific modelling paradigms which are optimized for different kind of models.

For instance, computational systems can be described using different models for different levels of abstraction. These models are called Models of Computation (MoCs) [Sav98]. Hence, a computational system could be described, for instance, through a logic circuit model or a state machine model, among others.

Therefore, it would be preferable to achieve the combination of different models of computation within the same simulation environment. An example of system which demands combination of different MoCs, is, for instance, an analogue mixed-signal system, which might require continuous time models to evaluate the analogue part, while the digital part can be simulated much more efficiently using event-driven simulation.

Furthermore, apart from modelling different parts of the system using different models of computation, it might be required to model the same part of the system using different MoCs. The appropriate MoC to be used can then be decided by the user, depending on the information he wants to extract from the model.

During the last years, several tools have explored this combination of different Models of Computation, with different MoCs definitions as well as the interfaces among them, such as SystemC-AMS [VGE03] or Ptolemy II [BLT10]. The first uses a dataflow approach, while the latter follows an actor-oriented approach. Dataflow is more intuitive when modelling electronic and communication systems, as their subsystems are typically described in terms of inputs and outputs, while actor-oriented paradigm handles concurrency much more easily.

When modelling Cyber-Physical Systems, the Models of Computation (MoC) concept, has to be therefore generalised to *Computational Models*. CPS models do not refer just to computation anymore. Network propagation or even other kind of models for physical processes have to be included as well. There is, however, some ambiguity in the use of the term *MoC*, since the fundamental problem is very similar, and specially since some MoCs, can be directly reused as computational models for further purposes beyond computation.

Hence, there are already some projects using SystemC-AMS [MPGD13] and Ptolemy II [DLV12] for Cyber-Physical Systems, exploiting the concept of MoCs, even when the models do not refer to computation anymore. For instance, MoCs to describe analogue circuits behaviour, can be directly applied to other physical properties that have nothing to do with computation, such as room temperature variation [KR07].

In the following sections, several simulation approaches, based on different multi-domain simulation platforms will be introduced.

3.2.3.2 Multi-Domain Simulation Platforms

Unlike simulators discussed in previous sections, whose primary application was modelling either networks, hardware or software, multi-domain simulation platforms aim to provide a framework to model several domains altogether. For the sake of this analysis, the discussion will be focused in the capabilities to model network, hardware, software and physical processes.

There are different approaches to multi-domain modelling, the first approach would be to couple simulators for different domains. However, simulation coupling is a very complex task that requires a very deep analysis in order to create efficient interfaces. Additional approaches are modelling languages, with no associated tools, such as Modelica, or complete toolsuites, such as Simulink.

Modelica

Modelica is an open standard modelling language used and designed to create physical models [MEO98]. Those models are usually part of larger heterogeneous and complex systems. As a result, Modelica makes special emphasis in model exchange.

Being an open standard, there are many tools available that implement Modelica models translation and simulation. In most cases, Modelica models are translated into C-code. The different solvers for the different models of computation are then implemented as C libraries.

The Modelica Association also maintains a Modelica Standard Library, which includes generic component models and functions for different domains.

Moreover, there are already library extensions to model embedded systems [EOH+09]. However, at the time of writing this thesis, embedded systems models in Modelica are used to include them in overall system models, rather than to actually design and optimize the embedded hardware/software platform.

Simulink

A completely different commercial approach for multi-domain dynamic systems modelling is Simulink. Simulink has proprietary license and is linked to the simulation environment developed by MathWorks. Therefore, Simulink offers integration with the MATLAB environment.

There are wireless sensor networks simulators written in MATLAB, such as **Prowler** [SVML03], a probabilistic WSN simulator, which, apart from its own features, permits making use of all MATLAB functions and visualization possibilities.

Prowler has also been migrated to Java. The name of this Java version is JProwler [SIS]. JProwler is more flexible, as it can be easily extended using java objects. However, the only implementation supplied is a MICA2 mote implementation with almost no modularity, and therefore not suitable for reuse. In addition, JProwler is rather a network simulator as those discussed in section 3.2.1, as multi-domain aspects are those obtained through Simulink and its integration with MATLAB.

Prowler includes a radio-propagation model capable of evaluating reception parameters and collisions. The advantage, in comparison with radio models included in other simulators, is that it not only models the deterministic attenuation along distance, but also includes a time variant factor. As a result of a collision, a message can be marked as corrupted or not corrupted. Bit-level data corruption is not provided.

Prowler provides a very complete channel estimation model. MATLAB responds flawless to this task, as it involves mostly matrix calculations. However, Prowler does not support hardware modelling. In order to model hardware devices, Prowler must be combined with Simulink. Moreover, Simulink offers a much wider scope of possibilities, such as physical models, to even create multi-domain cyber-physical systems models. There are many toolboxes and add- ons available for modelling all kind of dynamic systems, including physical models, HDL code generation, system verification, etc.

Nygren et al. [NC11], have presented a WSN simulation that uses Prowler for modelling communication, a Simulink model of a wastewater treatment plant and even includes a MATLAB energy consumption model of MICA nodes.

Simulators Coupling

The most simple approach to simulating different domains would be to simply combine two existing domain-specific simulators. However, the underlying complexity of simulator coupling is huge:

- Interfacing between different simulation paradigms is a complex mathematical problem that needs to be deeply studied in order to obtain efficient results.

- Different simulators may be implemented in a very different way, combining different programming languages and with their own user interfaces for parameterization, visualization, etc.

Ptolemy II [Lee09] is an open-source framework for actor- oriented design which enables the construction of domain-specific tools. Hence, it focuses on the interfacing problem and avoids dealing with the implementation problem. It supports several models of computation: process networks (PN), discrete-events (DE), data-flow (SDF), synchronous/reactive(SR), rendezvous-based models, 3-D visualization, and continuous-time models [Ber]. It offers interfaces to combine several Models of Computation (MoCs) in order to create a multi-domain simulation.

The Ptolemy II framework has been exploited to model sensor networks through VisualSense, which is a modelling framework, built on top of Ptolemy II, for the simulation of component-based WSNs models [BKL+04].

VisualSense exploits and extends the discrete-event domain of Ptolemy II. It provides a wireless sound detection model as an application scenario. VisualSense includes models for packets, packet losses, battery power, power loss, collisions and transmitting antenna gain (directional antennas).

The propagation model included in VisualSense is accurate, based on a general model which can be applied to other physical phenomena, so that it is also valid for sensor physics, e.g. the physical phenomena captured by sensors, such as temperature, pressure, etc.

There is also the possibility of executing TinyOS programs in Ptolemy, through Viptos [CLZ06], an integrated graphical development and simulation environment. Viptos is able to transform a diagram into a nesC program [GLB+03] (see Section 3.2.2.1) which can be executed in a TinyOS platform. It can create Ptolemy II models from nesC files as well.

VisualSense provides a graphical user interface (GUI) for both building the simulation and showing the results. Modules can be dragged and dropped and connected to build the desired scenario. If modules included in VisualSense are not sufficient, new modules can be created by connecting some of them, or by directly programming them in Java.

The graphical orientation and the flexibility concerning models of computation, make of VisualSense a very useful and easy to use simulator for small and generic high level simulations. However, networked embedded systems typically involve more complex simulation scenarios which require high simulation performance and automatic scenario generation.

Although Ptolemy II has made an outstanding work in developing interfaces between models of computation, in practice, many times there is the necessity to use simulation tools that have not been developed under Ptolemy II, but are standard or widely accepted tools in their respective domains. In this case, the designer will have to deal with both the interfacing and the implementation problems. The **Functional Mock-up Interface (FMI) Standard** is a great step towards the combination of models and simulators. Although independent, it is closely related to Modelica. It provides a twofold interface for model exchange and for co-simulation [Mod13]:

1. **FMI for Model Exchange:** Models can be executed from an external simulation tool which supports FMI imports. This main simulation can then parameterize the input variables and gather the results from the corresponding output variables. Thus, it is not actual co-simulation and therefore it does not require coupling nor synchronization [BOA+11].

2. **FMI for Co-Simulation:** The co-simulation interface does provide the synchronization methods in order to execute both simulations in parallel.

In both cases, models are exported into so called Functional Mock-up Units (FMUs), which include a XML description file and the dynamic libraries required to execute the model.

Using the FMI Standard, it is possible to import and export models from and to other simulators, respectively. HybridSim, for instance, a SysML-based framework, is capable of using FMI to co-simulate Modelica and TinyOS models [WB13].

SystemC-AMS

Although SystemC is focused on electronic system-level design and is not oriented to multi-domain simulation, the event-driven simulation approach had to be extended in order to simulate analog subsystems. These extensions are included in an additional library called **SystemC-AMS** (analog mixed- signal) [Accb].

Modelling analog systems requires continuous or equation based simulation. For this purpose, SystemC-AMS adds three models of computation [Ini13]:

1. **Timed Data Flow (TDF):** It is a discrete-time model of computation based on sampling continuous signals with a pre-defined time step. In 1.0 specification, exceptions to this time step, with additional time stamps, could be given. Since version 2.0, time-step can also be dynamically modified in what is called **Dynamic-Timed Data Flow (DTDF)**. Dynamic time-step modification is crucial to obtain satisfactory accuracy and performance, as it enables the user to manage when to prioritize accuracy or performance depending on the specific needs.

2. **Linera Signal Flow (LSF):** It is a non-conservative continous- time model of computation that models systems using differential and algebraic equations and represents quantities as functions of time.

3. **Electric Linear Networks (ELN):** It is a conservative continuous-time model of computation to define linear networks. They are modelled as differential and albegraic equations based on electrical primitives.

Although the AMS extensions are intended for modelling the analog parts of the embedded systems electronics, they can also be used to model other continuous physical properties, which respond to similar kind of equations. Although current extensions are very limited to model non-linear behaviour, they are sufficient to model a wide variety of physical processes that can then be included in an electronic-system level model based on an industry standard language already accepted for hardware and software co-design.

3.3 Energy Simulation and Profiling

Energy simulation in microprocessor based systems is not a novel issue. Apart from hardware energy consumption models, there were also analysis of energy consumption reduction by software

optimization [TMW94] and even energy simulation of a Real Time Operating System (RTOS) [DLRJ00].

To appropriately dimension batteries and energy budgets, system designers must consider the power consumption of the devices at a very early stage. However, most efforts in power and energy efficient systems are made during hardware design.

Hardware designers do know in deep the specification of power consumption of the components they select. They have also to dimension the power supply, the batteries, etc. and therefore they are very aware of the power consumption of the system. However, this awareness is lost for designers or users who do not know the hardware in deep.

The lack of awareness becomes a major optimization issue, because energy optimization does not end with hardware design. The impact of the hardware design in the overall energy consumption is limited. The hardware platform offers just the resources for the system to achieve its functionality, but is the usage of this resources the one that determines the final energy consumption of the system.

There are several approaches to implement power saving features:

- **Technological:** Until recent years, the most advancements in power consumption were technology driven. Miniaturization and nanotechnology has not only reduced the size and improve the integration, but also has significantly reduced the power consumption. However, the technology is getting close to the theoretical limits, and therefore, the improvements are getting smaller and slower.
- **Architectural:** Due to the high demand pressure in battery powered devices, and the technology limitations, architectural optimization is being thoroughly exploited in the later years. As a result, we can see now multi-core architectures, with dedicated efficient processors, clock gating [WPW00], power gating architectures with separated power domains, and aggressive power management policies to disconnect the power domains not in use.
- **Runtime:** Most operating systems include nowadays some power management infrastructure, in order to be able to react to the specific runtime circumstances. For that purpose, there are APIs that can be used, under some circumstances, so that some parts of the system can be disconnected or switched into a sleep mode. The applications or the user can also make use of this API.

Nevertheless, having all this infrastructure does not assure an efficient energy behaviour. Energy efficiency still involves a high degree of involvement and commitment from the programmer and the user side. However, making the best decisions is not always an easy task.

In practice, there are many decisions that have very much impact in the energy consumed by the system, ranging from the selection of hardware components to communication protocols or the application duty cycle. Estimations enable making the decisions having the lowest energy consumption. This is mainly done by means of simulation, which has to be performed anyway to assure the correct function of the device.

However, adding energy and power estimation to simulation might drastically affect simulation performance. This could lead to simulations that require longer execution time, by orders of magnitude, than the real system, [ELVAMS+06]. However, to assess energy lifetime, simulations should take much shorter time than real system operation.

3.3.1 Power Estimation

Power estimation in electronic systems has also been a matter of study for the last decades. With integration and System-on-Chip designs, it becomes more and more difficult to track how much power is dissipated with the granularity required to perform a thorough analysis.

Measures only provide coarse grain values which do not permit finding out responsibilities for power leaks. Furthermore, in ultra-low power systems, power metering might add unacceptable power consumption overhead.

The difficulty to estimate power even on the real platform, has led to simulated power estimation models. However, for this same reason, validity of these models is frequently in question. Even performing the model assessments is far from trivial.

The most extended approach of power models consists on the use of Finite State Machines (FSM) [FGSS98] [BHS98]. Digital systems typically have two well differentiated power consumption components: static and dynamic.

Thus, they can be described as a set of states, with different static power consumption, while dynamic power consumption occurs during state transitions.

The power consumption of a system with several subsystems would then be the summation of the power corresponding to the different states ($P_{state,i}$) plus the power consumed due to the transitions ($P_{trans,j}$), as seen in Equation 3.1

$$P = \sum_{i=0}^{N} P_{state,i} + \sum_{j=0}^{M} P_{trans,j} \tag{3.1}$$

Power consumption values can be obtained from specification or mathematical models. However, current system complexity makes evaluation of transistors states and switching probabilities infeasible. High level models are required, as the one proposed by Benini et al. [BHS98], which is also based in a Finite State Machine but considering power management states, instead of digital states.

The problem of the system-level power state machine is that it assumes an average power consumption value on each state. This is appropriate for most systems, but some systems, such as analog, would have undefined number of states.

Although **ns-3** is a very generic network simulator, there is a specific energy frameworks based on it [WNP11]. In this framework, energy consumer devices are modelled as finite state machines, with each state associated with its corresponding current draw value. However, the framework is also able to simulate devices with an undefined number of states (e.g. analog), as long as the current draw value can be estimated as a function of any other magnitude, e.g. an electric motor whose power consumption is a function of the rotation speed.

These framework also provides models for energy sources. It includes not only a basic linear battery model, but also a Rakhmatov-Vrudhula (R-V) model, which also considers the non-linear behaviour.

Another problem of energy estimation using the system-level power state machine is that time spent on each state is not always possible to account in simulation. For instance, in **PAWiS**, although power simulation is not supported directly in the framework, the framework includes an

interface to model CPUs and a power meter class where power related information can be reported to be logged. In the PAWiS module library, there are implementations of transceivers and CPUs. However, granularity of CPU models is not fine enough to obtain accurate timing. PAWiS does not have a cycle-accurate CPU model that can account for CPU active time intervals, which have to be estimated by the user.

This is the case also of **PowerTOSSIM**. PowerTOSSIM is an extension to TOSSIM to model energy consumption. It estimates power consumption per-node and per- component based on state transitions logged at runtime. This means that, as long as the state machine scheme remains the same, several power models, with different power consumption values for each state, can be applied after execution. However, accuracy of PowerTOSSIM is restricted by TOSSIM limitations, which are mainly the network and propagation behaviour and, as in PAWiS, the inaccuracy capturing CPU timing. However, unlike PAWiS, PowerTOSSIM uses a high-level estimation of execution time, instead of delegating on user estimations.

PowerTOSSIM separates tasks in blocks and uses a code-transformation technique to estimate the number of CPU cycles they require. Nevertheless, deviation still exists in comparison with cycle-accurate models when many asynchronous events (like interrupts) take place.

There are however, several versions of PowerTOSSIM. It was developed on top of TOSSIM for TinyOS 1.x. When TinyOS 2.x. was released, a newer version, PowerTOSSIM2, was created. Both were created for MICA2 motes. PowerTOSSIMz extends the latter one for MICAz nodes [PCC+08]. Apart from the different target architecture, PowerTOSSIMz includes a battery post-processor model which simulates the non-linear discharging of the battery. In spite of being a post-processor, it can also work at runtime. It implements a stochastic approach to battery model.

IDEA1 simulator, also includes power estimation infrastructure based on finite state machines [DMN10b]. In a first approach, IDEA1 included some estimation of the timing of the micro-controller unit based on software assembly code. Nonetheless, IDEA1 is not a real-code emulator, which means that these estimations only come from code analysis, but there is no instruction level simulation nor cycle-accurate simulation. In more recent work, instruction set simulators have been developed for AVR and MSP430 architectures [GNMO12]. Instruction-Set Simulators provide almost cycle-accuracy, with some slight deviations due to architecture performance optimizations such as, caches, branch prediction and out-of-order execution, which are so far not being used in low-power embedded microcontrollers as they do not improve the performance-per-watt. Therefore they can be considered as sufficiently accurate to estimate MCUs execution times.

AVRORA and COOJA with MSPsim are already cycle-accurate models for AVR and MSP430 respectively. They both have power estimation extensions which are therefore very accurate. The power extension for AVRORA is called **AEON** and is included from version 1.4 [LW04]. AEON also includes a free space radio propagation model for AVRORA, with distance attenuation but without modelling noise. On the other hand, the power profiling tool used in COOJA consists of an integration of the power profiler used in the Contiki operating system [DOTH07] into the **COOJA/MSPsim** simulation.

Energy estimation in Contiki consists on recording time stamps whenever a component is activated and making the corresponding calculations based on the time difference when the component is deactivated. There is also a visualization tool for COOJA with special focus on power consumption [OED10].

There are therefore many power estimation extensions to state-of-the-art embedded system simulators. However, even the system-level power state machine is too detailed for software and network optimization. Transparency of the different abstraction layers prevents the software developer from knowing which states are activated during a software task. Moreover, if this task is distributed in the network and impacts a group of nodes, the energy awareness is totally lost.

As a result, in Networked Embedded Systems, energy awareness is not solved with power estimations through sytems-level power state machines. It becomes necessary to track state changes and build higher level data structures that can answer how much energy impact a high-level task has in the overall distributed system. These data structures are called energy profiles.

3.3.2 Energy Profiling

The essential starting point on the way to energy optimization is gathering knowledge about energy consumption of the system. In traditional embedded systems, this can be done by tracking energy consumption values of different subsystems. Power consumption information are typically based on the hardware subsystems, as power models are defined based on hardware models and specifications.

Power consumption information has to be gathered at a very low level in order to be accurate. However, at higher levels, the tasks performed by the system are complex and typically involve several hardware subsystems. At the same time, abstraction implies transparency and therefore, the high level designer and the user are not always aware of the role of each hardware subsystem in any system function and even if they are, they do not know the implications in energy consumption.

In Networked Embedded Systems, there are three kind of energy profiles that are necessary for system optimization:

1. **Hardware:** Complex SoC architectures may require the aggregation of several simple subsystems into systems-of-systems that are more manageable and understandable at higher levels. Also many of their states might be correlated and aggregation can lead to simplification and performance improvement.

2. **Software:** Software tasks may make use of several hardware subsystems. The software developer will very likely want to evaluate the energy consumed in performing some software tasks and compare the energy consumption difference between alternative algorithms or implementations. Software profiles must therefore aggregate the energy consumed in the different hardware systems in order to perform the specified software task.

3. **Communication:** Networked Embedded Systems are distributed systems. Therefore, some tasks will involve energy consumption in several nodes. Networks are typically ad hoc, and sometimes require multi-hop communication. The energy consumed all over the network due to a single communication transaction is also a very useful metric for the designers in order to optimize overall energy consumption and prevent some critical failures due to battery depletion at very transitted paths. Thus, communication profiles must account the energy consumed all over the network in order to perform some distributed task that requires communication.

Hardware profiles are present in most power estimation extensions and simulation tools. Most of them are at least capable of estimating energy consumption in a local node by aggregating energy consumption of the different subsystems.

However, only some power estimation extensions include software profiling tools. Power profiles enable identifying the contribution of software parts to power consumption. Hence, the designer can focus on optimization of those tasks whose impact in power consumption is more significant.

Figure 3.5 shows how different energy consuming processes are observed at different levels of abstraction. Abstraction permits that the application programmer could send a message without knowing the involved phases described at system level software, and both the system and application programmer can accomplish their jobs without knowledge about the current consumed at the hardware level. Although this abstraction is needed to increase productivity and achieve complex tasks, the gap in the consumed energy knowledge has to be filled in order to enable design of energy optimized systems.

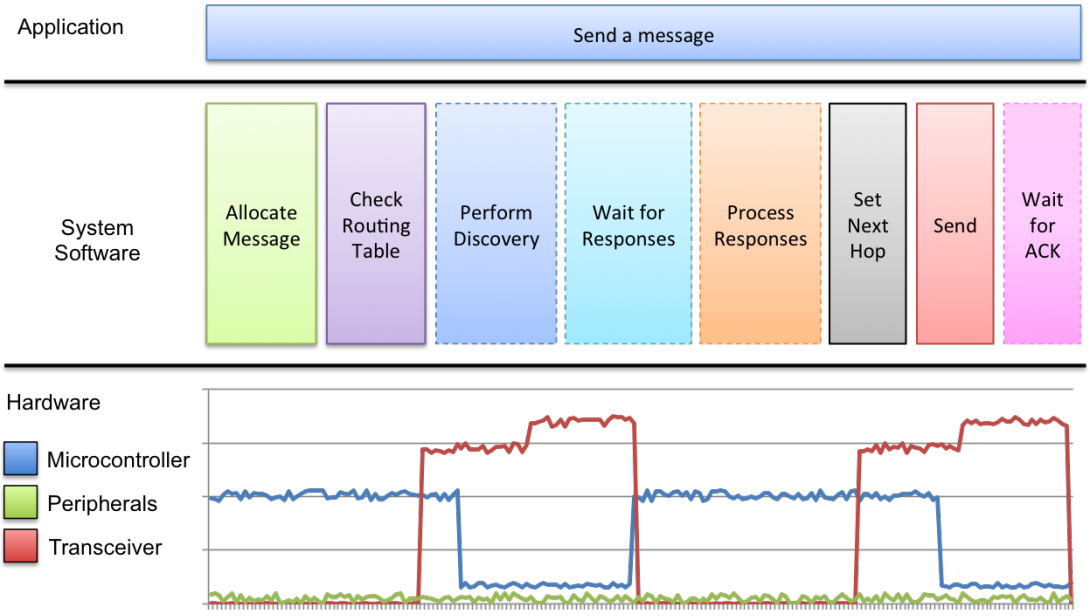

Figure 3.5: Energy consuming semantics at different levels of abstraction

Software profiles are already an old concept, that emerged due to program complexity. Complex programs involve many routines, processes and threads and therefore, information about resources utilization becomes difficult to obtain [GKM82]. **Rialto** software profiler [JMF+96] classified different threads into so called "activities". By grouping threads into coherent semantic categories, they could be fairly and efficiently scheduled in the CPU.

In networked embedded systems, the same profile concept is needed. However, there are mainly two significant attempts to evaluate software energy consumption in networked embedded systems: in AEON [LWTP05] and in Quanto [FDLS08]. **AEON**, as already explained in Section 3.3.1, is a power consumption estimator based on AVRORA. AEON has been used to partition TinyOS applications in so called *routines* and account their energy consumption. **Quanto** goes further and permits the user to define his own *activities*.

However, local optimizations are not sufficient to optimize energy consumption in Networked embedded systems. It is also crucial to be able to optimize the network as a whole [Agr11].

However, there are no examples of network- wide energy profiles in simulation. Quanto tracks network-wide power consumption, but it is an extension to TinyOS to be executed during runtime. It could probably also be simulated using TOSSIM, but TOSSIM accuracy is very restricted as it cannot capture microcontroller low-level events.

3.4 Discussion

As described in previous sections, there are several difficulties in energy aware networked embedded systems design. Simulation helps in overcoming those difficulties.

There are many simulation options to model this kind of systems. As these systems are the convergence of several technologies, there are many already existing simulation tools and environments. However, most of these solutions are not complete. Network simulators provide protocols and propagation models but lack low level details which are crucial to accurately estimate power consumption. Virtual Prototyping tools provide real code emulation but not all of them include a low level microcontroller model that could accurately capture the timing, required for energy consumption calculation. Those that include hardware accurate emulation, have also severe performance restrictions and are not capable of modelling propagation or networks with many nodes.

Furthermore, energy consumption is a cross-layer system aspect, as it is conditioned by hardware, software, network and even the environment interaction. However, the combination of different tools to model each of these aspects is very inefficient and requires a lot of effort. Therefore, the complexity of the system is also a challenge from the modelling point of view.

As co-simulation is not always affordable, the trend has been to extend previous solutions to include some of the new required aspects. The results are very different depending on the starting point, and therefore, this chapter provided an overview and analysis of the current state-of-the-art simulators, classified by their initial simulation approach. There are therefore both bottom-up and top-down evolutions from cycle-accurate and high level simulation models, respectively. However, from the simulators analysis, it can be concluded that there is no fully satisfactory holistic simulation approach. The different domains comprehended require the combination of different simulation approaches, and even though there are some environments that allow the combination of different Models of Computation (MoCs), such as Ptolemy II or Modelica with the Functional Mock-up Interface, they are far away from any Electronic System Level industry standard.

On the other hand, SystemC is a system level industry standard for hardware and software co-simulation. TLM extensions provides an abstraction in communication that have had very good acceptance in industry. There are many TLM hardware models available, including Instruction-Set Simulators (ISS), which are very helpful in order to obtain accurate estimations of the energy consumed by microcontrollers.

When exploring TLM capabilities, the communication abstraction approach turned to be also very appropriate to model the wireless channel. Furthermore, the Analogue-Mixed Signal extensions to SystemC permit not only to simulate analogue electronic subsystems, but also opens the door to including some continuous physical processes models, which can be described with the same Models of Computation as the analogue models.

This way, a consistent and comprehensive framework could be created within the SystemC ecosystem, with emphasis on the hardware and software co-simulation, but considering also the network

and the physical processes that affect the final performance of the system and therefore have great influence in energy consumption.

Having the appropriate simulation infrastructure solves the problem of including all the factors that intervene in the energy consumption problem, however, this is not sufficient to enable high level energy optimization. The low level power consumption values have to be aggregated into energy profiles that have some meaning for the high level designer. These energy profiles can then expose in a high semantic level which tasks, conditions or operations reduce the energy efficiency of the system and can then enable making energy efficient decisions when selecting and tuning high level aspects such as the communication stack and its parameters, the resources distribution alternatives, the network topology, the threshold values for reporting sensor measurements, etc.

The state-of-the-art analysis revealed that this energy profiling for high level optimization is a barely explored feature. There are some exceptions, in which software profiles from a local perspective can be found, but there are no energy estimations to simulate the overall energy efficiency of a distributed system and to explore which design alternatives can extend the whole distributed application lifetime, and not only the local hardware/software node lifetime.

According to all this state-of-the-art research, this thesis will provide an integrated solution to perform the cross-domain and multi-level simulation required for efficient and complete energy simulation, considering not only hardware and software, but also network and physical processes that are involved in system operation.

Furthermore, this thesis will contribute the infrastructure to create high level energy profiles that provide the semantic framework that will enable high level energy optimization of a wireless distributed and interactive system.

4 Energy Simulation and Profiling

Chapter 2 analysed and characterized the energy awareness problem, describing all the elements involved and clarifying all necessary concepts and technologies that lead to the approach presented in this Chapter. In Chapter 3 the State-of-the-Art of distributed embedded systems simulation has been presented and discussed, making special emphasis in energy aware capabilities.

This Chapter describes a new approach to improve power and energy consumption awareness in networked embedded systems at higher levels of abstraction, where awareness is almost vanished due to abstraction transparency. To fill this gap, energy gathered from state-of-the-art power models will be aggregated into energy profiles that have a direct meaning for high level optimization.

The path selected to fulfill this goal is framed within Model Based Design (MBD) methodologies, i.e. simulated data will be gathered, processed and refined in order to deliver it, in the best possible form to the different designers at different abstraction levels, such as the firmware designer, the network designer or even the application designer.

4.1 Energy Aware Methodology

As discussed in Section 2.5, energy optimization of networked embedded systems requires a cross-level analysis and synergies among all design levels. Starting from the selection of hardware components and finishing with the network architecture, all levels must be considered in order to make the best decisions possible.

However, there are some difficulties to consider all these levels at design time:

- Manufacture of the embedded platform is costly and time-to-market is very short. Software, network and physical environment must be considered already during hardware exploration phase and all hardware optimizations must be made before manufacture. Software and network optimization must start before a real prototype is available. F
- Deployment of networked embedded systems is typically under unknown conditions and the systems must be autonomous, as post-deployment maintenance might not be feasible. Simulations permit testing the distributed systems deployed under a whole set of environmental conditions and with a whole set of possible network topologies, in order to identify potential energy leaks.

- Obtaining real energy consumption data from an operative platform is unfeasible. The embedded systems are ultra-low power devices that cannot afford monitoring and profiling their energy consumption. Furthermore, system integration restricts the possibilities for accounting energy consumption to different subsystems.

For all these reasons, the methodology to achieve energy awareness proposed in this thesis is integrated within Model-Based Design methodologies and virtual prototyping. The addition of energy awareness to the existing methodologies can be separated in three phases:

- **High level power models:** They must provide accurate enough energy consumption data while keeping an acceptable simulation performance. The power model proposed here is based on Finite-State Machines (FSMs). This approach exists already in the state-of-the-art [FGSS98] [BHS98] (see Section 3.3.1). However, there is no standard or formalized approach. Thus, a formalization of the used power models is required.
- **Simulation requirements:** In order to estimate energy consumption from power states, the time spent at each state must be estimated. This estimation is done through simulation. To accurately estimate system activity, a virtual prototype is not sufficient. It must be extended with models for network communication and cyber-physical interaction. It is therefore required to specify and develop a comprehensive simulation framework that enables modeling all the required aspects.
- **Energy Profiling:** Energy optimization is not complete without classifying, characterizing and aggregating the power consumption data provided by the power models in energy consumption profiles that contain workable or practicable information for the high level designer.

All parts are fundamental and co-dependent. Using the best power models is meaningless if the information they supply is unusable. On the other hand, information provided by profiles can be completely misleading if the data supplied by the power models is far away from reality. In networked systems with cyber-physical interaction, both wireless communication and physical processes determine the behavior of the system and therefore, the demand for power consumption of the system along time.

In all cases, simulation performance plays a crucial role. Accurate data and good profiling is only useful if simulations can be performed very fast. In the distributed embedded systems world, this means being capable of simulating even several years of system operation at design time. Including models for wireless communication and physical processes as part of the simulated system is an additional challenge for simulation performance.

4.2 Formalization of Power State Machines

The basis underneath the whole energy simulation problem is the use of the most appropriate power models. Those models are responsible for the accuracy of the simulation. Furthermore, in many cases, they will also have very significant impact in overall simulation performance.

Although the best models to be chosen depend on the specific application scenario, in this section, a flexible and abstract power model concept, based on Finite State Machines (FSMs), with some

variations and extensions, is proposed and discussed. The concept has already been used in different power modelling approaches. The application of FSMs to model power and energy consumption is very intuitive. However, there is a lack of formal documentation in literature. As there is no standardized formal definition to use as a reference, a custom formal definition for the sake of this work will be made in this section. Power State Machines, as defined here, could be reused for other use cases as well.

The idea of Power Finite State Machines (Power-FSMs) is very simple, and is based on these main assumptions:

- Functional behaviour of electronic subsystems can be reduced to a restricted number of operation modes or states.
- Power consumption within these states remains stable, within boundaries and therefore can be averaged into a constant value.
- Big variations in power consumption only occur when switching from one state into another.

The combination of the different operation modes for all the subsystems, will conform the characterization of so called Power States, which together with the transitions constitute the Power Finite State Machine (Power-FSM).

Although these assumptions may give the impression of an oversimplifying view, they suit a wide spectrum of practical cases, specially in energy efficient electronic systems. The reason for this is that specific operation modes are already defined in most complex electronic systems for different purposes, such as power management or just ease of use.

For instance, nowadays, a typical approach for energy saving and ultra-low power design consists in distributing different subsystems in different power domains. Power gating is used in these domains in order to switch them on or off independently. Once the architecture is set in this way, a good power management strategy enables switching on these domains only for the minimum time required to carry out their tasks, avoiding energy consumption on static power dissipation or on keeping different signals, like clocks, while they are not actually required. An example of this power management policies is "Run Fast Then Stop" and is very commonly applied because it is the most effective way to reduce static power consumption, while dynamic power consumption remains equivalent.

A consequence of power management policies, such as "Run Fast Then Stop", is that they transform the behaviour of these subdomains into an almost dichotomic operation: either activated and operating at full load or completely disconnected. This reduces the number of possible configurations, simplifying the list of possible power states. Thus, average power consumption estimation has to be performed in just this collection of very specific operation modes.

4.2.1 Simple Power State Machines

The most simple power state machine just considers a number of power states and ideal transitions among them. This simple power state machine can be formally defined as simplified Mealy machines:

Definition 1. A **Power State Machine** is a tuple (S, S_0, Σ, T, P) consisting of:

- a finite set of states (S)
- an initial state ($S_0 \in S$)
- a finite input alphabet Σ
- a transition function ($T : S \times \Sigma \longrightarrow S$) that regulates next states
- a power consumption function $P : S \longrightarrow \mathbb{R}_{\geq 0}$ which assigns an average power consumption value to every state.

Figure 4.1 shows a diagram representing the Power-FSM of a transceiver subsystem. Table 4.1 contains the power consumption values for all the states in the power state machine.

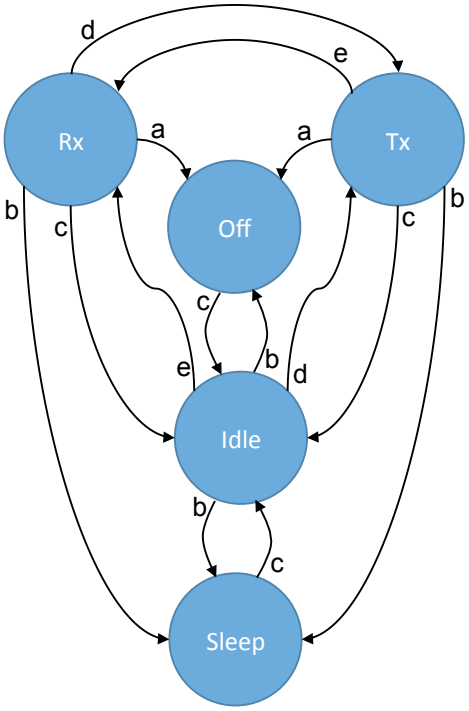

Figure 4.1: Example of Power State Machine for a transceiver

Table 4.1: Power consumption function

State s	**Power Consumption** $P(s)$
Off	$P(Off) = 0W$
$Sleep$	$P(Sleep) = 0.02\mu W$
$Idle$	$P(Idle) = 16.8mW$
Rx	$P(Rx) = 36.9mW$
Tx	$P(Tx) = 42mW$

In order to estimate the energy consumption of a subsystem represented by a simple power state machine, the different states, as well as the time spent on each state have to be computed. The Equation 4.1 shows how to calculate the energy of a subsystem defined by a power state machine of N states. $P(x)$ is the power consumption function. t_i is the time spent in this state.

$$E = \sum_{i=0}^{N} P(S_i) \cdot t_i \tag{4.1}$$

4.2.2 Power State-Transition Machine

State transitions are sometimes negligible in terms of energy, either because power consumption does not vary in relation to power consumption during the adjacent states or because their duration is very short. However, in some cases they require some time, such as oscillators or PLLs, which always need some time until they stabilize their output. Waking-up from a sleep mode also typically requires some time, which might be counter-productive, if the system went to sleep for a too short time period. This is specially important when applying aggressive power management policies, which send the different subsystems to sleep as soon as possible. An accurate estimation of the cost of going to sleep and waking up the device is crucial in order to make the best decisions possible.

Therefore, to increase the accuracy the simple power state machine can be enhanced to consider also energy consumption of state transitions, increasing the estimation accuracy. Furthermore, adding transition delays also provides better accuracy from the functional point of view, as all delays produced when switching states will be computed in the simulation.

The first and most simple approach is to consider transitions as new states, very similar to power states. However, there are several differences in relation to power states:

- Time spent during transitions is known and can be considered to be always the same.
- Power consumption typically responds to a transient profile, and therefore not constant.
- Although power consumption is not constant, energy consumed during a state transition can be estimated to be the same every time this transition occurs.

This extended power state-transition machine is formally defined as an extension to the power state machine, which includes the energy of the transitions:

Definition 2. A **Power State-Transition Machine** P is a tuple $(S, S_0, \Sigma, T, P, E, D)$ consisting of:

- a finite set of states (S)
- an initial state ($S_0 \in S$)
- a finite input alphabet Σ
- a transition function ($T : S \times \Sigma \longrightarrow S$) that regulates next states
- a power consumption function $P : S \longrightarrow \mathbb{R}_{\geq 0}$ which assigns an average power consumption value to every state.
- an energy consumption function ($E_T : S \times \Sigma \longrightarrow \mathbb{R}_{\geq 0}$)
- a delay function ($D : S \times \Sigma \longrightarrow \mathbb{R}_{\geq 0}$)

Table 4.2 shows the energy and delay functions for the Power State Machine diagram of a trasceiver represented in Figure 4.1. Transitions with negligible time duration and energy consumption are omitted in the table. In this example, significant transitions are those that need voltage regulator start-up, crystal oscillator start-up and PLL lock, as this three operations require some time to complete.

Table 4.2: Energy consumption and delay functions of the transitions

State s	**Input** σ	**Next State** $T(s,\sigma)$	**Energy** $E(s,\sigma)$	**Delay** $D(s,\sigma)$
Off	c	$Idle$	$24.2\mu J$	$1.44ms$
$Sleep$	c	$Idle$	$13.6\mu J$	$0.84ms$
$Idle$	e	Rx	$7.68\mu J$	$192\mu s$
Tx	e	Rx	$7.68\mu J$	$192\mu s$
$Idle$	d	Tx	$7.68\mu J$	$192\mu s$
Rx	d	Tx	$7.68\mu J$	$192\mu s$

In this case, energy estimation of a power state-transition machine with N power states and M inputs is given by Equation 4.2. In the power state-transition machine, in addition to the energy estimation component from Equation 4.1, the contribution of transitions to energy consumption is added, being $E(x,y)$ the energy consumed during transition σ_j from state S_j, and k_j the number of times this transition occurs.

$$E = \sum_{i=0}^{N} P(S_i) \cdot t_i + \sum_{j=0}^{M} k_j \cdot E_T(S_j, \sigma_j) \tag{4.2}$$

4.2.3 Limitations

The requirements for estimating energy consumption using Power Finite State Machines can be elicited after inspecting equations 4.1 and 4.2. Energy calculation requires the following data:

- Average power consumed in each power state.
- Energy consumed in every transition.
- Time spent on each state.
- Number of transitions triggered.

While power and energy consumption can be obtained from hardware data sheets and/or measurements, the number of transitions and the time spent on each state are both application driven characteristics, which depend on the firmware, the software, and, as explained in Section 2, even on the network and the environment. Therefore, these two unknown variables must be obtained from simulation.

Nevertheless, obtaining the accurate timing for the different state transitions is a distinct problem depending on the device to simulate. In some cases, timing can be obtained from a functional level simulation, while in other cases a very detailed simulation might be required.

Focusing on the main subsystems present in distributed embedded systems, there is the case, for instance, of the transceiver, which is typically governed by the application and the communication protocol used. Hence, the time required by a transceiver to send or receive a message, is determined by the bit rate of the protocol and the message length. The time the transceiver spends listening for incoming transmissions is typically defined at the application level. Thus, time simulation for a transceiver can be estimated at a very high level, without modelling the transceiver in depth.

On the other hand, modelling time for a microcontroller unit usually requires a deep knowledge of the microcontroller architecture. Software is typically written in high level generic programming languages. Estimating the time required to execute one algorithm is far from trivial. Depending on the microcontroller architecture, the same code must be translated into a different number of instructions. Furthermore, nowadays microcontroller architectures have sophisticated optimization mechanisms, such as pipelines, cache memories, branch predictors, etcetera. By using these mechanisms, the real timing might diverge significantly from time estimated at the instruction level. However, implementation details to model such mechanisms are not always disclosed.

Accordingly, there are some severe trade-offs when modelling a microcontroller. Cycle-accuracy, apart from being a huge bottleneck for simulation performance, is not always feasible, for the aforementioned reasons. Availability of Instruction Set Simulators is better, but they are specific of the processor architecture used, and therefore, they can be used for optimization once the system architecture is decided, but they are not very useful for architecture exploration. Besides, they also restrict simulation performance significantly.

Furthermore, embedded systems are part of a distributed and cyber-physical application, where the physical processes and the network events play a crucial role in system activity. Modelling both aspects is therefore crucial to trigger realistic state transitions.

In conclusion, simulation is required to enable the use of power finite state machines to estimate energy consumption, by providing the timing of every state transition. This timing, however, depending on the subsystem, might require a functional high level simulation, or a very detailed simulation. The trade-off between performance and accuracy must be optimized for each application case and different simulation approaches have to be followed in order to tip the scales in the most appropriate direction according to the specific requirements.

4.3 Simulation Requirements

Virtual prototyping and Model-Based Design methodology are very widely used and accepted for embedded systems design and play a crucial role in distributed embedded systems as well. However, while complexity increases, the demand for more and more models of more and more types or nature increases as well. As a result, the requirements for simulation environments that enable the creation and interconnection of those models become more critical and difficult to fulfil. It is therefore crucial to correctly identify the simulation framework requirements, which will determine the whole simulation framework design process.

The requirements for the energy awareness approach were already analysed in previous chapters. However, the requirements for the simulation platform that could enable this energy awareness still have to be discussed.

4.3.1 Simulation Performance

One of the essential characteristics of resulting simulations is their performance. The first step is to define what performance is to be achieved. Depending on what is to be modelled, performance requirements can be very different.

If we consider the energy consumption problem, within the distributed embedded systems context, it is usually required to be able to simulate even several years of operation in very short time, so that the feasibility, i.e. energy lifetime, of the system can be evaluated during design stage. For some devices, energy lifetime required can easily surpass ten years.

A simulation framework can achieve better performance by providing abstraction. However, as already mention, abstraction typically opposes awareness. Therefore, the abstraction performed must still expose the parameters the designer needs to monitor and analyse.

4.3.2 Multi-level Simulation

This requirement is partially a consequence of the simulation performance requirement. As already mentioned, performance is increased through abstraction. However, abstraction also overlays some simulation details that might be of interest.

As the design process covers very different aspects, the details that are crucial for some simulations might be irrelevant for others, and therefore, performance could be improved by using different abstraction levels depending on the simulation scenario.

Another crucial reason for multi-level simulation is the need to have concurrent models at all possible design stages. Some parts of the system might already be known in detail while other parts can still be in a very early design phase. The simulation environment should allow and integrate both cases. Furthermore, it is very useful to simulate the same system at the functional level and at implementation levels to validate and verify them.

As a result, capability of using different levels of abstraction depending on the circumstances is also a hard requirement for the simulation framework and the most efficient way to deal with the performance-accuracy trade-off. In addition, providing higher abstraction levels where needed also accelerates the prototyping process.

4.3.3 Multi-domain Simulation

This is already a well-known requirement in today's embedded systems simulation, and actually one of the most important advantages of the Model-Based Design methodology. Embedded systems require hardware and software to be designed almost at the same time. However, software depends on the hardware platform that will execute it. Moreover, knowledge about the software to be executed permits better optimization of the hardware designs. Co-simulation of both hardware and software enables simultaneous development and immediately provides hardware awareness to software developers and vice versa.

In distributed embedded systems design, the network becomes a new additional aspect that joins this co-design problem. The network topology affects the hardware and software requirements, and both hardware and software determine network topology possibilities.

Furthermore, although the environment is not part of the design, it affects significantly the system operation and therefore, awareness of the relevant environmental conditions has to be provided to the designers so that they can produce successful designs. Model-Based Design methodologies include Hardware-in-the-Loop (HiL) testing so that the hardware can be tested under simulated conditions that resemble those at the deployment location.

However, simulating different domains usually requires following different simulation approaches that typically lead to using different simulators. The disadvantage of mixing different simulators is that the simulation cores have to be synchronized in order to provide a consistent output. This synchronization is a source of inefficiencies a performance loss.

Again with simulation performance in mind, it would be preferable to develop a simulation framework that uses the same simulation environment to model all the required domains: software, hardware, network and physical environment.

4.3.4 Flexibility

Setting up an embedded system simulation typically requires instantiating and connecting some hierarchically structured components. On contrast, setting up a distributed embedded system simulation might require instantiating a huge amount of nodes. Furthermore, these nodes might have different roles and, on consequence, different hardware/software configuration.

Hence, it is necessary to provide the means to the user to be able to instantiate a whole network of embedded systems with different configurations. Otherwise, the simulation framework would only be useful for setting up very simple examples.

Some embedded systems simulators provide component-based graphical user interfaces where the designer can manually drag and drop and interconnect the different elements of the system. However, this mechanism also results infeasible for networks with hundreds or even thousands of nodes, which might be the case in some Wireless Sensor Networks.

Additionally, variability and heterogeneity of the distributed embedded systems architectures and applications, makes covering all possible situations an infeasible task. Therefore, the simulation framework requires extensibility and some sophistication which most likely requires using some high-level programming language rather than a more restrictive and specific modelling language.

4.4 High Level Energy Awareness: Profiles

High-level is a relative concept. Compared to gate level power estimation, a hardware component power estimation can be described as high level estimation. In consequence, before approaching the high level energy awareness problem, it is necessary to define what high-level means in the context of this thesis.

When a hardware designer makes decisions with energy awareness in mind, he can look up different hardware components specifications, and select those with most suitable electrical characteristics. When he has to determine the best possible hardware architecture, he still can compare the electrical characteristics of the components included and make an estimation of their utilization, however, the analysis already becomes more complex. When the software designer tries to optimize his code for energy efficiency, he might follow some guidelines, but to really be aware of the

effectiveness of those optimizations he must also have knowledge about the specific hardware architecture and components. If the software is being developed when the hardware architecture is still not fixed, the problem becomes unapproachable.

If the designer of the distributed system wants to optimize the system to extend its lifetime he will need knowledge about the energy required for executing some tasks in different hardware/software platforms, the energy demand pressure of each of those systems versus the available energy supply, and the energy required for communication between those platforms, which not only means the energy consumed in effective communication, but also the energy wasted in fruitless communication and communication overhead. Only with that information he can wisely distribute the computational load in order to extend the energy lifetime of the system.

So, high level energy awareness in the context of this thesis means that energy optimal decisions can be made for the design of hardware architectures, firmware, application software, networks and distributed systems. A diagram showing all the levels that are consider in this approach is shown in Figure 4.2.

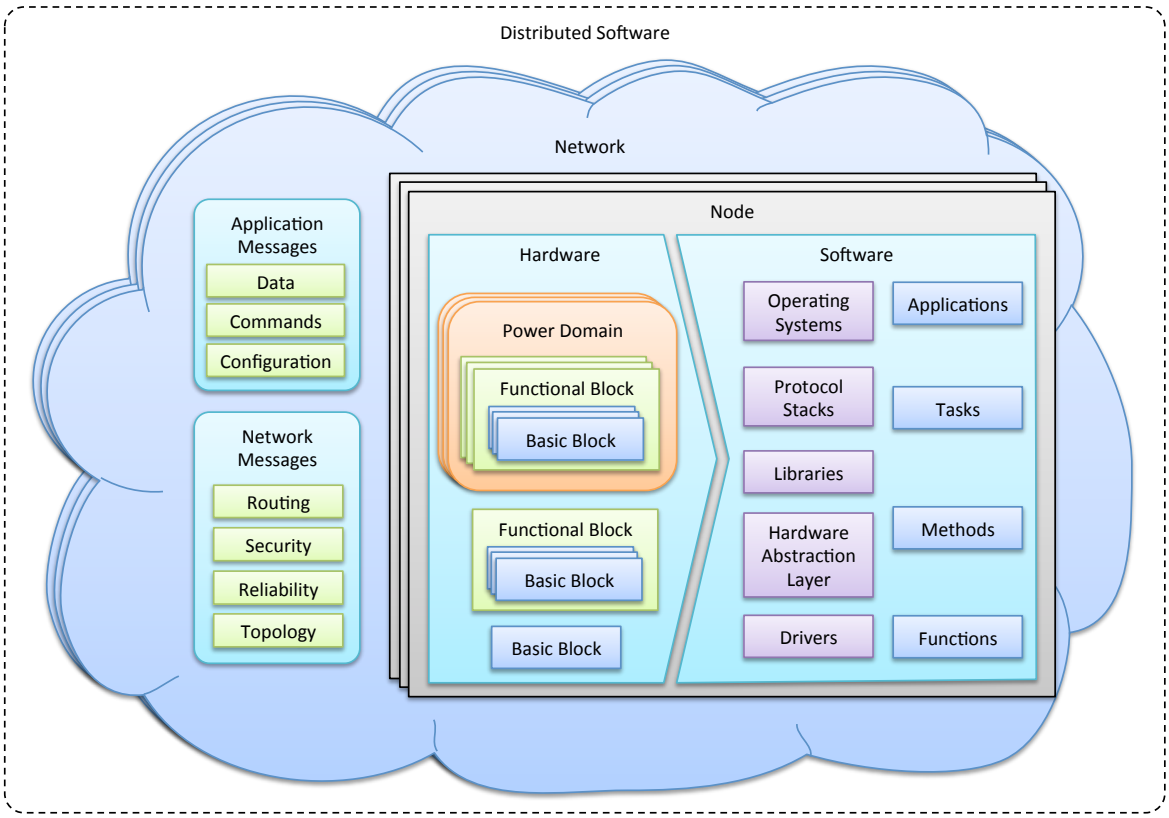

Figure 4.2: Power models are associated to either hardware basic or functional blocks. However, there are many other higher level structures that require estimating the accountable energy consumption for optimization.

Energy consumption is basically the power consumed along time. Power consumption ultimately occurs in hardware and can be calculated from its electrical characteristics. To obtain the energy, power consumption must be accounted along time. However, to optimize the designs it is not sufficient to just calculate the energy consumed.

Energy optimization requires to complement the energy consumption numbers with information about the logical elements that were involved in that energy consumption. Those logical elements

vary depending on the design level. In this thesis, four main groups of logical elements are established:

1. **Hardware:** Hardware elements are typically hierarchically structured. As explained in Section 2.1, the typical hardware device consists of several hardware subsystems that, in turn, consist of simpler hardware components. Power models will typically be available only for very simple components. In order to obtain the power model for a whole subsystem or a whole device, the power models available for the components integrating it, have to be aggregated into the overall model.

2. **Software:** Software can be partitioned into meaningful units which perform some basic tasks or activities. These tasks typically involve several hardware subsystems. Power information from the power models of those hardware subsystems have to be aggregated in order to create a software activity power model.

3. **Communication:** The communication process between a sender node and a destination node involves a number of software tasks executed in both nodes. If the network requires multi-hop routing, there will be even more tasks executed in other nodes that are actively or passively involved in the communication. In order to estimate the energy consumption of the communication process, the data obtained from software and hardware power models of the affected nodes has to be aggregated into a communication energy model.

4. **Distributed Software:** Networked embedded systems can be seen as distributed systems where high level tasks involve the collaboration of several nodes in different locations. With their increasing complexity, the number of distributed tasks is starting to grow. It becomes therefore necessary to obtain energy consumption and efficiency information of each of these tasks separately. This way the designer can easily decide which tasks are compromising the energy lifetime of the overall network.

Therefore, energy profiles can be defined as follows:

Definition 3. An **energy profile** is the aggregation of power models to build new compound models associated to high-level and even abstract elements.

The resulting profile is therefore a model that can be used to estimate both power and energy, restricted by the limitations of the integrating power models. For instance, in the proposed energy aware integral approach, power modeling is limited to average power provided by power state machines.

The goal of these energy profiles is to answer the energy optimization related questions of the designers at different abstraction levels. These profiles must have the appropriate semantics so that the energy consumption data can be appropriately and effortlessly interpreted. They must also enable easy comparisons and evaluation of different design alternatives in order to make the optimal decisions.

Figure 4.3 depicts the different directions in which low level power consumption data has to be abstracted.

The following sections provide detailed descriptions of the different types of energy profiles proposed.

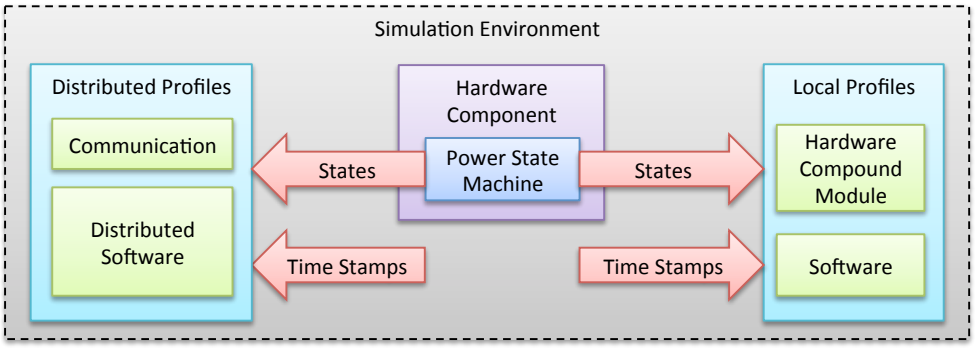

Figure 4.3: Power states and time stamps of state transitions are recorded and interpreted in 4 different directions: hardware, software and communication and distributed software.

4.4.1 Hardware Profiles

The basic element for power simulation is the one for which there exist a power model, which in this work are power state machines as described in Section 4.2. Although a whole System-on-Chip (SoC) or Printed Circuit Board (PCB) belong to the hardware level, they are complex and heterogeneous hardware designs that include smaller pieces of hardware.

Although complex power state machines would be, in principle, possible to be created for a complex system as well, in practice it is better to follow a modular approach. An overall combined power state machine could improve performance as it enables the analysis of the set of joint states and all incompatible states can be ruled out. However, it has two main drawbacks:

1. It hinders the flexibility required, for instance, in hardware exploration, where the state machine would need to be rebuilt whenever a subsystem is modified. Exploration is one of the benefits of early energy simulation.

2. It reduces the granularity of the power states definition, i.e. the power modes of the different subsystems would be masked in overall power states, which can be detrimental to the purpose of this work.

Furthermore, triggering state changes is not a significant limiting factor of simulation performance. Therefore, using power state machines for single components is preferable, as they can be easily reused in different implementations, and enables aggregating power states in the way that best fulfills the requirements of the designer.

However, more complex hardware power models might be necessary. As hardware components are typically organized in a hierarchical way, the proposal to obtain those more complex models is to index the state machines of all the components of a physical or logical hardware cluster.

A **hardware profile** is a power model of a complex hardware subsystem, which consists on the summation of the power consumption values of the current states of the indexed power state machines, as in Equation 4.3, where P_{HW} is the power of the hardware subsystem, S_k is the power corresponding to the current state of power state machine with index i and N is the number of indexed power state machines.

$$P_{HW} = \sum_{i=0}^{N} P_i(S_k) \tag{4.3}$$

The energy of a **hardware profile** is the summation of the energy given by the power state machines of its subsystems.

$$E_{HW} = \sum_{i=0}^{N} E_i = \sum_{i=0}^{N} (\sum_{j=0}^{M_i} P_i(S_{i,j}) \cdot t_{i,j} + \sum_{k=0}^{L_i} k_{i,k} \cdot E_{T_i}(S_{i,k}, \sigma_{i,k})) \tag{4.4}$$

Therefore, every simulated hardware model must contain a set of all the power state machines that are part of it. The model can then provide the power consumption by adding the values of all the power state machines in the set. Figure 4.4 shows the algorithm to obtain the profile of a hardware module with several power state machines.

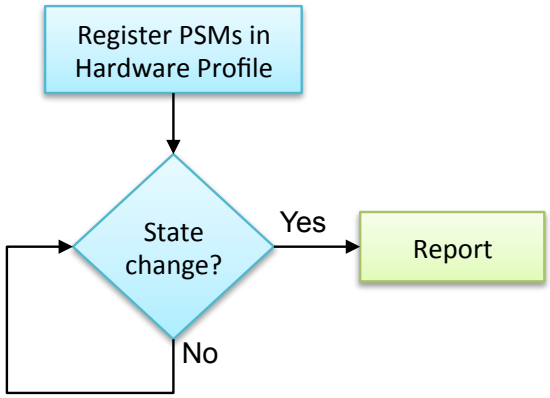

Figure 4.4: Hardware profiling algorithm

Hardware energy profiles permit the automated power consumption data aggregation for complex hardware components and also makes easier to aggregate power consumption data for higher levels of abstraction.

4.4.2 Software Profiles

The lack of energy awareness at higher levels has led to a scenario in which energy consumption is reduced by reducing system performance, e.g. by reducing the duty cycle in which the application is active. However, with the appropriate information, energy consumption optimization should not reduce energy consumption at the expense of system performance, but it should increase the effectiveness of the energy consumed.

Software profiles have been used in computation since concurrency and pre-emption enabled the competition between different tasks. A pre-emptive scheduler must balance the load with fairness. At the beginning this could be done by sharing the access to resources among the different system processes. However, as computation became more and more complex, some high level tasks started to be split into several operating system processes. In order to prevent fairness deterioration, software profiles were proposed in order to account all the processes that belonged to the same high level task. This profiles were called activities. They have been applied to networked embedded

systems in [FDLS08], but only as runtime extension to operating system. Software profiles in simulation of networked embedded systems have not been found.

There are mainly three software development planes in a wireless embedded system:

- The basic software layer, which serves as an operating system and typically includes a scheduler and the drivers and APIs that conform the hardware abstraction layer. For the sake of efficiency, this layer is very slim in wireless embedded systems, removing all unused functionality, which requires memory and consumes power.
- The communication protocol stack, which includes all the network functionality. Communication layers are usually independent and just use and offer interfaces to the layers above and below, respectively. However, power consumption concerns to all the layers of the stack and therefore a cross-layer optimization becomes necessary to achieve energy awareness.
- The application software, which uses all the abstraction provided by the the operating system and the communication stack. Abstraction is necessary to enable productivity in application development. However, it hides the use of resources to the application developer, who lacks the basic knowledge to estimate it properly.

Depending on the development plane, the user might want to define his own high level profiles. To increase the energy effectiveness, it is very helpful to quantify the energy consumption of these high-level defined tasks. For the moment, these high-level activities are still very easy to define, as the energy restrictions on the devices contribute to a very straightforward list of tasks a node must perform.

Although every application can have its own requirements, in most cases a sensor node with a restricted energy budget will not perform tasks other than sensing, processing and transmitting. However, this simplistic approach will not scale well in more complex scenarios and applications. Complexity growth will increase the demand of more and more software profiles to enable isolating the most energy inefficient software elements and evaluating any possible improvement.

In order to offer maximum flexibility and enable energy accounting for both simple and sophisticated software tasks at all possible software development stages, this work proposes user-defined activities.

A **software profile** is therefore a power model for abstract software concepts called activities. The power consumption of an activity is the summation of the power consumption values given by the power-state machines associated to the hardware subsystems involved, while the activity is active. When a sensor node is marked as performing an activity, all the energy consumption data it generates will be labelled with that activity.

Figure 4.5 shows the algorithm to log power consumption and activities in order to build a software energy consumption profile.

An example of power consumption during a "routing" activity is given in Figure 4.6. A message is received that has to be forwarded to another node. Before the reception occurs, the simulator detects that the message has to be forwarded and then the activity is changed to "routing". All the subsequent operations, such as receiving the message, header decoding, routing table lookups, transmitting the message again, etc. will consume power. The power consumption data will be labeled as power consumed in the "routing" activity. This way, a complex task, that involves

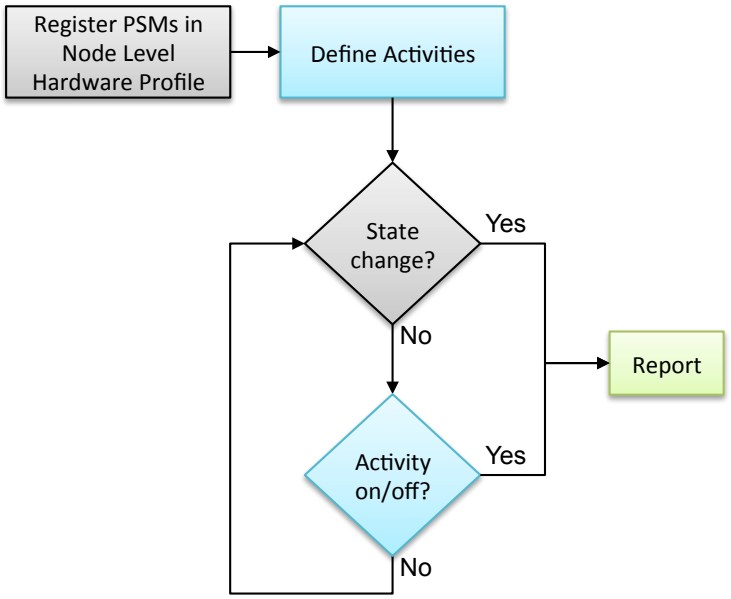

Figure 4.5: Software profiling algorithm

Table 4.3: Software profile example

Routing			
State-transition actions	**Microcontroller**	**Transceiver**	**Time**
Start receiving the message	Sleep	Listen → RX	Starting time
Finalize reception	Sleep	RX → Idle	Size/Bitrate
Start processing	Sleep → Active	Idle	Delay
End processing	Active → Idle	Idle	Processing Time
Carrier Sensing	Idle	Idle → RX	CCA time
Start sending message	Idle	RX → TX	Delay
End sending message	Idle → Sleep	TX → Sleep	Size/ Bitrate

several hardware subsystems will have an associated profile. The different states and transitions of the involved subsystems are shown in Table 4.3.

The energy of a software profile E_{SW} is a function that rules the relationship between any abstract sofware-related concept, or activity, and its accountable power consumption. It can be defined as:

$$E_{SW} : A \longrightarrow \mathbb{R}_{\geq 0}$$

where A is a finite set of abstract user-defined software concepts.

Using the simple power state machine, the energy of the "routing" activity example can be calculated as:

$$E_{SW}(routing) = E_{SW,TRX}(routing) + E_{SW,MCU}(routing)$$

Both transceiver (TRX) and microcontroller (MCU) have associated a power state machine. For the transceiver, the energy profile would be:

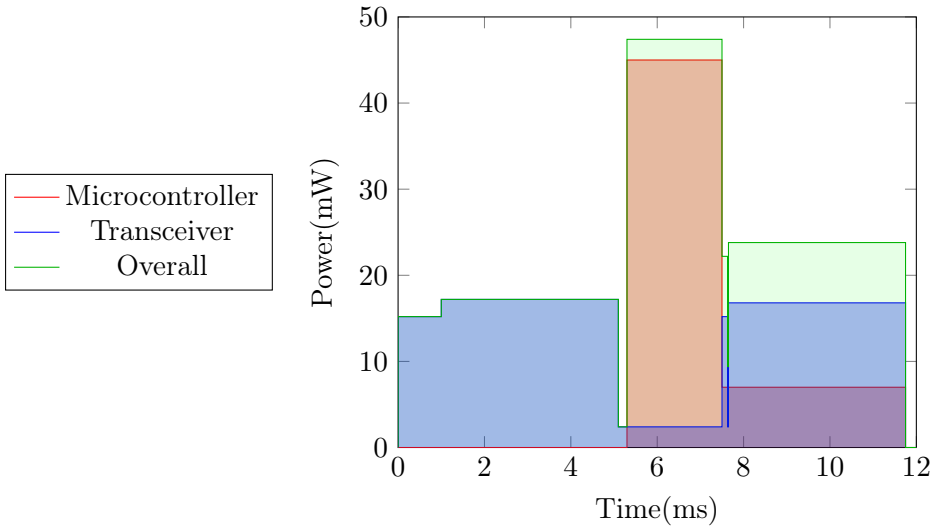

Figure 4.6: Power consumption for the "routing" software profiling example.

$$E_{SW,TRX}(routing) = P(RX) \cdot t_{RX} + P(Idle) \cdot t_{Idle} + P(TX) \cdot t_{TX}$$

And for the microcontroller:

$$E_{SW,MCU}(routing) = P(Sleep) \cdot t_{Sleep} + P(Active) \cdot t_{Active} + P(Idle) \cdot t_{Idle}$$

The time estimation is obtained, as in the general case of power state machines, through simulation. Activities must be defined and triggered by the user, while coding the software application.

Unlike power states within the same state machine, activities are not exclusive, i.e. several activities can occur simultaneously. For instance, both periodic tasks and measurement report could be defined as independent activities. However, measurement reports may occur due to periodic or non-periodic events. Likewise, periodic tasks may involve more tasks.

4.4.3 Communication Profiles

Previous sections discussed energy profiles for hardware and software. Some approaches for both kinds of profiling can be found in literature (see Section 3). However, both cases are only effective for local energy optimization. Nonetheless, in a distributed system, local nodes optimization do not always correspond to system optimization. Therefore there is a missing step in order to enable networked embbeded systems energy optimization. This element is communication. Communication energy profiling is a novel approach proposed here and one of the key contributions of this thesis.

If a complex hardware subsystem takes any action, hardware profiling will aggregate the energy consumption of all its elements in order to obtain the overall energy consumption value. In the case of software, the software profiles will account all energy consumed by the software tasks across different hardware subsystems. However, when a node communicates, his action has an effect in the energy consumed in other nodes in the network.

However, there are no means to account this energy consumption to evaluate the impact that such communications have in other nodes and in the overall distributed system. Furthermore, similarly to hardware architecture exploration, there are many decisions that can be made in the distributed system architecture about how to distribute the computational resources. This decisions will be crucial in the overall system performance and energy lifetime. It is therefore of great importance to create the means to compare and evaluate those architectures for the sake of the system.

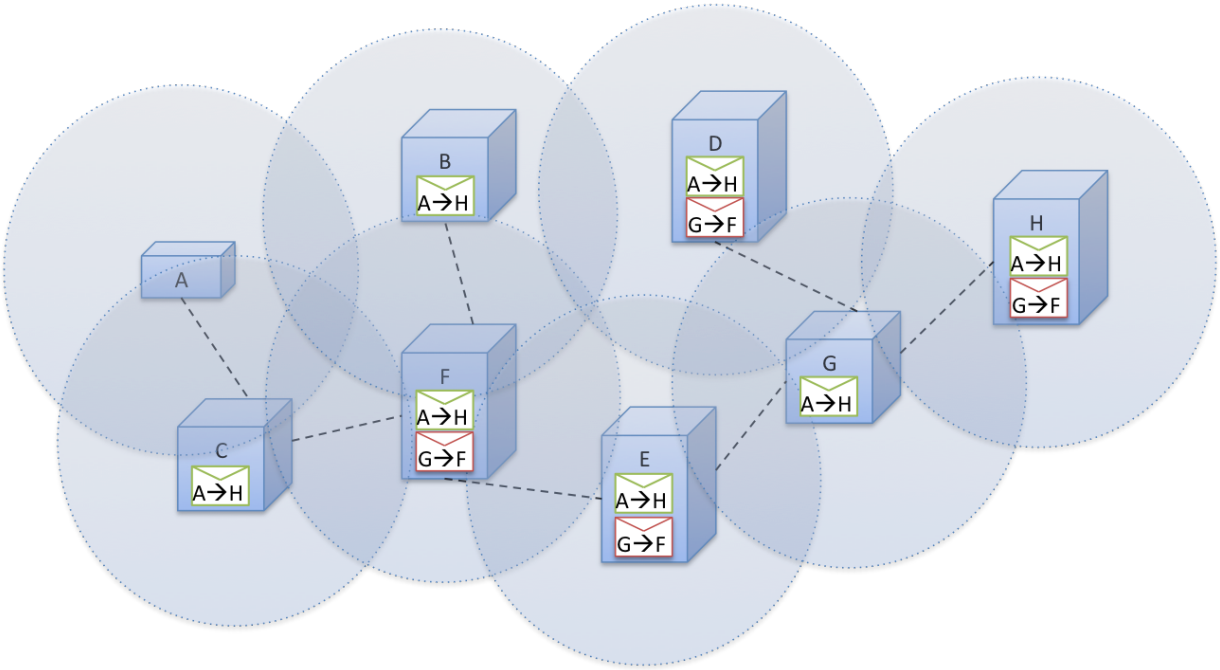

Figure 4.7: Nodes affected by two example communications (A to H and G to F).

In order to account communication, the energy consumed from a single local node perspective is not informative enough. In a wireless ad-hoc network, when a node sends a message, it is broadcast to all nodes in its neighbourhood. Every node listening will at least receive the preamble and decode the destination address to check whether they have to process the message or not. Figure 4.7 shows this effect, with a message symbol for all nodes affected by communication between A and H (in green) and G and F (in red).

If there is a routing algorithm, and before sending a data message, a route discovery is sent, many nodes in the network will be involved in that process and will consume power for it. On contrast, a protocol stack with a more simple routing protocol, might not send route discoveries, but might require more power consumption per communication in order to find the path to the destination node. All these effects must be observable through a communication profile in order to evaluate communication energy consumption and make the best design decisions depending on the application and the scenario. A single change in topology might create a whole new scenario with different traffic balance requirements and different optimum solutions.

A **communication profile** is a power model for communication. It accounts all power consumption, across different hardware subsystems and across different network nodes, that is involved in transmitting a message from the information source to the information destination.

Communication profiling is very costly to implement it at run-time. In wireless communication, messages can be re-transmitted along different paths simultaneously, where some paths just lead

to unsuccessful communication. Collecting the information from all affected nodes is unfeasible in networks with big amount of ultra-low power nodes. However, accounting information from all nodes is absolutely possible in a simulated network.

The communication profile proposed here, uses the data payload as the key element. Communication could be defined as the act in which a sender sends a message to a receiver. The whole process is therefore linked by the message itself. Hence, the idea is that any actor that participates in the communication process, annotates the power consumed for their contribution in the simulated message. These actors are the sender, the transit nodes, the receiver, and any other consuming power element that contributes in transferring the information from its source to its destination. Figure 4.8 represents the algorithm to log power consumption, activities and transaction annotations in order to build a communication energy consumption profile.

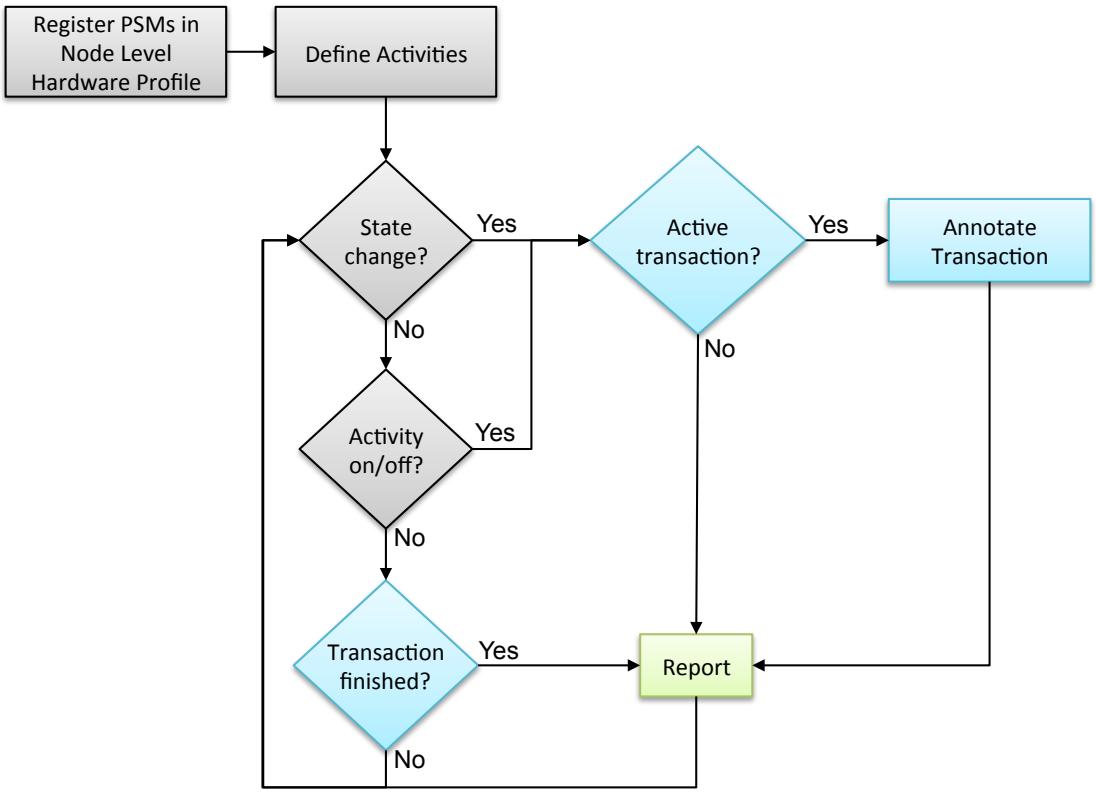

Figure 4.8: Communication profiling algorithm

Table 4.4 shows an example of communication profile for a route discovery message, which also includes the route response. In the example all nodes where listening, but if nodes are duty cycled not all of them would be listening while messages are sent. For the route response, some nodes might just receive the beginning of the message, until they can decide whether the message is for them or not, but this will depend on the implementation. The table shows all the events that will be annotated in the communication profile with their corresponding energy consumption values. Adding all the values will provide the energy consumption of that communication element.

The energy of a communication profile, E_{NW} can therefore be calculated as:

$$E_{NW} = \sum_{i=0}^{N} E_{i,HW} \tag{4.5}$$

Table 4.4: Communication profile example

G → F route discovery							
	Nodes						
Power consuming activities	B	C	D	E	F	G	H
Generate route request						√	
Transmit request			√	√		√	√
Receive route request			√	√	√		√
Process request			√	√	√		√
Listen to responses			√	√		√	√
Generate route response					√		
Receive route response	√	√		√		√	
Process response	√	√		√		√	

where $E_{i,HW}$ is the energy consumption of the i node hardware profile and N the total amount of nodes in the network.

Therefore, the communication profile provides the overall energy impact a communication operation has in the whole network. This way, energy expensive communication can be detected. Furthermore, local node energy estimations can be compared with overall communication profiles in order to find out unbalanced network load distribution. Traffic load distribution is crucial in networked embedded systems, because it might lead to the isolation and unreachability of some parts of the network topology. This is a risk in particular in networks that are established with an unplanned topology. A weakness in this aspect can be identified and corrected, installing additional transition nodes where it corresponds.

4.4.4 Distributed Profiles

Once there are structures defined to profile hardware and software from a local node perspective, and to profile the energy consumed in network communication, new meaningful profiles at the distributed semantic level can be established.

Networked embedded systems are becoming the basis for complex distributed applications. The impact of the different tasks to be performed must be carefully evaluated by the users and administrators in order to obtain the best results possible and prevent failures. An embedded systems network may implement different functionality, such as over-the-air software/firmware upgrade and system configuration, sensing and monitoring, scenario recognition, networking operations, etc. Creating energy profiles to evaluate the impact on the system energy lifetime of these activities is very valuable information.

The problem is similar to the one described in local software profiles. The computational resources are spatially distributed and carrying out tasks involves energy consumption in some of those embedded systems. Distributed profiles must then account the energy consumption across the network for a specific high level task. Hence, those tasks could be named as distributed activities. As in the local software case, these tasks can be defined by the user. However, in the distributed case, multiple activities may take place concurrently. Therefore, it is not possible to just account all network activity in the same energy profile. In order to create the distributed profile it is necessary to account the energy consumed in processing information in specific nodes plus the energy consumed in all communications that are part of the high level task.

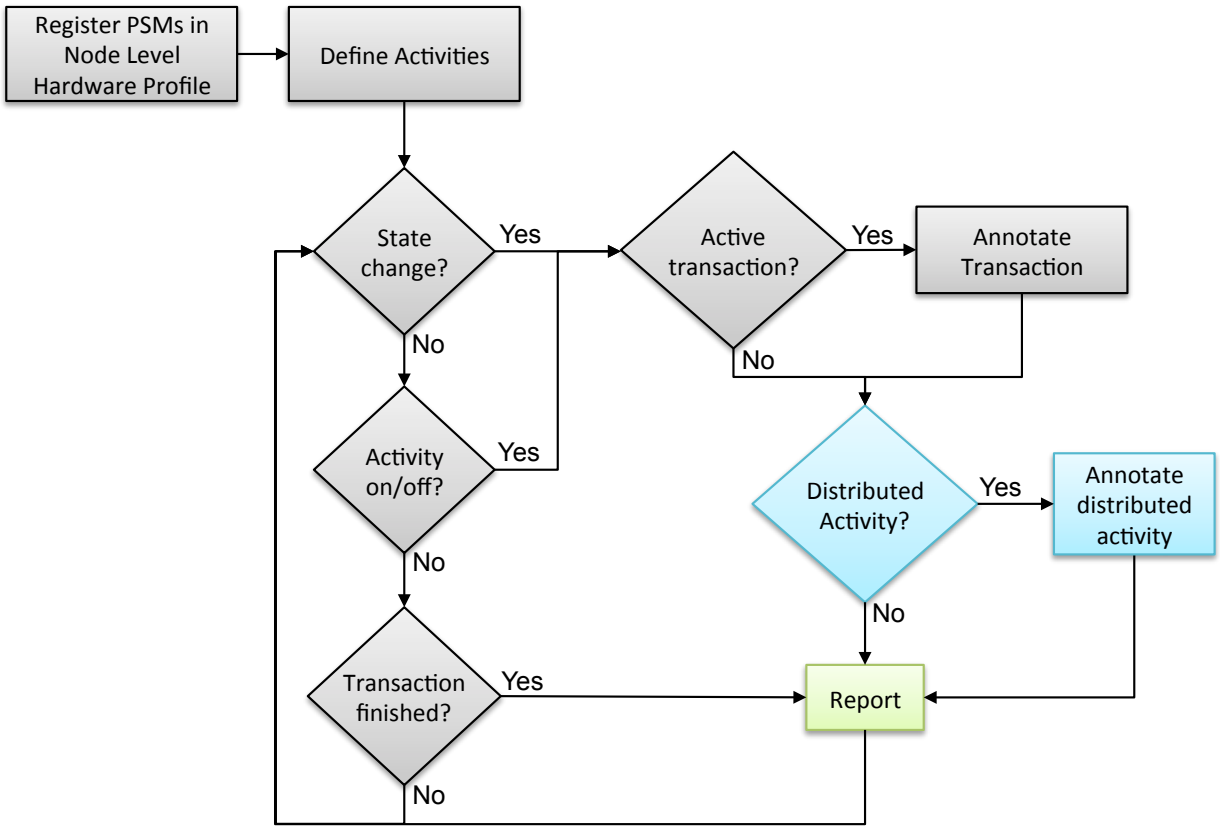

Figure 4.9: Distributed profiling algorithm

Figure 4.9 depicts how to integrate the distributed activities labels into the existing profiling algorithms. All processing that is related to a distributed activity is included in the profile by labeling state and activity changes with the corresponding distributed activity. If there is also communication involved, the transaction is also labeled with the distributed activity, which is then propagated through the network. This way, although some nodes may ignore they are contributing to the high level activity, the simulator can still keep track of this through the communication profile.

The energy of a distributed profile would just be the summation of all the energy consumed during the states and time intervals that are labeled as taking part in it.

5 An Energy Profiling Framework

In this section, a simulation framework called SICYPHOS (Simulation of Cyber-Physical Systems), is presented and described in detail. This framework aims to assist in energy optimization of distributed embedded systems by applying the concepts and techniques proposed along previous chapters, specifically Chapter 2 and Chapter 4.

In Chapter 2, hardware architecture of distributed embedded systems was discussed, as well as all the elements that intervene in the energy optimization problem. Chapter 4 proposed an approach to include energy awareness in design methodology. The proposed approach aims to account energy consumption and associate it to high-level concepts. Simulation plays a central role in the realization of the approach:

- It is required in order to execute the power models and estimate the time that enables obtaining the energy consumption from the average power in the models.
- Comprehensive simulation is required in order to consider virtual prototypes, wireless communication and cyber-physical interaction. All these aspects are crucial in networked embedded systems, which are very open and interactive and their operation is heavily determined by their environment.
- Finally, simulation is the most powerful way to profile energy consumption information. Profiling in the real system is not realistic, as it would modify the energy consumption of the system itself. Implementing external infrastructure is not feasible for networked embedded systems, where all nodes and communication would need to be monitored.

State-of-the-art simulation frameworks, discussed in Chapter 3, do not offer complete solutions to implement the required multi-level and multi-domain comprehensive framework required in order to obtain realistic estimations of energy consumption. Therefore, the purpose of the proposed framework is twofold:

1. Fill the requirements for simulation of networked embedded systems. This includes models and infrastructure to simulate hardware, software, wireless communication and physical processes. Furthermore, the simulator must enable the generation and parametrisation of the whole system, which may contain a big amount of nodes with their spatial location and their own system architecture.
2. Implement the proposed energy aware framework, which must include the power models, the energy profiles and the mechanism to log all the information.

5.1 Simulator Architecture

Before describing the different parts of the framework, the global software architecture must be described. SystemC has been selected as the basis for the whole simulation. As discussed in Chapter 3, SystemC is already an industry standard for hardware and software co-design, widely adopted in the development of embedded systems. Being a C++ library, it is easily extensible. Therefore the framework implemented is a C++ library and an extension to SystemC/TLM library. As a consequence, an additional simulator is not required and the SystemC simulation kernel can be used to simulate the models. In this work all simulations are executed using the SystemC proof-of-concept simulation kernel, which is open-source and freely available for download.

The SystemC simulator is event-based and as a consequence, the SICYPHOS Framework provides discrete-event simulations. Discrete-event simulation is suitable for modelling digital systems, software and network. However, as already mentioned, the SICYPHOS Framework must also provide models for analogue systems and physical environment. Those models typically involve continuous variation according to differential equations, and therefore, discrete event simulation seems not to be the most appropriate approach. The SystemC Analogue Mixed-Signal (AMS) extensions library provides the modelling formalisms or Models of Computation (MoCs) to support analogue modelling. Timed Data Flow (TDF) is the Model of Computation which provides a higher abstraction level and is the one used in SICYPHOS Framework to model physical processes. SystemC-AMS 2.0 Standard added the possibility of establishing a time-step dynamically during execution, enabling, for instance, improving performance when switching between different operation modes with different time precision requirements. This is of special interest in our application scenario, as some systems will operate in both active and sleep states, with extremely different time granularity requirements.

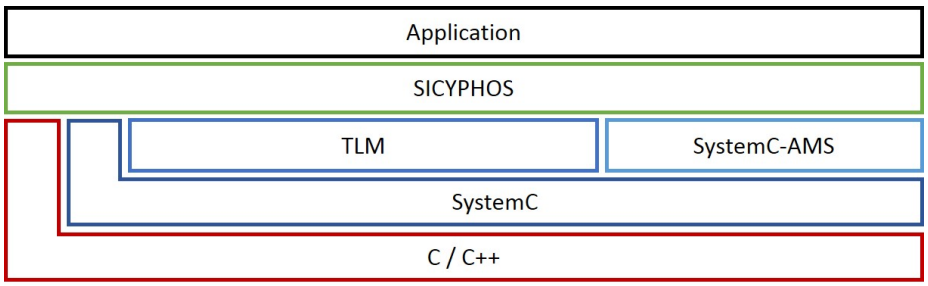

Figure 5.1: Architecture of a simulation using SICYPHOS Framework

Therefore, the TLM modelling paradigm provides the missing infrastructure to fulfil the missing requirements from Section 4.3.

SystemC/TLM and SystemC-AMS extensions, are not intended for simulating networked embedded systems. The proposed framework applies, leverages, adapts and extends the modeling formalism provided by this set of tools, even out of their intended context, in order to serve as a networked embedded systems simulation framework. Figure 5.1 shows how these languages, libraries and frameworks are structured in a simulation. The basis for the whole simulation framework is C/C++. SystemC serves as the simulation kernel that will drive the whole simulation. TLM provides the communication abstraction used in prototyping and in also in the wireless communication model that will be explained next. SystemC-AMS is used to model analog subsystems and also physical processes that are part of the cyber-physical interaction.

The implemented framework can be divided in two main parts:

1. **Wireless TLM:** It contains the models and tools for both intra- and inter-node communication. It includes a top-level API for all systems with wireless connectivity and an XML parser to specify the network topology.

2. **Energy Aware Framework:** This part implements the models described in Chapter 4: power state machines, energy profiles and a logger to efficiently record the simulated information.

Actually a module library that simply collects all the implemented hardware, generic software and protocols for further reuse can be considered as a third part. All three parts are represented in Figure 5.2.

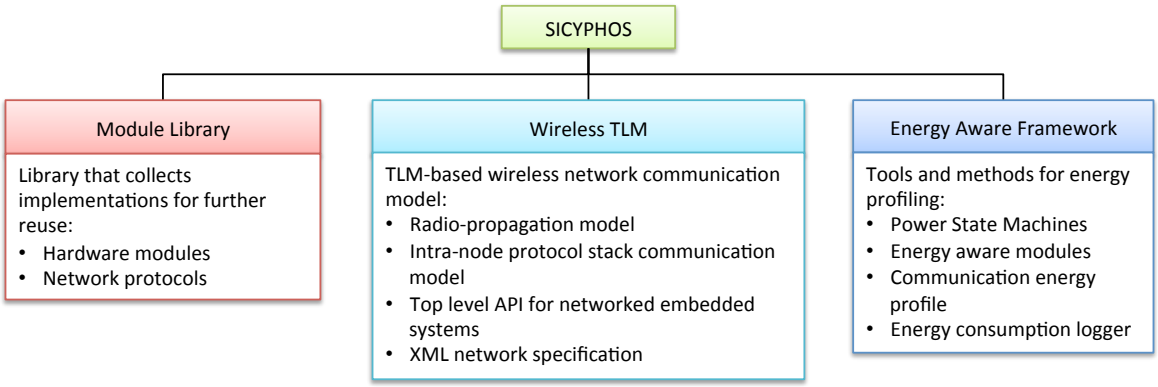

Figure 5.2: Components of the SICYPHOS Framework

5.2 Wireless TLM

The first aspect lacking in SystemC and its extensions, in order to model distributed embedded systems, is a propagation model to simulate wireless communication among embedded systems.

Simulation of wireless communication is crucial for the evaluation of distributed embedded systems. However, the overhead of an accurate propagation model must not hinder the performance of the whole system simulation.

For the sake of performance, SICYPHOS implements the wireless communication model within the SystemC simulation environment. Furthermore, SICYPHOS makes use of the Transaction-Level Modelling approach and extrapolates it to the wireless communication case. This approach was proposed in [DMHG10].

Transaction-Level Modelling was intended to dwell above Register-Transfer Level in digital electronic systems. Bus communication low level details, with pin and cycle accuracy are abstracted into the so called *transaction*. If we consider an interconnect memory mapped bus, this bus would distribute the transaction from the initiator subsystem to the corresponding target. The transaction abstracts all the bit, signalling and pin level information.

SICYPHOS follows a similar approach for wireless communication. In the wireless case, the medium, usually the air, plays the role of the memory mapped interconnect bus. Initiators put their messages in the air, and the air distributes them to all subsystems in range. The target

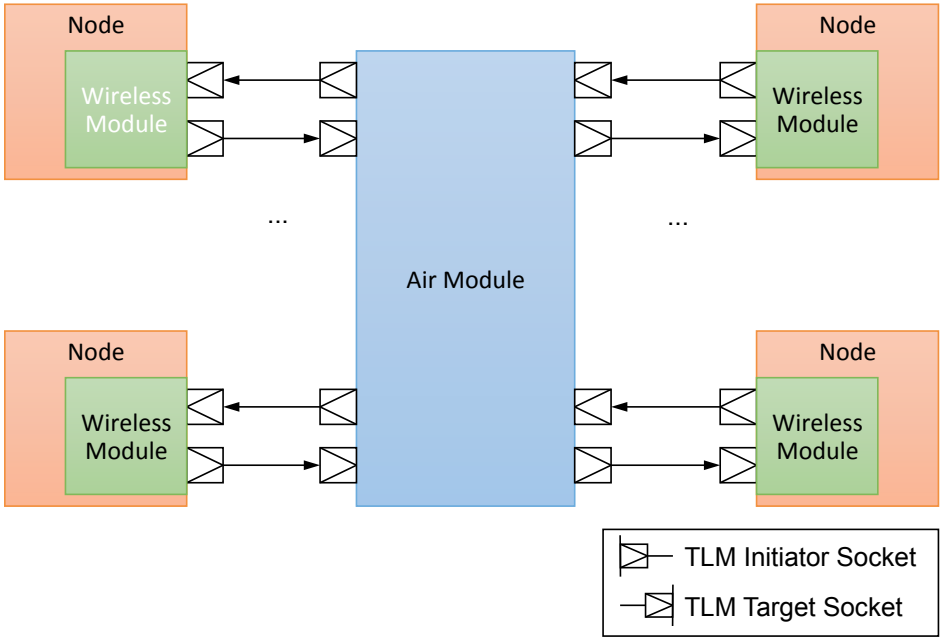

Figure 5.3: Wireless TLM Network Architecture

corresponding to the destination address receives and processes the message. The whole picture of the TLM wireless model is represented in Figure 5.3.

Hence, the TLM wireless model abstracts signals and bits and enables modelling communication at a higher abstraction level. Transmissions are evaluated in terms of bit error rates, obtained from the Signal-to-Noise Ratio (SNR), and delay. The message and the specific protocol headers are also abstracted. The only crucial information that has to be tracked, is the bit length and the protocol bit rate, as they determine the time required to transmit and receive a message, and therefore the power consumed by the transceiver during the power states in which the transmitter and the receiver are active.

5.2.1 Wireless TLM Modules

TLM requires every connected module to be a SystemC module. TLM sockets can then be attached to these SystemC modules. Communication is granted by connecting the sockets and implementing the required communication interfaces.

The wireless communication model in SICYPHOS provides two basic SystemC modules, which are both represented in Figure 5.3. A *Wireless Module*, which is embedded in every node in the network with a wireless interface, and an *Air Module*, which interconnects all *Wireless Modules*.

Wireless modules have both initiator and target sockets, as they can both transmit and receive information. The *Air Module* has multi-target and multi-initiator sockets, as an undefined number of nodes can be connected to it.

5.2.1.1 Wireless Modules

The so called *Wireless Module* represents the wireless communication physical layer. Every node with wireless connectivity must embed this module in order to communicate with other network

entities.

The SICYPHOS Framework provides the `wtlm_module` class, which is an abstract class that contains the terminal side of the communication physical layer. As SICYPHOS Framework aims to be generic and some implementation details are application dependant, such as the bit rate or the modulation, there are some virtual callback functions that have to be implemented by the user. Those virtual callback functions are:

- `accept_air_data_start:` This callback function returns whether an incoming transmission can be actually received or not. It depends on implementation details such as whether the receiver is listening or not.
- `calc_bit_errors:` This callback function calculates the number of bit errors based on the Signal-to-Noise Ratio (SNR). The function to obtain the Bit Error Rate (BER) using the SNR depends on the modulation used for the transmission. Modulation may vary depending on the protocol used, and therefore, has to be provided by the user.

Every wireless module also provides two pairs of sockets a simple initiator socket and a simple target socket to provide bidirectional communication with the air.

Additionally, the wireless module provides another pair of simple sockets to provide communication to the upper communication layers. This sockets can be connected either to additional protocol modules or to the top-level API module, as it will be explained in Section 5.2.3.

5.2.1.2 Air Module

The air module is a SystemC module and is also implemented as a singleton design pattern. This means that during simulation there is one and only one instance of the air class. The singleton pattern contribution is twofold: it prevents from instantiating different air medium classes and it allows every node to acquire the correct instance to be connected with.

The air module must be connected to every wireless module in the simulation. This is done by means of TLM convenience multi-sockets. However, the air must know the multi-socket index assigned to each node, as well as its position, in order to distribute the messages and calculate the attenuations appropriately.

For this purpose, the air contains a registration function that all wireless modules must call in order to obtain connectivity. During this registration, the wireless modules are stored in a map data structure, maintained by the air module, which associates each wireless module to its corresponding socket index.

The Air Module is also responsible for computing all the propagation effects. For this purpose, apart from the socket indices map, every node is also registered together with its spatial location, which is then used by the Air Module to calculate the propagation delay and the attenuation.

Attenuation is considered in two ways:

- **Attenuation matrix:** The air populates and maintains an attenuation matrix based on adjacency and distance. The equation to calculate the attenuation can be parametrized to adapt it to different propagation conditions. For instance, the attenuation exponent in free space condition is 2, while in indoor communications an exponent of 3 or 4 provides more accurate results.

- **Specific attenuation:** In a network there might be many particular cases, which are not well represented by the general distance based equation. For these cases, for instance, when there is an obstacle, SICYPHOS permits establishing specific attenuation values for the corresponding node pairs.

- **Time variant attenuation:** There are time variant effects, e.g. fading, that might affect the transmission, but only in some cases. SICYPHOS permits setting the statistical parameters that better reflect this effects and computes them before every transmission.

With all these effects, the message is processed and distributed to all the nodes where the transmission will have an effect. There is however, a truncation parameter, which is a transmission power threshold value, in order to avoid distributing the messages to nodes where the effect of the transmission would be negligible. This truncation is critical in order to keep acceptable simulation performance in nodes with thousands of nodes.

5.2.2 Wireless TLM Interfaces

Apart from the initiator and target sockets, TLM provides two main transport interfaces: blocking and non-blocking. The difference between both is the level of detail regarding timing.

- **Blocking Interface:** It just models the start and end of a transaction, which is therefore an atomic operation triggered by a single function call. Delay is computed by calling SystemC `wait()`.

- **Non-blocking Interface:** Apart from modelling the start and end of the transaction, it supports modelling other timing points within it, which can be split into several phases with different interactions between initiator and target.

SICYPHOS TLM wireless model makes use of both kind of interfaces in its own two different types of communication, from the nodes to the air, and from the air to the nodes.

5.2.2.1 Wireless Module to Air

Transmitting a message to the air is an operation assumed to occur without interruptions. When the transceiver starts the transmission of the message, it must send all the bits without any possibility of interaction. There should also not be any variation in the signal properties at the transmitter.

Therefore, node-to-air interface can be implemented as a blocking interface, where only some delay can be added. The transaction is then received on the Air Module target socket, and delivered to the Air Module, which will process it to compute the attenuation and forward it to all the nodes within range.

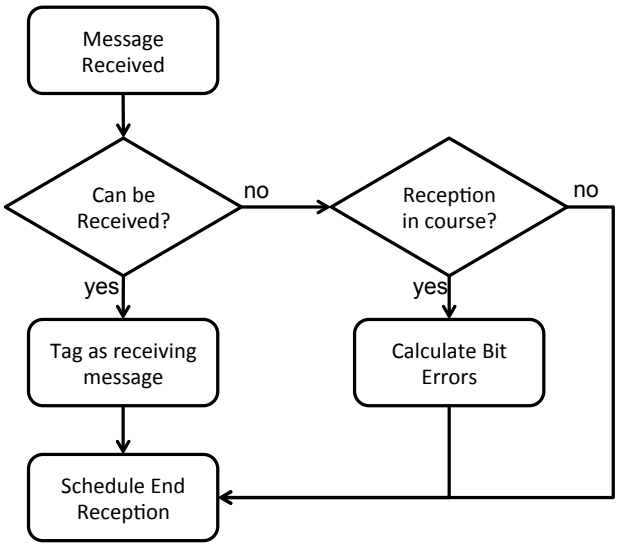

Figure 5.4: Flow chart for Begin Reception phase algorithm

5.2.2.2 Air to Wireless Module

Once attenuation and delay is computed in the *air* module, it forwards the transaction to the target *wireless* modules. From the point of view of the *air* module, the operation is finished just after this forwarding.

However, at reception, the process must be split in order to compute noise and interferences. As a result, and although there is no communication through the backwards path, this transaction is better implemented using the non-blocking interface.

There are two phases: one to indicate the beginning of the reception and another to mark the end.

- **Begin Reception:** When a message reaches the node, the total received power value is increased with the transmission power of the message. Then, there are two alternatives: either it can be actually received by the node and the message is treated as data, or it cannot, and the message is treated as noise. In the former case, the messaged is tagged as being received. In the latter case, if there is a reception in course, the callback to calculate bit errors 5.2.1.1 is called in order to calculate the bit errors until the reception conditions (SNR) change. In any case, an event is schedule at the time in which reception ends, according to bit rate and message length. The whole process is represented in Figure 5.4.

- **End Reception:** When the event queue delivers the end reception event, the total received power is decreased by the value of the corresponding message transmitting power, and there are again two possibilities. If the message is actually being received, the bit errors are calculated according to the Signal to Noise Ratio (SNR) and the modulation, and the data is delivered to the upper layers. If the message is not being received, and therefore, treated as noise, if it is not affecting any message reception, it can be discarded. If there is a reception in course, then the callback for bit error calculation must be called in order to compute bit errors until the change in SNR. Figure 5.5 illustrates the process.

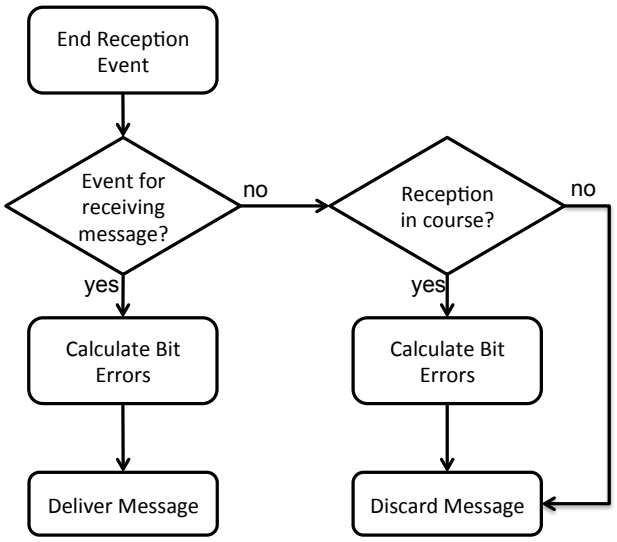

Figure 5.5: Flow chart for End Reception phase algorithm

A blocking interface for the Air-to-Wireless-Module communication has also been implemented. This interface is more simple and efficient but it does not simulate collisions during reception: the same propagation conditions are assumed from the beginning to the end of the message reception.

5.2.3 TLM Intra-Node Communication

The wireless TLM model just covers the communication physical layer. However, wireless communication typically involves some protocols, at least at the MAC and network layers. Therefore, the whole communication stack must be implemented.

A full implementation in the target platform native code, would be necessary to estimate the processing time for the different tasks and calculate the power consumed in the microcontroller. However, during distributed system architecture exploration such an implementation would not be feasible.

For this reason, SICYPHOS offers infrastructure to rapidly implement communication protocol stacks at least at a functional level, in order to estimate the communication overhead and evaluate the behaviour, at the network level, of different communication protocols and network topologies.

Actually, one of the most powerful and unexplored optimizations a distributed embedded system designer can perform resides, in fact, in the communication stack. SICYPHOS aims therefore to provide flexibility to configure different protocol stacks in order to easily evaluate and compare different protocols.

Hence, the impact in energy consumption of adding, for instance, collision avoidance at the MAC layer could be assessed. Network protocols can also be explored to estimate which one provides a longer distributed system lifetime.

SICYPHOS leverages TLM interoperability features to provide the required flexibility. It provides a protocol class with two pairs of sockets, one for bidirectional upwards communication and the other for bidirectional downwards communication. The delivery of a message from one layer to

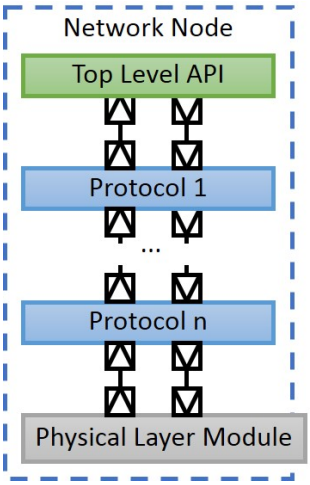

Figure 5.6: TLM Intra-node Network Protocol Stack Architecture

the adjacent one, is assumed atomic, and, consequently, is implemented using the TLM blocking interface.

Every protocol must extend the basic protocol class and implement the corresponding TLM transport interfaces. This way, the user can set up his own protocol stack just by connecting the sockets in the appropriate order. The protocol at the bottom has to be connected to the wireless module (see Section 5.2.1.1, while the protocol at the top has to be connected to the top level API module (see Section 5.2.5.

If no protocol is implemented, the sockets of the wireless module can be directly connected to the top level API. Figure 5.6 shows the TLM architecture of a SICYPHOS protocol stack.

However, to achieve real interoperability, not only the communication sockets have to be present. Messages are changed by the protocols, by adding some protocol specific bits as message headers. To model these headers, TLM generic payload extensions are used (see Section 5.2.4.3).

5.2.4 The SICYPHOS Transaction

One of the key elements of the SICYPHOS approach is the concept of transaction. A transaction in SICYPHOS is the central communication data structure: it is used for both the wireless and the intra-node communication models. As communication is implemented using TLM, the SICYPHOS transaction has to be based on the TLM generic payload class, which is the main parameter of TLM interfaces.

The SICYPHOS transaction fulfils two main functions:

- Efficiently simulates communication.
- Serves as the basis for the communication energy profiling.

However, the TLM generic payload is conceived for memory mapped bus communication. Therefore, for the sake of SICYPHOS communication model, some attributes have been reused for different purposes and others had to be added either by C++ inheritance or by the TLM generic payload extensions mechanisms.

5.2.4.1 Adapting TLM Generic Payload

In this section, the different attributes of the TLM generic payload are enumerated, as well as how these attributes will be reused in SICYPHOS Framework.

The TLM generic payload, as the name already suggests, is design to be versatile, in order to be usable in most common communication scenarios. Therefore, some of the original TLM generic payload attributes can be reused. The following are also used in SICYPHOS, with a slightly different meaning due to the context change:

- **Address:** Contains the address of the final destination node for the corresponding message.
- **Data length:** Contains the value of the length, in bits, of the payload data contained in the message.
- **Data pointer:** Contains the pointer to the data carried in the payload.

The rest of the generic payload attributes are ignored.

However, these attributes are not sufficient for SICYPHOS purposes. Thus, SICYPHOS includes its own transaction class, derived from the TLM generic payload, which, apart from the previously mentioned attributes, includes the following:

- **Transaction ID:** Identifies the transaction with an unique ID. This attribute is just for debugging and simulation purposes. It facilitates logging the information for each transaction.
- **Headers Length:** Contains the length in bits of the headers added to the data message. This way, the full message length can be obtained, by adding data and headers length attributes. The headers themselves will be part of the transaction by using TLM generic payload extensions mechanism.

5.2.4.2 Memory Manager

A common practice when using TLM generic payload, suggested in the SystemC Language Reference Manual [IEE12], is to use a so called *memory manager*.

SICYPHOS includes its own memory manager implementation, conceived as a singleton design pattern. The memory manager prevents the simulation from instantiating and destroying generic payload objects for every transaction. Instead, a common pool of transactions is kept.

Whenever an initiator needs a new transaction it asks the memory manager for it, which provides one of the unused ready instantiated transactions from the pool. When the transaction is not longer in use, the memory manager resets it and returns it to the pool. This way, a new transaction is only allocated if there is none available in the memory manager pool. In order to control when to reset a transaction and put it in the transactions pool, a reference counter is used.

Apart from the improvement in performance obtained from avoiding memory allocation, there is more control over the memory usage. The potential memory leakage risk is restricted, as it reduces the number of instantiated transactions.

5.2.4.3 TLM Payload Extensions

TLM provides a generic payload extension mechanism which permits customizing the transaction data structure for different purposes other than the memory mapped buses.

Hence, when the attributes provided by the generic payload class are not sufficient, further attributes can be added to an extension.

Moreover, generic payload extensions are ignorable, i.e. they do not need to be required in order to fulfil communication. The user can decide whether to make them ignorable or mandatory, depending on his needs.

Unlike message encapsulation, extensions do not have to be decoded in a specific order to correctly interpret the data contained in the payload, improving interoperability.

As a result, different communication set ups can be evaluated very fast, using interoperability features. However, interoperability must, at some point, lead to specific implementation. But the user is always capable of adding the appropriate rules, in later stages, to evaluate the communication details required for verification.

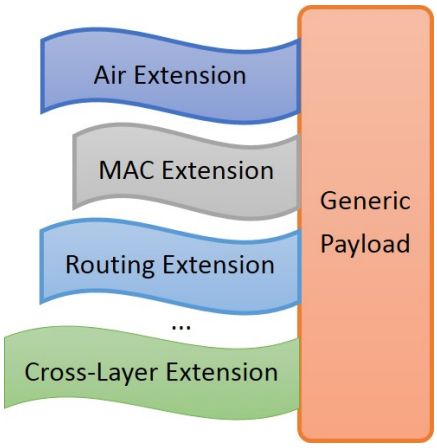

Figure 5.7: TLM Payload Extensions for Network Headers and Simulation Information

Figure 5.7 depicts a generic payload with its extensions. The main aspect of extensions is that they do not encapsulate the messages like bit headers. On contrast, extensions are attached, like labels, which can be read and written in any order.

5.2.4.4 Wireless TLM Transaction

Once the TLM transaction and generic payload concepts have been introduced, the SICYPHOS wireless TLM transaction can be explained in detail.

Apart from the adaptation required for the different communication purpose, which has already been explained in Section 5.2.4.1, there are several new concepts and optimizations that have to be performed for the wireless communication case.

When designing the Wireless TLM transaction, two main requirements were driving the implementation:

- Message exchange has to be as light as possible, in order to improve simulation performance. Most network simulators are not usable for distributed embedded systems due to their poor performance. However, although the network behaviour has to be simulated, for SICYPHOS purposes, network details can be abstracted in order to improve simulation efficiency.
- The final purpose of the SICYPHOS Framework is to perform energy optimization. From a functional point of view, a simple message exchange has no meaning in terms of energy efficiency. In such redundant and non-reliable networks, many messages are useless for the overall functioning of the system. Therefore, what matters in an energy efficiency analysis is the actual task performed by the distributed system.

Those two requirements lead to the wireless TLM approach followed in SICYPHOS.

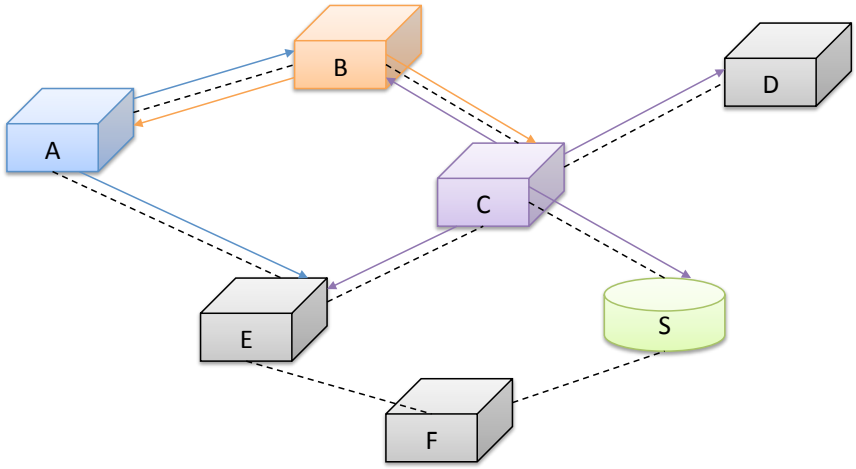

Figure 5.8: Network with all transmissions involved in sending a message from node A to the destination sink node S.

The SICYPHOS transaction does not represent a single network data message. In fact, a single SICYPHOS wireless transaction is used to simulate all messages created in the network in order to perform an end-to-end transmission. Figure 5.8 illustrates this transaction concept.

The figure shows a wireless network. Node A sends a message with the sink node S as the final destination. Depending on the protocols and routing algorithms, the behaviour of the nodes receiving the message may vary: some will not have the receiver switched on (in grey), and will not be affected. Other nodes might be responsible for forwarding the message (coloured nodes) to their neighbourhood. On its way to the destination node S, the message from A will have reached nodes B, C, D and E, will have been ignored by E and D and forwarded by B and C.

All transmissions that take place in the network due to node A sending a message to the sink node S, have the same payload. Therefore, in SICYPHOS, they will share a single generic payload data structure. On the other hand, there is also variable data, such as headers or simulation parameters, which are kept in their corresponding generic payload extensions. This approach has the following advantages:

- As the same data structure is used for the whole end-to-end communication, no copies of the first messages are instantiated. As a result, less memory is used, and there is no overhead due to new message copies instantiation and allocation, enhancing simulation speed.

- Transaction data structure has a more consistent functional meaning than network messages. It contains information about the whole history of a message transmission: where it was originated, what is the final destination, through what nodes it was forwarded, etc. This information enables a thorough analysis of communication behaviour, where efficiency can be much better evaluated.

However, on the other hand, the implementation of this approach has also its drawbacks. The major drawback is that, although the payload of the message should never change during re-transmissions, some bits on the headers and some other information has to change in every hop. However, several changes may happen in parallel which might affect data consistency.

Therefore, the changing part of the transaction, has to be stored and addressed in a proper way so that the appropriate values are always used. As every node can just receive one message at a time, the changing information, which is stored in generic payload extensions, is stored in maps, whose keys are the identifiers of the nodes in which the message is being processed. Furthermore, as messages may even pass through the same node twice, and transaction history is also of crucial importance for energy optimization purposes, the information corresponding to a node is stacked so that former transmissions can also be analysed.

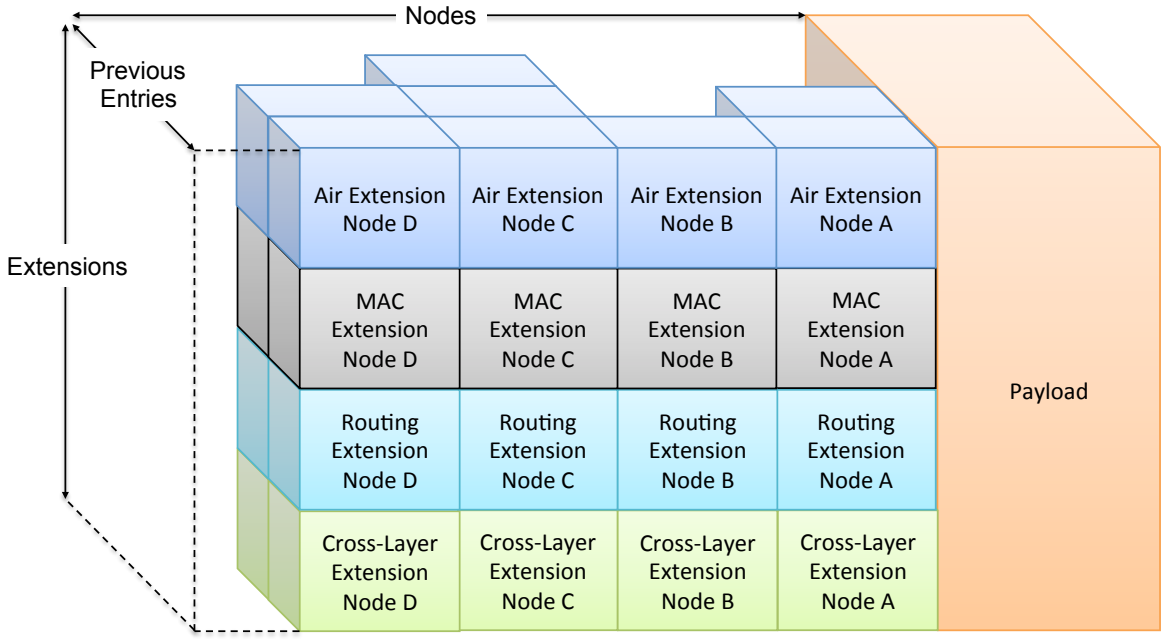

Figure 5.9: SICYPHOS Transaction. There is an immutable data payload plus a multi-dimensional structure for storing variable data, organized according to the nodes involved, the data abstraction layer and the sequence in the lifetime of the transaction.

Figure 5.9 shows how all the transaction variable data is organized using the transaction extensions. Every extension has a different colour and consists of a map, that indexes the information using the node identifier as the key. As due to meshed structure, a message might reach a node several times, the mapped value not only contains the most recent data but a vector containing all previous entries.

If we apply the transaction concept to the network scenario in Figure 5.8, the SICYPHOS transaction would look like the one shown in Figure 5.10.

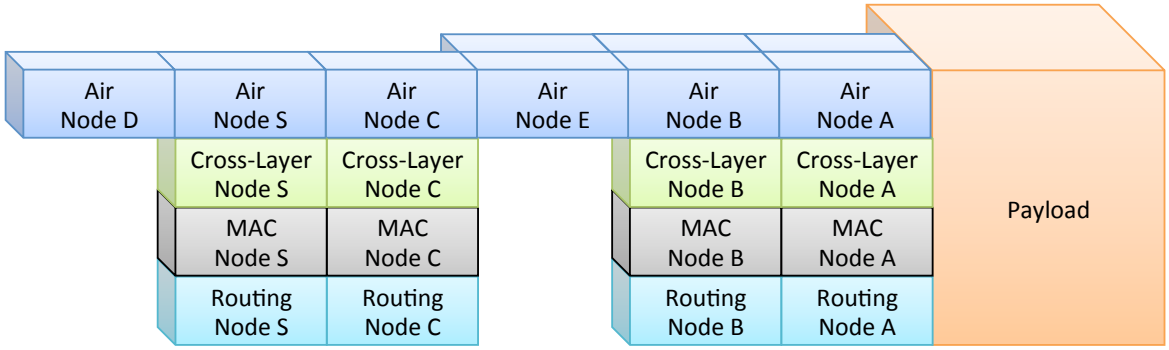

Figure 5.10: SICYPHOS Transaction for the scenario in Figure 5.8

In order to simplify this approach for the user, SICYPHOS Framework provides a protocol extension template class. This template class implements all the method overrides that TLM requires for any generic payload extension, in order to handle it appropriately, such as those to copy or reset. It also simplifies the addition of parameters to the extension, so that the user must only make use of two methods:

- `push`: This method takes the module name and a template class as parameters and stores them in the internal map. The module name would be the key and the template class the mapped value.
- `get_extension_entry`: This method takes the module name as a parameter and returns the most recent mapped value associated to it.

The user must therefore define a protocol extension entry with the parameters that would be part of the corresponding header. After this, he can just store or obtain them by using the aforementioned methods.

5.2.5 Top-Level API

The top-level API is the interface for the application code. Every node has two sockets (initiator and target) for bidirectional communication and its own event queue. The API is therefore related to both features.

The constructor just requires the SystemC module name parameter, as every SystemC module. It defines then a SystemC thread, activated by events in the event queue, which is the main loop and entry point for the application.

The `main_loop()` method calls the initialization method of the node and enters a control loop which iterates whenever there is an available event in the event queue. For each event it calls the method to handle it.

Apart from the main loop method, the top-level API provides a method `send_message()`, which is used to send transactions through the downwards communication path to the communication layer immediately below.

Finally, the `deliver_upwards()` method implements the blocking interface for receiving transactions from the lower layers. Every communication message will be received here and will be encapsulated in a new event, which is added to the event queue.

The top-level base class also maintains a pointer to the corresponding physical layer module, which has to be assigned when building and registering the node in the environment.

To complete the API there are two virtual functions that have to be implemented by the user:

- `initialize()`: It contains the initialization code for the node application. The most important task is to add the initial event to the event queue, so that the application can start.
- `handle_event()`: This is the callback function for handling events from the event queue. Whenever an event is taken from the queue, this function is called. The user must therefore implement the interpretation of the event type as well as the corresponding actions to be performed.

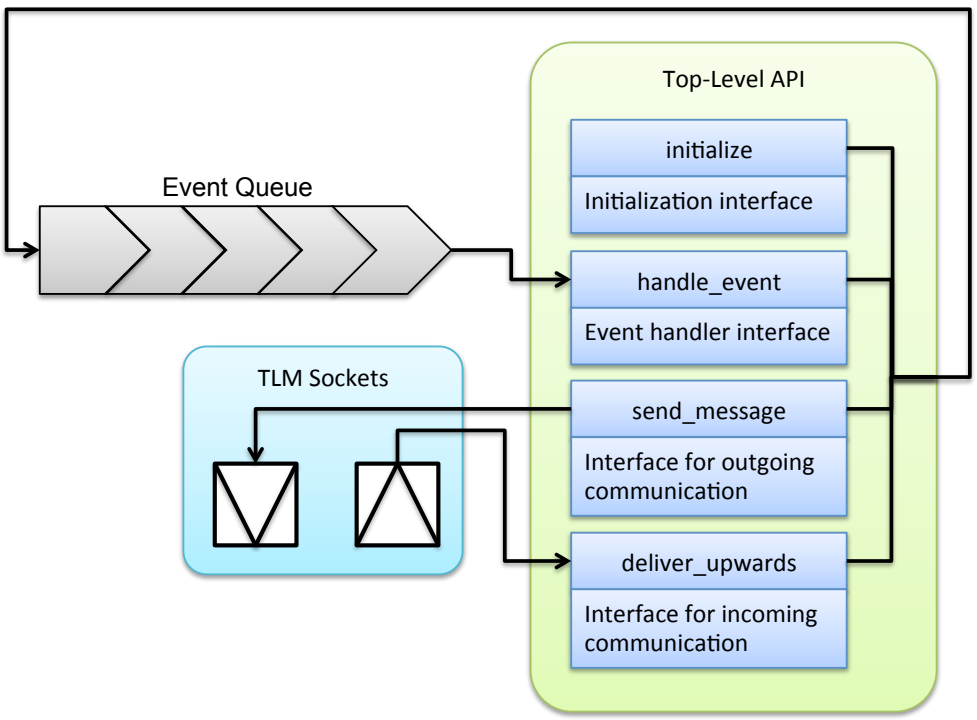

Figure 5.11: Node base structure. Contains an event queue, incoming and outgoing TLM sockets and the interfaces to use them.

The whole picture of the top-level API module and how it works is shown in Figure 5.11.

5.2.6 Nodes Parametrisation

One of the main difficulties when simulating distributed embedded systems is to be able to instantiate a big amount of nodes, which can be of very different types. Sensor networks are typically heterogeneous, with different nodes having different hardware and software.

It is therefore necessary to provide a mechanism to instantiate and parametrize all the nodes in the network in a simple and efficient way. Besides, in order to explore different possibilities, it is also a requirement to provide a mechanism to change the network without recompiling the simulation.

For this purpose, SICYPHOS provides an XML parser, based on RapidXML header library [Kal09], which is open source and requires no linking, avoiding any compatibility issues.

Using SICYPHOS parser, the user can specify the network topology by specifying each node with the following parameters:

- **Node name:** Every node is a SystemC module, and therefore, must have a name to identify it. This will also be helpful for debugging.
- **Node type:** As in a single network there can be very different types of nodes, this parameter permits identifying them, so that the simulator is able to know, which kind of node to instantiate.
- **Node position:** The transmission properties of the network is based on the 3-D position of each node, and the distances among them. Thus, a 3-D position must be specified, by providing the coordinates x, y and z.

The XML parser is part of the simulator. However, node architectures are defined by the user. To be able to instantiate user-defined nodes using the embedded parser, SICYPHOS provides an implementation of the Factory Method design pattern [GHJV95], named `node_factory`.

Figure 5.12 shows how the factory method is used to automatically instantiate user defined node classes using the frameworks' XML parser.

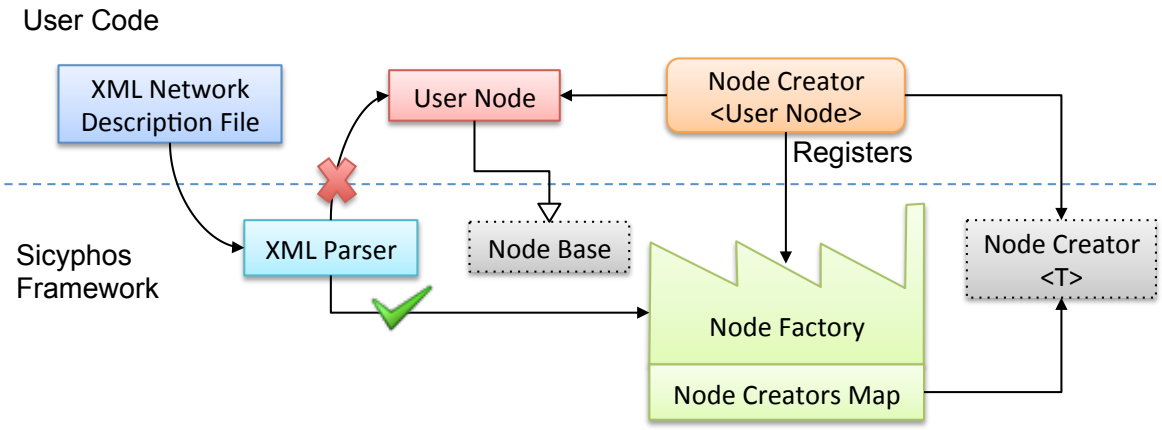

Figure 5.12: User defined nodes cannot be instantiated from the XML parser. To solve that, the user must register a Node Creator element in the Node Factory. The XML parser can then use the Node Factory to create the registered node implementation.

The `node_factory` has a template class `node_creator` that instantiates a node of the template type, with parameters for the SystemC module name and the spatial position. It also registers the node in the air module so that the wireless interface is fully operative.

The `node_factory` is then a singleton class that maintains a map with the node type as the key and the corresponding creator method as the mapped value.

The `node_factory` class provides then the `register_creator()` function that registers a node type, which will be specified in the XML configuration file, with the corresponding `node_creator` template specialization. The user must register all the node types in the main program before calling `sc_start()`.

The parser can then call the `create()` function in the `node_factory` to instantiate the user-defined node automatically, with its corresponding name and spatial location.

5.3 Energy Aware Framework

The final purpose of the system simulation in the context of this thesis is energy consumption simulation and profiling. Therefore, apart from the network and embedded system models, that are the basis for the overall simulation, the central part of the implemented approach is to create a framework to generate power consumption information and to track this power along simulated time in order to create the energy profiles discussed in Chapter 4.

This energy aware framework consists of three main parts:

1. **Power Models:** Based on power state machines, and in particular on those defined in Section 4.2. This power state machines do not intend to be novel or particularly accurate. They just serve as the energy consumption information source to later create the energy profiles.

2. **Energy Profiles:** This is the central part of the framework, specially those concerning the distributed nature of networked embedded systems, which are insufficiently covered by State-of-the-Art simulations.

3. **Simulation Logger:** Models and profiles are not complete without a simulation logger that efficiently records all simulation outputs to enable further improvements and optimization of the simulated system.

These three parts will be explained in detail next.

5.3.1 Power State Machines

The energy aware framework supports the power modelling approach using finite state machines proposed in Section 4.2. For that purpose, it includes three main classes: power state, power transition and power state machine.

The main implementation concept for the power state machine is using a similar approach to linked lists data structures. The power state machine is therefore a reference to the current power state, which contains the links to all possible next states.

Next sections describe all the elements of the power state machine implementation.

5.3.1.1 Power States

The class `power_state` models every system configuration in which power consumption is stable and converges into a constant average value. It consists on the following main attributes:

- **name:** An identifier for the power state, e.g. `tx_on`, `sleep`, `deep_sleep`, etc.
- **power:** Average power consumption at the corresponding state. This value can be measured or obtained from the data sheet. Integrated over the time spent on the power state, it will provide the energy consumption.
- **transitions:** It is a vector that contains a list of all possible state transitions that can be triggered from the state. This list is used to prevent the user from triggering impossible state transitions. Therefore, the power state machine also serves to control the simulation and as a high level model of the digital subsystems.

As every power transition has its own characteristics in terms of delay and power consumption, the `power_state` class includes an internal class to model state transitions.

Power Transitions

The class `power_transition` is an internal class of power_state, which models the transition between two states. It has the following attributes:

- **Next State:** The destination state the system will enter when the transition finishes.
- **Delay:** The time spent in the transition, which is the time the system needs to change from one stable power consumption state to another.
- **Energy:** The energy consumed in the state transition.

During transitions, power consumption usually varies significantly and a constant power consumption cannot be assumed. As the model is focused on estimating energy consumption, state transitions provide a fixed energy consumption value instead of a power consumption value.

5.3.1.2 Power State Machine

Finally, the framework includes a class to model the Power State Machine. As already mentioned, the power state machine is implemented like a linked list. It provides the API for the user, which has the following functions:

- `init_state_machine:` Initializes the state machine with a state given as parameter.
- `get_current_state:` Returns the current state of the state machine.
- `change_state:` Changes the state to a new state given as parameter. The power state machine checks if the new state is listed in the transitions vector of the current state. If it is, it changes the state and reports the state change to the simulation logger. Otherwise, it throws a forbidden state transition exception.

For logging purposes the state machine also includes a memory of 1 state, i.e. it also contains the previous state, a name or identifier, and a pointer to the SystemC module that it is associated with.

SICYPHOS energy aware modules maintain a collection of the power state machines that model their energy behaviour. When a power state machine is created, it is also registered in the collection. Hence, in order to obtain the power consumption of a SICYPHOS energy aware module, a getter function will just return the summation of the power values of all current states of all registered power state machines.

5.3.2 Energy Profiling

The power state machines explained before provide the means to model power consumption of hardware subsystems. However, it is still necessary to bring the information from this models to higher semantic levels, which will make this information usable for energy optimization of hardware architecture, software and network. Energy optimization not only concerns hardware design, but also higher levels, such as operating system, communication protocols, application software, network design, etc. As explained in Section 2.5, high level optimization is barely unexploited and can provide significant improvements in overall system energy efficiency.

The Energy Aware Framework supports energy profiling and provides several methods of energy consumption accounting, that will be described in the following sections. In Chapter 2, energy profiles were divided into four different groups, which are all considered in the Energy Aware Framework:

- **Hardware:** Power consumption data originates in hardware power models. However, there are complex hardware architectures and it becomes necessary to provide the energy consumption of complex hardware structures, which might involve several power models.
- **Software:** SICYPHOS offers software activities definition by the user, which automatically account for energy consumption occurring on the device. These activities cover all kinds of software in a a wireless sensor node: operating system and hardware abstraction layer, communication protocol stack and application.
- **Communication:** The SICYPHOS Framework assists the designer by providing the means to track communication to estimate its efficiency. This is done through a energy extension to the wireless TLM transaction, as it will be explained next.
- **Distributed Software:** The central energy logger provides the means to create distributed activity profiles that account all the energy consumed all over the network in high level distributed tasks.

All these profiles are explained next. The way to account energy can be classified in two categories, depending on the main entity energy consumption is associated with.

1. **Physical Device:** This is the local perspective where hardware and software energy consumption is indexed by the physical node or embedded system where the power is consumed. The framework element responsible for tracking this information is the **energy aware module**.

2. **Distributed Logical Unit:** This is the distributed perspective, when some software or network tasks involve power consumption across different nodes in the overall system. The framework element responsible for accounting all distributed energy consumption is the **energy aware transaction**.

All the information is logged through a central energy logger, which records the complete data with all the labels provided by the profiling methods, so that the simulation output can later be rendered and analyzed.

5.3.2.1 Hardware Energy Profiling

Power models in the modelling framework are based in system-level Power State Machines. However, architectures in networked embedded systems are increasingly complex. They are many times systems-of-systems, where it becomes necessary to obtain the aggregated energy consumption of different subsets of subsystems.

In the framework, the aggregation of Power State Machines is enabled by the Energy Aware Module, which is an extension of the SystemC module base class `sc_module`. Every hardware module that will have associated an energy profile must extend the `energy_aware_module` base class. The interfaces provided by this class in order to create energy consumption hardware profiles are:

- `register_state_machine:` All state machines that belong to the module can be stored in a vector using this function. The vector can be populated manually or automatically making use of the parent-child relationship of SystemC modules.
- `get_current_power:` This function returns the power consumption of the module at the current time. Power is calculated by summing the power consumption values of all the current states of the power state machines contained in the state machines vector.

With this functions, the user can easily obtain the power consumption of a compound module, although the power state machines are only associated with its children modules.

5.3.2.2 Software Energy Profiling: Activities

Developing software for embedded systems is typically done using an Application Programming Interface or an Operating System that abstracts the hardware in order to increase productivity and enable tackling more complex tasks.

The downside of this abstraction is that the software developer loses the awareness of what underlying hardware is being used for the different software tasks. Initially, embedded systems had very simple architecture, with single and simple microcontroller, a transceiver and some small group of peripherals. The programmer could handle easily the different power modes of the transceiver and the program flow was easy to follow as there was only one main process that handled everything. However, complexity is continuously increasing in order to achieve optimizations. Additional subsystems are being integrated in order to perform more operations. Furthermore, some subsystems

are replicated, with more specific and power efficient units, that can deal with some specific situations and save a lot of energy in combination with some aggressive power management policies.

For instance, a very low power wake-up receiver could be used to relieve the main transceiver from listening intervals that consume too much energy. This receiver cannot deal with the whole communication process but is capable of identifying incoming messages that should be handled by the device. Likewise, ASICs, reconfigurable logic or even low power specific microcontrollers are frequently used to deal with some tasks that could have been executed by the microcontroller, but with less energy efficiency.

Therefore, the software developer can nowadays very easily lose track of what is happening behind the scenes. SICYPHOS provides software profiles to prevent this. These profiles are called activities. There are some activities proposed by default, but the programmer can extend this enumeration with his own self-defined profiles.

As in hardware profiles, the object that the framework provides to create software profiles is the Energy Aware Module. Every Energy Aware Module has an activity attribute that can be switched by the user. All the operations performed while an activity is active, can therefore be accounted under the same activity label.

The Energy Aware Module provides therefore a C++ set of operative activities. It provides then the following functions:

- `activity_on:` Activates the activity provided as parameter by inserting it in the activities set. It automatically creates a report to the energy logger.
- `activity_off:` Deactivates the activity provided as parameter by erasing it from the activities set. It automatically generates a report to the energy logger.
- `get_activities:` Returns the set of current activities. This function is typically used by the logger to write all active activities in the energy profiles.

The energy logger will track all changes and will record all active activities in every logged entry, so that the user can later calculate the energy consumed in realizing every activity.

5.3.2.3 Communication Energy Profiling: Energy Transaction Extension

The most important energy profile in Networked Embedded Systems are the communication profiles. The distributed nature of Networked Embedded Systems plays a major role in the energy optimization problem. The distribution of resources affects even the hardware exploration which is the basement for all energy optimization.

There is a trade-off between the computational capability of a device and its needs of communication. A very simple unit consumes less power but is not capable of processing the information to make more efficient use of the wireless channel. A very smart unit might be capable of processing the data and even be able of executing the algorithms to make the required decisions with no need of communication with a central unit.

Therefore it is specially important in Model-Based Design to be able to estimate which distribution of computational resources will be more energy efficient. In order to achieve this, it is

necessary to compare the energy consumed in processing some data with the energy consumed in communicating with the competent unit. Furthermore, if there is multi-hop communication this cannot be just evaluated locally, but in terms of the overall system, as communication will involve energy consumption in several nodes of the network that might compromise the energy lifetime.

SICYPHOS uses the transaction approach presented in previous Section 5.2.4.4 to create network-wide energy profiles that can account the energy consumed all over the network in relation to an end-to-end communication.

These profiles are implemented as optional extensions to the WirelessTLM transaction. This extension contains a map of all the nodes that have taken part in the communication and their associated energy consumption value. There is an additional accumulated energy value which is the summary of all energy consumed. A diagram of the energy extension can be observed in Figure 5.13. Next chapter will explore examples on how to optimize the system using this energy profile.

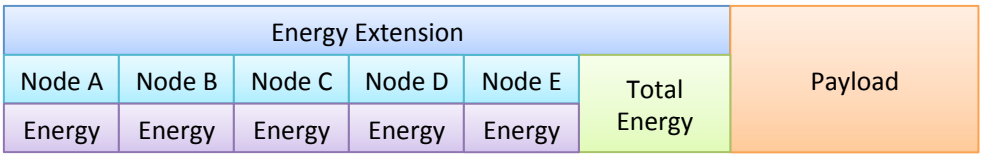

Figure 5.13: Energy extension, containing the total accumulated energy and the map of nodes and their consumed energy.

The so called `energy_extension` uses the same extension template described in Section 5.2.4.4, and therefore can be read and written using the same API.

5.3.2.4 Distributed Software Profiling: Distributed Activities

Finally, the energy aware framework provides methods to create profiles for high level distributed tasks, as described in Section 4.4.4.

In the same way as in software local profiles, the user can define distributed activities. The main difference is that in this case, a distributed activity will involve multiple nodes. Therefore, the implementation actuation is twofold:

1. Nodes must be flagged with the corresponding activity. For this purpose, a distributed activity attribute is added to energy aware modules, just as it is done for local activities.

2. Communication must also be flagged with the corresponding activity. This permits automatically propagating the distributed activity to any node taking any action on the transaction. This is done by including a distributed activity attribute in the energy extension described in Section 5.3.2.3.

With distributed activity labels added to both nodes and communications, the logger can build the energy profiles that account for the energy consumed in performing those distributed tasks.

5.3.3 The Energy Logger

Apart from providing the classes to create the power state machines and the means to create energy profiles, the energy aware framework must also provide a mechanism to log all this information. For this purpose, the framework includes a unified logger.

All power consumption related information is reported to a singleton class named `power_logger`. Being a singleton, there exist only one instance per simulation, which is in charge of processing and formatting all the reports and records them in an output file.

The advantage of this implementation is that every change in power consumption or its context that is modelled using the framework energy aware tools, can be automatically reported to the central power logger without any user action. The user can then extend the power logger function in order to provide the required outputs.

The energy logger is responsible for three main tasks: record every change in power states or activities, record transactions history on every transaction release and keep track of distributed profiles.

5.3.3.1 Logging State and Activity Changes

Every state and activity change is automatically reported to the central logger, which records the time stamp and registers all the required context variables to create the different kind of profiles.

For this purpose, the logger provides a method named `report()`, to register any energy relevant activity. The method takes to parameters: the simulation time stamp and the simulation module. The logger creates a power logging entry, which is a struct with the following attributes:

- **time:** Time stamp of the power logging entry.
- **module:** SystemC module that sent the report.
- **power:** The average power consumption value of the new power consumption state.
- **activity:** The current (local) activity or software profile, if any, of the module.
- **transaction ID:** The transaction associated to the change in power consumption, if any.
- **dist_activity:** The current distributed activity, if any.

Printing information in a file or in standard output (console) is very costly and can dominate the overall simulation performance if done for every state or activity change. To avoid this, the power log entries are buffered and printed whenever the buffer is full. This way the amount of input/output accesses are reduced.

5.3.3.2 Record Transactions History

Additionally, the energy logger offers the possibility to record transaction history. Whenever a transaction is finished, before deletion, the logger can print all the information contained in its extensions, which includes the air extension (see Section 5.2.4.4) and the energy extension (see Section 5.3.2.3).

This means that for every message generated in the network, the logger can log a full record containing which nodes contributed, successfully or unsuccessfully, in its delivery to the final destination, and how much energy was spent on it all over the network.

This functionality must not be triggered by the user, it is automatically done before a transaction is cleaned and returned back to the transactions pool.

5.3.3.3 Distributed Software Profiles

Finally the energy logger can also create and register distributed energy profiles. The user can add a distributed activity to reports and transactions. As all nodes and communications must go through the same single energy logger, it can build distributed activity profiles that contain:

- All power state changes all over the network related to this high-level distributed task.
- Full transaction history for all communications related to the high-level distributed task.

This profile enables estimating not only how much energy is spent all over the network in performing the distributed task, but also to separate the energy spent in processing and the energy spent in communication, to explore whether a more centralized or decentralized approach gives better results in terms of energy consumption and overall application lifetime.

6 Energy Profiling Performance and Evaluation

The techniques and their implementations proposed and described in previous sections open the door to new energy analysis for networked and distributed embedded systems. On the other hand, filling the semantic gap with energy profiles entails a cost in simulation performance. The cost and benefit of applying the proposed techniques must therefore be evaluated in order to wisely integrate them in the model-based design methodology.

This chapter aims to provide some insight about what the energy profiling techniques may offer the designer. Moreover, it aims to provide some notion about how this techniques may affect the simulation performance and how to deal with the different trade-offs that will necessarily challenge the designer.

The first point of this chapter is to show how the energy profiling mechanisms implemented in the framework can provide helpful information and enable energy optimization at high levels of abstraction. In other words, an introduction to how to use the framework and a demonstration that the energy profiling techniques and their implementation fulfil their purpose.

The second concern, not only for energy profiling, but for wireless sensor networks simulation in general, is simulation performance. Energy estimation is specially demanding, as discussed in Section 4.3. High-level abstraction has to be used carefully in order to avoid losing power consumption information. Furthermore, in open systems like sensor and networked nodes, the environment must be considered as a part of the model. For this purpose the performance of the simulation using the framework will be compared with a state-of-the-art simulation.

Last, but not least, the extent of the models and implementations is discussed, so that their validity and limitations can be comprehended. Although the validity of the overall modelling approach highly depends on the specific models adopted by the user, which are out of the scope of this work, it is necessary to evaluate the potential results that can be achieved using the proposed techniques.

6.1 Energy Profiling Test Scenarios

The first part of this analysis is to show the capabilities of energy profiling in high level energy optimization. This will be shown through two examples:

1. The first test scenario is a minimum example that shows how to rapidly setup an energy aware simulation of a wireless network using the proposed framework.
2. The second scenario consists of a multi-hop network where the energy of communication will be profiled to compare two different routing alternatives.

6.1.1 Setting Up a Simulation

This section explains how to set up a basic simulation scenario. The first part will focus on creating the wireless model, while the second part will describe how to add energy awareness to our model.

6.1.1.1 Wireless Simulation

The minimum wireless network simulation requires the following elements:

- The implementation of the two main abstract classes of the framework: the top-level module (`node_base`) and the wireless modules (`wtlm_module`).
- The XML file with the specification of the network topology and wireless communication parameters.
- A main file, which is the entry point of the program and must start the simulation.

Top Module

The basic module of the framework is the `node_base` implementation. This module contains the top-level API described in Section 5.2.5. Every node model must inherit from this `node_base` class and extend it with the application code and the internal communication structure.

The application code can be included by implementing the abstract methods already described in previous chapter, in order to interact with the node event queue: `initialize()`, where the first event can be added to the event queue, and `handle_event()`, which contains the actions to take on a specific event.

The internal communication structure must be configured in the constructor, where the TLM based protocol stack must be configured. The minimum protocol stack would require connecting the sockets for internal downwards and upwards communication from the physical layer directly to the `node_base` TLM sockets. Any additional protocol, such as MAC or routing protocols should be connected here, by binding the TLM sockets in the appropriate order.

Physical Layer

The minimum requirement to create a wireless network is to implement the physical layer. The framework provides an abstract class named `wtlm_module` that the user must implement with the specifics of the physical layer that will be used. The virtual methods to be implemented have already been explained in Section 5.2.1.1. Basically the user must provide the necessary application dependent information for the following questions:

- The calculation of the bit error rate, which depends on the modulation used.
- The acceptance of incoming transmissions, which depends on the operation mode of the transceiver, i.e. reception can only occur if the transceiver is in receive mode.

XML Network Specification

The network to be simulated must be specified in an XML file. This way, the number of nodes, their location, their type and the wireless communication parameters can be modified with no need to recompile the program.

The XML has two parts: the topology and the configuration. In the topology all nodes are enumerated with their name, type and spatial location. In the configuration part, some global parameters can be specified, such as the background noise, the minimum receiving power which governs the truncation algorithm for wireless broadcasting or the overall duration of the simulation.

Main File

The main file is the entry point of the program. To start a simulation the main file must perform the following steps:

1. Register in the node factory (see Section 5.2.6) the node creators for all the node types defined by the user.
2. Parse the XML network specification file which will instantiate and connect all nodes and configure the wireless communication parameters.
3. Start the SystemC based simulation, using `sc_start()`.

With these four elements a fully functional simulation of a wireless network can be created. Wireless propagation will automatically be calculated based on the transmitted power provided when sending a message and the spatial position of the different nodes.

6.1.1.2 Energy Aware Simulation

The energy aware framework is totally independent from the wireless framework. The minimum requirements to start using the capabilities of the energy aware framework are:

1. Any module required to be energy aware must inherit from `paf_interface`, which is an abstract class that contains all the infrastructure required for local (module-based) energy profiling, described in Section 5.3.2.
2. Register, using the method provided by the `paf_interface` any power state machine that describes the power consumption behavior of the subsystem in focus. These state machines can be created using the `power_state_machine` class provided by the framework.

With these two steps, any state transition will be automatically tracked and logged by the simulation. To enable high-level profiles, there are different ways, depending on the nature of this profiles:

- Software profiles (local or distributed) are tracked as long as an activity (local or distributed, respectively) is defined and active. Therefore the user must define the activities and just use the methods provided by the `paf_interface` to activate or deactivate them.
- Communication profiles are created as long as an energy extension is attached to the transaction in use. Therefore, the user must add this extension to any energy aware transaction before starting to use it.

All the output is logged by the central energy logger in comma-separated values format, which can be later analyzed in any external tool.

6.1.2 Routing Energy Efficiency

This example uses a network of overall 25 nodes:

- 24 ordinary nodes, that can generate and route messages
- one sink node that is the destination of all messages in the network and just receives the messages from the other nodes.

The topology of the network is shown in Figure 6.1. The dashed lines show which nodes are neighbour nodes and therefore might directly communicate with each other. The maximum number of neighbour nodes for that topology is 8 neighbours. The maximum distance to the sink node is two hops.

The network topology proposed in this example is regular and simple, so that the results can be followed and understood by a human observer, who can easily solve the different steps in the transaction lifetime and therefore understand transaction consumption results. However, the same approach applies to any randomized or more complex scenario.

6.1.3 Example of Transaction

In a first example, every node is entitled to forward the message, but as the maximum distance to destination is 2 hops, there is a time-to-live (TTL) parameter, initialize to '1', and decreased on every hop, that prevents the message from being forwarded twice.

Every ordinary node has a period of 2 seconds and a listening duty cycle of 25%, i.e. the receiver is switched on for $0,5$ seconds in every period.

At some point, node B sends a message. Figure 6.2 shows the listening duty cycles of the nodes in the neighbourhood, indicating, in red, which nodes actually receive the message.

As shown in Figure 6.2, nodes F and H receive the message from B. As all nodes also act as routers and it is the first hop, i.e. $TTL = 1$, both F and H forward the message further.

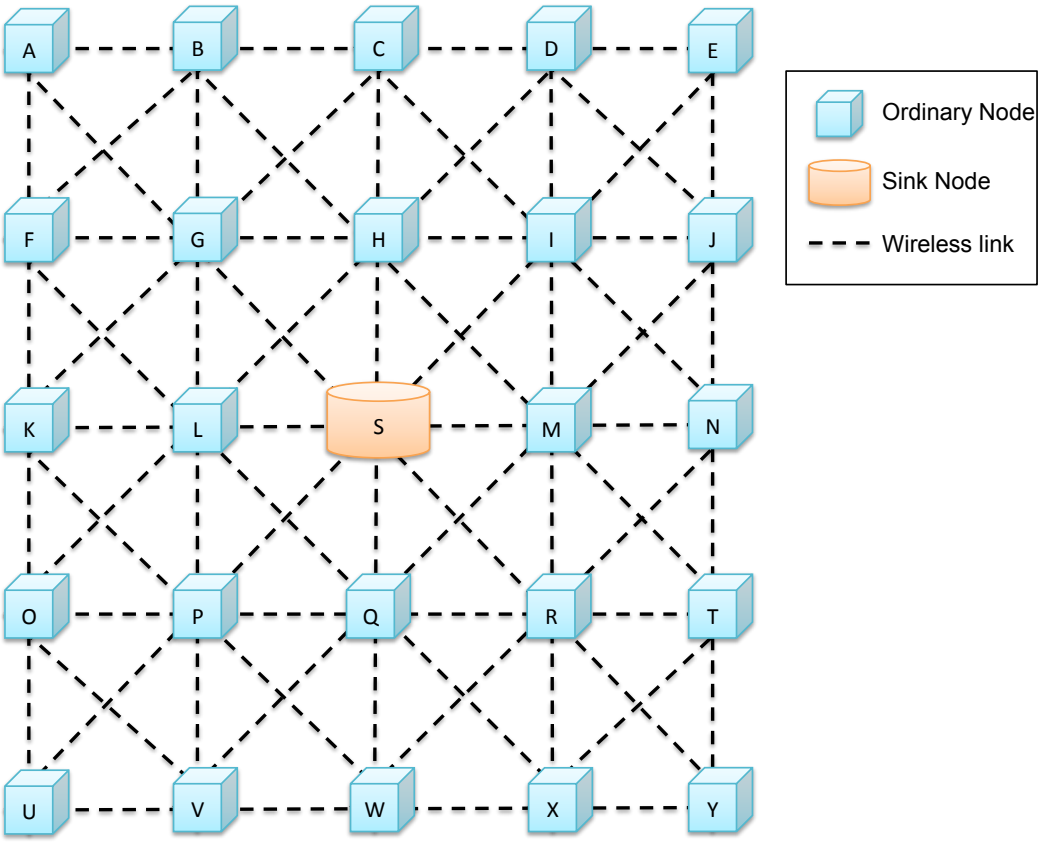

Figure 6.1: Network topology used in test scenario.

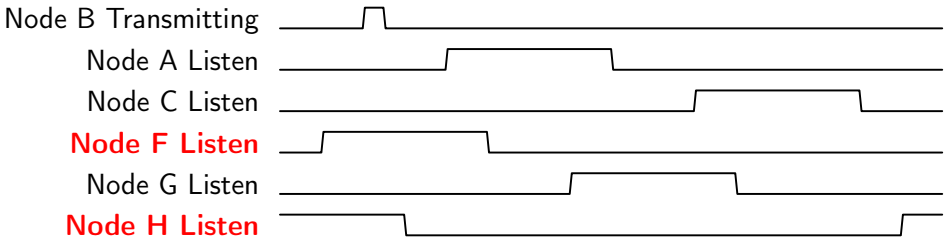

Figure 6.2: Listening duty cycles of nodes within node B communication range.

For node F, Figure 6.3 shows the listening duty cycles of nodes in its neighbourhood. Again, the nodes that actually receive the message are marked in red.

In this case, node K would be the second hop. Therefore, $TTL = 0$ and the message will not be forwarded further. In this case, node K discards the message.

For node H, Fiugre 6.4 depicts the listening duty cycles of nodes in its vicinity. Nodes able to receive the message are marked in red.

Node I is the same case as node K, it is the second hop and the message cannot be routed further, so it is discarded. On contrast, node S is the sink node and therefore the final destination of the message. Node S process the message and the communication process is finished.

The transaction that results from the described case is shown in Figure 6.5. The message is initially sent by node B, forwarded by nodes F and H, received and discarded by nodes I and K,

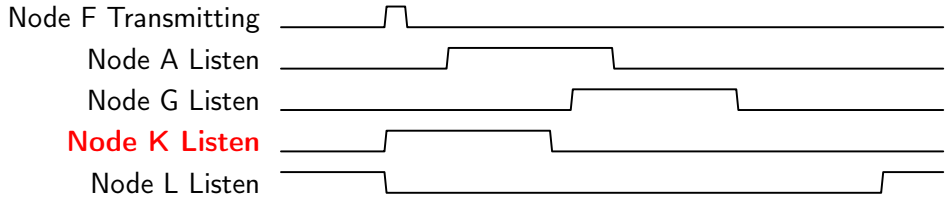

Figure 6.3: Listening duty cycles of nodes within node F communication range.

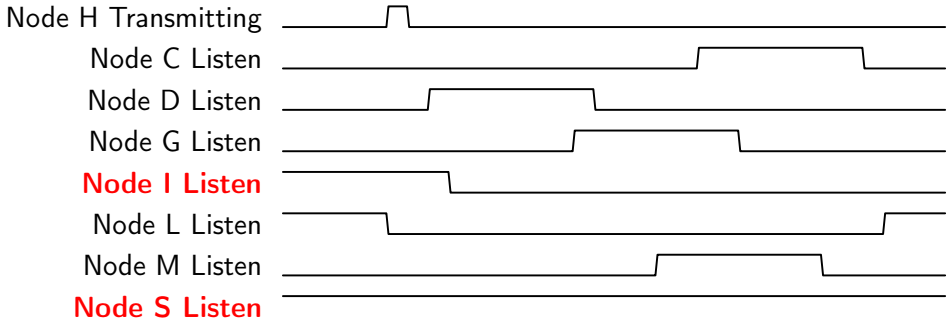

Figure 6.4: Listening duty cycles of nodes within node H communication range.

and received and finally processed by node S. All these operations have an energy cost, which is annotated in the transaction.

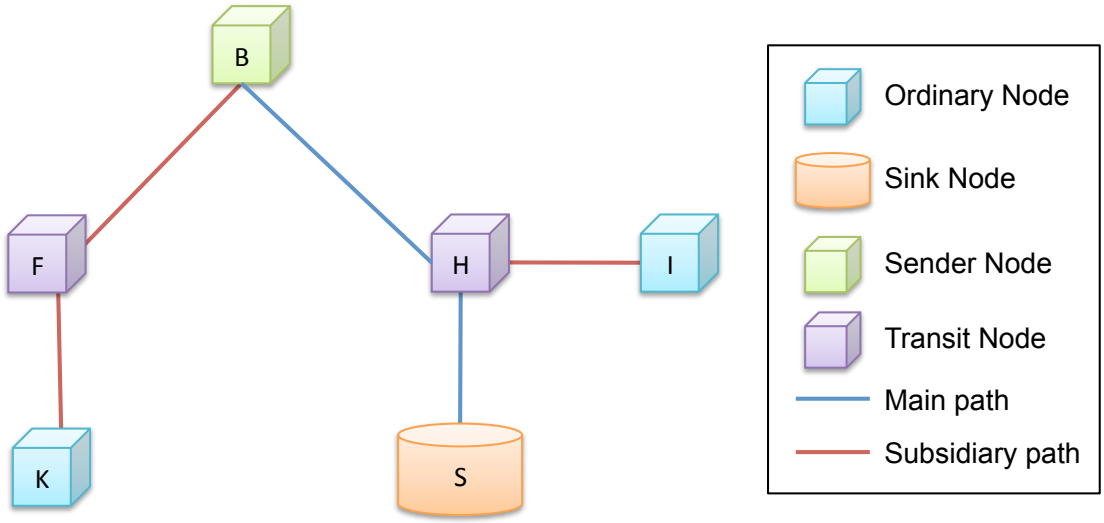

Figure 6.5: Topology of a transaction. All messages exchanged in the network with the same data content are shown.

Hence, the transaction keeps track of all communication operations all over the network that are related to the same end-to-end communication. It can be argued that the same result could have been achieved by using messages with a common identifier, but the transaction approach has clear advantages:

- As shown in Section 6.2.1, the transaction approach also improves simulation performance. The transaction optimizes the memory requirements by using a common piece of allocated

memory for all the information that remains invariable along the whole end-to-end communication process.

- Having separated messages with common identifiers reduces the possibilities of improving simulator logging intelligence and leaves all data mining to post-analysis of logged data. With the transaction it is very easy, for instance, to record the accumulated energy consumption data and log only those transactions whose energy consumption is over a specific threshold. Creating simulating logs is actually one of the biggest pitfalls for simulation performance, and therefore, providing the simulator with the tools to identify which information is more relevant and should be logged, is a significant boost to simulation performance.

6.1.4 The Transaction as an Energy Profile

The next step is to show how the transaction energy profiling approach can be leveraged to make high level design decisions. The transaction profiling permits estimating the cost of communication, i. e. how much energy was consumed in the network to transmit some specific information.

The network topology plays a crucial role in the energy cost of a transaction and the impact in the whole network. The next simulations evaluate the cost of transactions depending on the nodes which generated them. The first example is shown in Figure 6.6. The bars represent the simulated energy of transactions depending on the nodes that originally generated them.

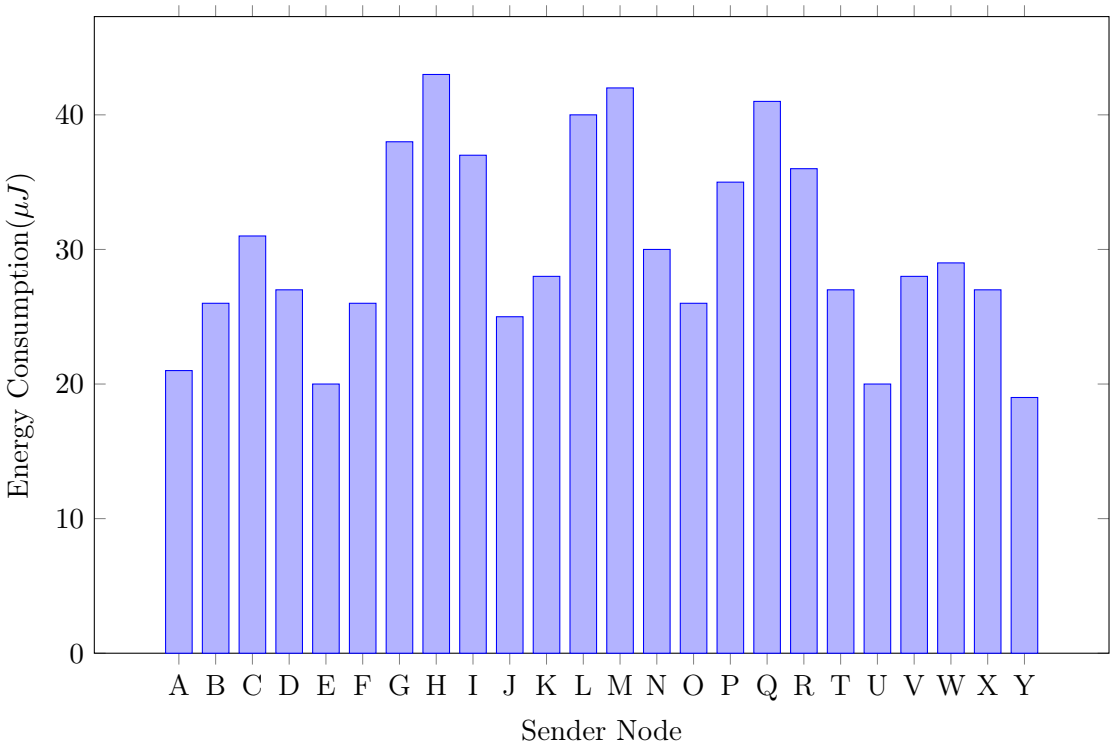

Figure 6.6: Transactions average energy cost depending on the nodes that generated them. Broadcast case.

In this first example, the routing scheme is broadcasting with a Time-To-Live (TTL) of one. In this case, most energy expensive transactions are those, whose sender node has a higher number

of neighbours. The more neighbour nodes, the more possibilities for the message to be forwarded further and cause more energy consumption. As $TTL = 1$ only direct neighbours are entitled to retransmit the message. Thus, the nodes whose transactions have less energy impact in the network are the nodes in the corners, which only have 3 direct neighbours. However, these transactions are more prone to fail, as the chances to get a transit node listening are reduced to these 3 neighbour nodes.

On the other hand, the nodes that generate the transactions with a higher energy impact on the network turn to be nodes H, L, M and Q. All of them have 8 neighbours, one of them is the sink node, but the other 7 are potential transit nodes that would forward the message. However, there are four more nodes with 8 neighbours (G, I, P and R) that have less energy impact. To understand the difference it is necessary to evaluate the second order neighbours. Node G and its symmetrically analogous nodes have 8 direct neighbours plus only 6 second order neighbours. However, node H and its symmetrically analogous nodes, have 8 direct neighbours plus 11 second order neighbours. This means that after forwarding, there are 5 potential receivers for messages from node H than from node G. If these nodes are listening they will spend energy receiving the message.

Now, in Figure 6.7 it can be observed a similar diagram for the same network but with preconfigured (ideal) routes. The graphic depicts two clear groups, which in the network correspond to two cases: nodes directly connected with the sink node and nodes at one node distance from the sink node.

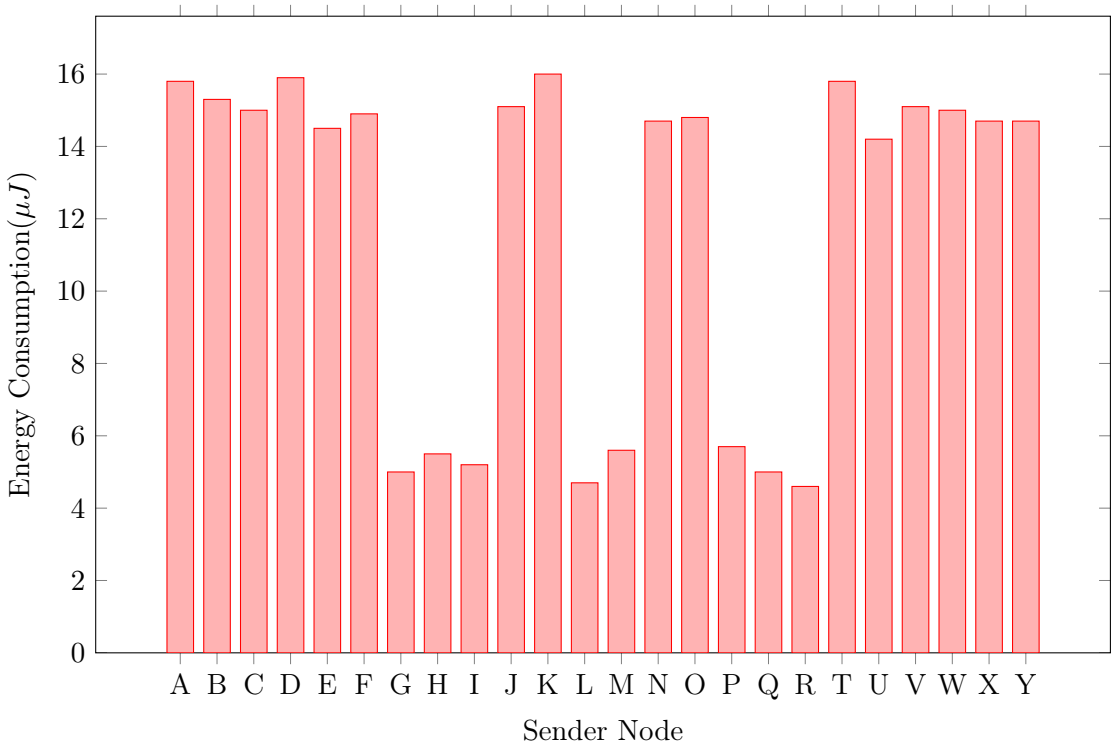

Figure 6.7: Transactions average energy cost depending on the nodes that generated them. Ideal routing case.

The number of neighbours has now a minor influence, as only the node selected as next hop in the preconfigured route is entitled to forward the message. Therefore, although neighbours will

still receive the message if they are listening, they will just discard it after checking that it is not for them.

Now, depending on the frequency of generating messages it is easy to establish a comparison between both approaches. Moreover, the energy margin for the routing algorithm overhead which is worthy can be calculated, and depending on how volatile routes are, a good decision can be made.

6.2 Simulation Performance Evaluation

The challenge in providing energy awareness through modelling and simulation is not only to achieve supplying the required data. A model can be extended and refined in order to reveal more and more details. However, the trade-off between accuracy and simulation performance is the main difficulty to overcome. The model not only has to provide the required information, but it must provide it within some simulation time boundaries.

The extensions to state-of-the-art simulators that enable a better estimation of energy consumption can be classified in two categories:

- **Energy profiles:** They fill the semantic gap in order to provide high-level abstraction without losing energy awareness. Keeping track of energy consumption implies a simulation overhead. To compensate this overhead, in this work, energy profiling has been implemented in combination with some high-level modelling approaches. The results of that combination will be evaluated next.
- **Environmental effects:** Cyber-Physical Systems cannot be evaluated without considering the environment, which is an active part of the system and determines its behaviour. However, adding physical processes to simulation is a very challenging task, that usually requires a combination of event-based simulation (typical for hardware-software simulation) and continuous simulation paradigms (for physical processes). The evaluation of this work shows therefore the performance of some hybrid models and how they scale.

6.2.1 Transaction Level Abstraction

The first evaluation consists in an example of wireless sensor network, which has been implemented using the PAWiS Framework, an OMNeT++ based WSN simulator, described in Section 3.2.1.2, and a similar implementation using the SICYPHOS framework and exploiting the transaction-level modelling approach.

The PAWiS implementation uses an ordinary message approach, where every point-to-point transmission is modelled separately and every message is cloned as many times as the number of potential receiving nodes.

Both frameworks provide a similar propagation model, based on attenuation and capable of modelling interferences and collisions. They also provide generic functionality to model wireless communications. However the users must define their own hardware and software models, their application, as well as user-defined protocol stacks and communication architecture. Both are based

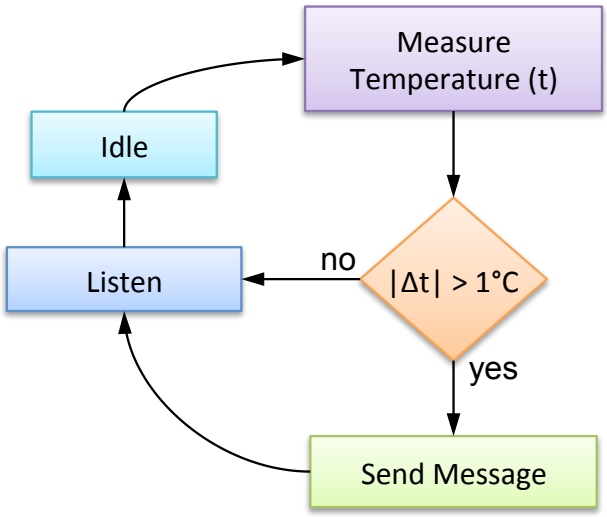

Figure 6.8: Flowchart of temperature measurement application.

on C++ and event-based simulation. While OMNeT++ focus is network engineering, SystemC focuses on hardware and software co-design.

The example consists in a temperature sensor application. The communication stack is integrated by a basic MAC layer to determine when the node is listening and messages can be received and a very simple routing layer, to be able to test how multi-hop communication performs.

The application implemented consists of nodes measuring temperature periodically. The flowchart for the temperature measurement application is shown in Figure 6.8. If the temperature variation is over a user-defined threshold, the new value is reported through the network.

Likewise, the flowchart for message reception is shown in Figure 6.9. The destination of temperature data is always the sink node. However, it may not be directly reachable. Therefore, ordinary nodes must work also as transition nodes. Every node in the network which is listening and within the range of the sender node, receives the message and checks errors and integrity. If the receiver is not the sink node, it forwards the message. To avoid infinite loops, a Time-To-Live value is attached to the message, which decreases on every hop. If this value reaches 0, the message is discarded.

Error calculation must be implemented out of the framework too, as it depends on modulation. In the example, errors are calculated for a 4-QAM modulation.

Simulation results, shown in Table 6.1, are intended to check the new TLM-based framework performance, in terms of simulation speed. This results are compared with those obtained using the PAWiS framework.

Table 6.1: Simulation Time Results

	PAWiS	SICYPHOS
Nodes: 3 - Time: 24h	2,462s	2,397s
Nodes: 15 - Time: 24h	47,783s	32,819s
Nodes: 25 - Time: 24h	3mins 3,6s	1min 2,27s

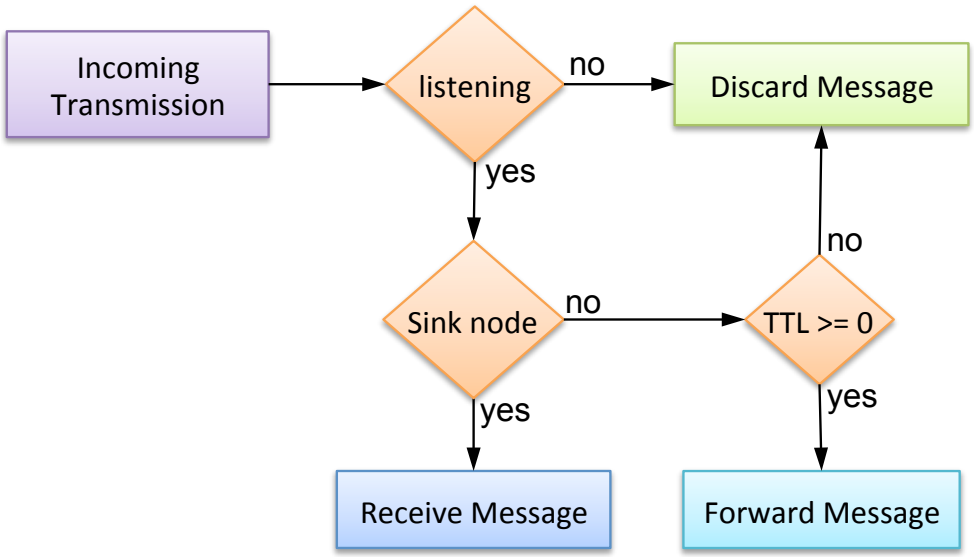

Figure 6.9: Flowchart of message reception application.

The table presents results for 24 hours of simulated time and different amount of nodes. In the first simulation the number of nodes to simulate is 3, which is an absolutely attainable value, so both frameworks require very little time. The SICYPHOS framework required 2,6% less time, which is almost negligible. In the second simulation, a network with 15 nodes is tested. In this case, the SICYPHOS framework improved PAWiS results by 31,32%, which makes a significant difference. With 25 nodes the improvement is even higher, with 49,7% less time for the SICYPHOS framework.

Performance improvements can be accounted to SystemC simulation kernel being more efficient than OMNeT++. However, the improvement increase with the number of nodes can only be accounted to the higher-level approach followed by the SICYPHOS framework, which has the following advantages:

- The SICYPHOS transaction uses the same data structure for all the messages with the same payload. Therefore, when a node forwards a message, it must just fill its specific headers and the particular conditions for every receiving node, while in OMNeT++ and other WSN simulators, the whole message must be copied with new memory space allocation for every potential receiving node.

- The SICYPHOS transactions are used through a memory manager, which maintains a pool of instantiated transactions and prevents from allocating and populating new memory space every time a message is generated.

In particular, in mobile ad-hoc networks simulation it is crucial to handle the complexity of huge networks. Simulators typically include some truncation threshold to avoid simulating the propagation to nodes where the effect is negligible. However, in dense networks, it is still easy to have to replicate a message too many times. In this scenarios, the transaction level approach gains special relevance, as the same payload is used and no data structure copying is required.

6.3 Multi-Domain Simulation

A fundamental step in the way to energy awareness of Cyber-Physical Systems is to integrate the environmental effect into simulations. CPS are very interactive and their usage of resources depends highly on the environmental conditions.

This evaluation aims to be a performance analysis to study the impact and ponder the expectations on performance of hybrid simulation using the framework. Comparisons with other simulators are intentionally avoided due to the infeasibility of defining an analogous simulation environment. There are very few simulators with a similar scope, and every simulation environment has a different focus and handles the trade-offs from a different perspective, making emphasis in different aspects and using models with different detail levels, which ultimately leads to unfair comparisons.

The performance analysis and evaluation is done using virtual scenarios based on a fridge model validated during the SmartCoDe Project [MDH+12].

6.3.1 SmartCoDe Project

SmartCoDe project aimed to build a Cyber-Physical Energy System, in which a distributed energy management application could control the energy demand and adapt it to the energy supplied by a wind turbine and to the real-time prices provided by the electric utility companies.

As part of the project, a SoC was designed, which included wireless communication based on ZigBee standard, power metering, power supply, and smart-card based security [MDG10].

The SmartCoDe application was developed using the SICYPHOS Framework, which enabled building a virtual prototype, on top of which, the application software could be developed while the hardware platform was unavailable. After manufacturing the application code was successfully migrated from the simulated prototype to the real PCB prototype.

Therefore, the SmartCode node implementation demonstrates the capability of the SICYPHOS framework in model-based design methodologies.

6.3.2 Evaluation Scenario

The evaluation scenario is an energy management application that controls the compressor in a number of fridges depending on an energy cost function. This energy cost function, is built based on the electricity market prise fluctuation, as well as the amount of renewable energy that is and will be available, estimated using weather forecasts.

The energy management application is built over a distributed system consisting of an energy management unit, controller nodes attached to every fridge and temperature sensor nodes, which must be installed inside the fridges. Communication is wireless using ZigBee communication protocol stack. According to the ZigBee terminology, we can distinguish the following device types:

- **ZigBee Coordinator (ZC):** It is the Energy Management Unit (EMU). Apart from being the network coordinator, it acts as a gateway to the external information such as weather forecasts and electricity market price.

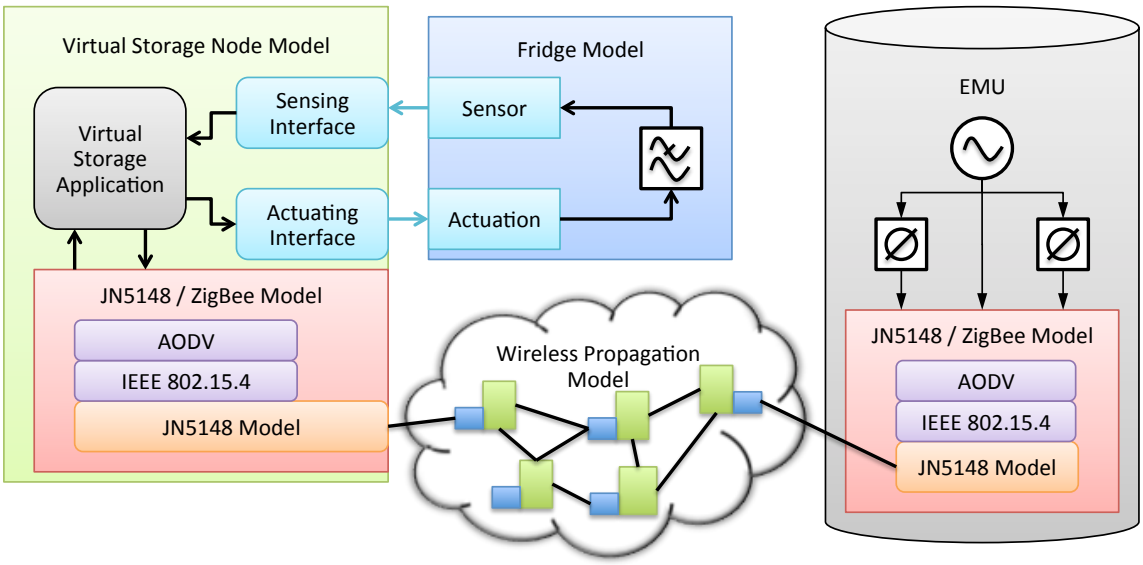

Figure 6.10: SmartCoDe application.

- **ZigBee Router (ZR):** They are the nodes attached to the appliances, and responsible for switching them on or off. They must be therefore connected to the power grid and they have available energy to act as routers.
- **ZigBee End Device (ZED):** The temperature sensor nodes are implemented as end devices. They have to be inside the fridges and they operate on batteries. Therefore, they must sleep most of the time and only wake up periodically to report the temperature inside the fridge.

Regarding the physical part, the simulation includes physical model for the temperature variation within the fridges. Temperature variation is modeled as an RC low-pass filter, which is an already validated approach [KR07] [EWP12]. This low-pass filter, depicted in Figure 6.10 can be modelled very easily as a SystemC-AMS module, using the Timed Data Flow (TDF) Model of Computation, which already provides the Laplace trasform tools to compute the low-pass filter transfer function, shown in Equation 6.1.

$$H(s) = \frac{1}{1 + RCs} \tag{6.1}$$

Table 6.2: Simulation performance for different temperature simulation time periods

Simulated Time	Execution Time
1h	1.024s
24h	22.949 s
1 week	158.518 s

Using this simple physical model, the energy management algorithm for the fridge could be designed, validated and verified using a simulated scenario, before the hardware platform was available and before the whole network was deployed.

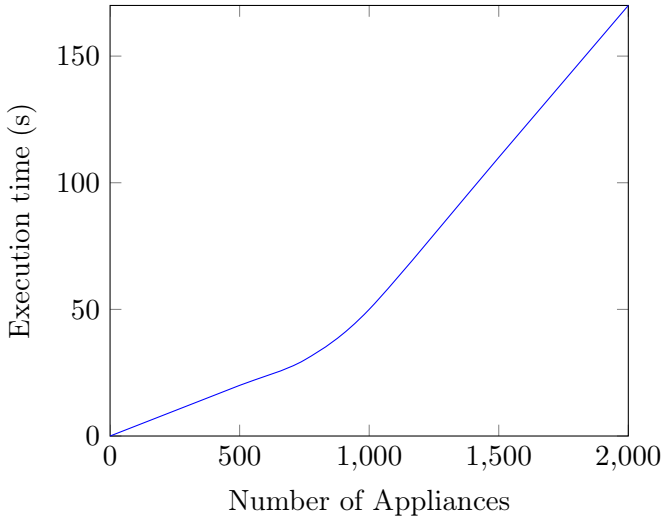

Figure 6.11: Simulation Performance for different number of appliances.

Figure 6.10 summarizes the whole SmartCoDe application model. Table 6.2 shows the execution time for a scenario using 4 fridges. Execution time grows almost linearly even for the very short simulation, which reveals that the overhead due to initialization of SystemC modules and TLM communication data structure has no significant impact in execution time. Figure 6.11 shows the execution time growth in relation to the number of simulated fridges (with no communication), where two different intervals can be identified with linear growth but different slopes. The main reason for the increasing slope starting at about a thousand nodes can be found memory. For instance, when the number of simultaneous transactions exceeds the maximum number of transactions in the memory manager pool, the generation of new transactions is more expensive in terms of performance.

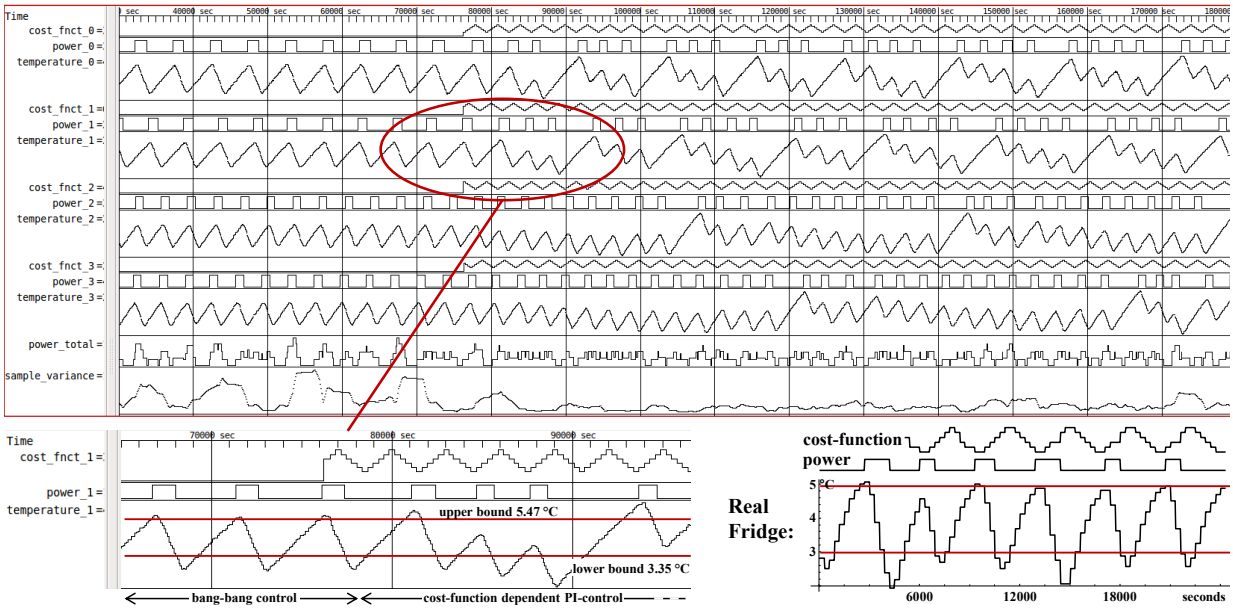

Figure 6.12: Simulation Screenshot, together with an example from a real fridge. [MDH+12]

After development in the simulated environment, the application was ported to a hardware plat-

form, specifically the JN5148 from Jennic/NXP and a real fridge was controlled in laboratory conditions. The output of the simulation together with the real fridge is shown in Figure 6.12.

7 Discussion and Outlook

Ultra-low power embedded systems, together with wireless communications have advanced greatly in the last decades, empowering the development of new technologies such as Wireless Sensor Networks (WSN) and new forms of Cyber-Physical Systems. More important than the low price of the devices is the inexpensiveness and simplicity of the deployment of a big amount of networked devices. These networks of devices are the foundation for all kind of distributed applications that can retrieve information and even interact with the environment of the devices at their location.

A primary challenge of this kind of systems is the energy supply. In order to keep the installation and maintenance of these systems easy and at a low cost, they must be autonomous devices with wireless communication and wireless power supply. Hence, batteries and energy harvesters have become typical power sources for these devices. The former has a limited energy budget, while the latter has a limited energy consumption flow. It is therefore crucial to optimize the energy consumption to maximize the energy lifetime or to obtain the required performance in spite of the energy restrictions.

Energy optimization affects all layers of abstraction of the system. A wireless sensor network, with hundreds or even thousands of nodes is a fault-tolerant system, with a high level of redundancy. The system will be operative even if many of its nodes run out of energy. Energy optimization must therefore consider not only the effects of energy consumption in a node, but also the global effects of energy consumption all over the network. Energy optimization measures may require a coordinated network strategy, i.e. a distributed energy optimization which maximizes the lifetime of the distributed application, even though some nodes might run out of energy far before the end of the overall system lifetime.

Most applications are designed to run unattended for very long periods of time. Ease of devices installation has led to scenarios where the location of the devices is not planned or even random. Furthermore, the system behaviour is greatly affected by the environmental conditions at the deployed location. It is therefore very complicated to validate and verify the system in laboratory conditions. Experiments would require recreating very specific conditions and very long operating time for evaluation. Model-Based Design methodologies enable the use of models to evaluate the long term operation of many very different scenarios with very divers node distribution and a variety of environmental conditions.

In particular, models are specially useful for energy consumption evaluation. They enable the exploration of energy optimization strategies and configurations, evaluating their impact in a hardware platform that is specifically being designed in parallel and is therefore not available.

Moreover they enable estimation of energy consumption metrics that are difficult, if not impossible, to obtain from measurements. Furthermore, they enable the evaluation of very different and long term scenarios, which are otherwise not possible.

This thesis has proposed some specific models for power consumption and some profiling techniques to create high level energy consumption information. The approach and implementations have been described in previous chapters, as well as the results of their application. This chapter will discuss the approach and the final results from a general perspective.

In addition, infeasibility of evaluation on the real systems also hinders the validation of the used models. It is therefore necessary to discuss the limitations of these models and how these limitations can affect the final system together with some strategies and trends to improve the models reliability and prevent critical failures due to an unrealistic or incorrect model.

Finally, the energy optimization problem will definitely be a matter of research in the next future. Energy efficiency improvements will lead to new applications that demand more complex computation that will continuously challenge the technologies and methodologies of system design. This chapter ends with some comments and thoughts that aim to provide vision and outlook about the future of the energy optimization topic.

7.1 Power and Energy Consumption Models

The State-of-the-Art analysis revealed that although there is general consent in the use of modelling to estimate power and energy consumption in electronic embedded systems, there is no standard approach to create those power and energy models. Additionally, there has been a chaotic proliferation of simulation platforms. Most of them are, in theory, flexible and extensible, but in practice they have been mostly volatile and ad hoc solutions, which have been hardly reused for further problems with different focus.

7.1.1 Hardware Models

Even for the omnipresent approach based on Finite-State-Machines to model the power consumption of digital systems, there is a lack of consistency and consensus. They are used in very different ways and, in most cases, formal definition is absolutely missing. This lack of formal definitions of Finite-State-Machines applied to energy estimation, is probably the cause for every approach using a different and specific definition.

Traditional Finite-State-Machine approach is based on digital systems and dynamic and static power components. At every state, only static power is consumed, while dynamic power is consumed during state transitions. The Finite-State-Machine proposed here has several advantages for high-level modelling.

- It does not require knowledge about how many transistors and voltage levels change in the system to estimate its energy consumption.
- It is based on the electrical characteristics that are public and disclosed in the comercial product datasheets.

- Associates the power consumption states with the system functional states so that the high level designer can easily trigger the state changes without knowing the low-level hardware implementation.

The Power-State-Machines proposed leverages from modern power management strategies where there is a high-level of control of active subsystems within a chip, through the use of clock and power gating, which specify different domains where clock signal or even supply voltage can be switched on and off.

However, there is one subsystem that has to be model in detail in order to obtain accurate estimations. Energy consumption of microcontrollers depends on how much time is required to execute the software tasks. This time depends on the microcontroller clock frequency, instruction set and even deeper implementation details, such as the cache-memory architecture and further execution optimization strategies such as branch prediction or out-of-order execution. For very low-power applications, the best performance per power in current state-of-the-art is being obtained by very simple microcontroller architectures, mostly RISC architectures with no kind of speculative execution. The use of at least an Instruction-Set Simulator (ISS) is therefore recommended in order to obtain accurate estimations.

7.1.2 Limitations of Computational Models

Along this work, different models have been developed and used for different projects, that have shown their convenience and helpfulness along the whole design process and in the energy optimization problem in particular.

However, the same reasons that motivate the use of models for the energy optimization problem, make them very complicated to be validated against real data. They can be calibrated and partially validated through some measurements and experiments that can be carried out in laboratory, but there are major difficulties to achieve a complete validation that remain unsolved:

- Unavailability of the embedded platform until manufacture. Models are used to enable hardware and software co-design, so that software development can start before the hardware platform is manufactured and available.
- Infeasibility of performing measures on the System-on-Chip (SoC). Even with a hardware platform available, it is not always possible to obtain the measurements that would be required to create an accurate model. Furthermore, SoCs will most likely contain Intellectual Property (IP) blocks where details about implementation are not available.
- Unpredictability of the operation conditions. Operation conditions are represented by variables that may range within different ranges. Although simulation permits multiple executions, finding the corner cases and worst case scenarios is typically another whole scientific challenge.
- Long term evaluation. Some long term effects will not be foreseen in the models used in the design phase. Technology changes very fast and there is no experience nor perspective to evaluate long term effects in the technology in use.

These hurdles are therefore inherent to the current technology requirements and limits the reliability of the models. To avoid the wrong models to jeopardize system design, it is crucial to continuously refine the models through model verification and validation, as long as possible, of all assumptions made when creating the concept of the model.

Furthermore, these limitations greatly encourage the reusability of the models. The lack of standardized cross-disciplinary modelling platforms and languages results in models that required huge effort to be created, verified, refined and validated and can later no longer be used in further projects because the application requires a different simulation approach. The use of already validated models could decidedly simplify and improve reliability of the Model-Based Design methodologies.

The FMI Standard has been a significant advancement in this direction, but further improvements in model reuse must still be made in the next future.

7.2 Conclusions on Energy Profiling

Apart from the hardware and software co-design. A virtual prototype permits estimating measurements that would require the real platform to be available. Furthermore, some measurements are infeasible even with the hardware platform available and deployed. In particular, measuring energy consumption has very hard constraints.

Distributed embedded systems are often integrated by very inexpensive devices, where measurement circuitry is not affordable, in terms of cost, size and even energy consumption.

Measurements in software require data processing and energy consumption. Moreover, finding out the subprocess that is responsible for every slice of energy consumed is costly and complex and is often not affordable by the very low cost and low power devices that integrate wireless sensor networks.

The presented virtual platform that recreates hardware, software, network and physical environment has shown that it is possible to obtain estimations of system performances in the final conditions. Furthermore, it enables the definition of more complex data structures, that are only possible in simulation, that account for energy consumption of high level components and permits a deeper and more complex analysis.

This work has exploited this possibility in order to obtain high-level information of energy consumption. The result of this research are the different types of energy profiles, described in Chapter 4.

These energy profiles have been successfully implemented in the SICYPHOS framework. Based on the simulation results, some conclusions can be extracted.

7.2.1 Hardware Profiles

Power consumption models are based on hardware models. For this reason, the necessity of defining hardware profiles might not be foreseen. However, network nodes are heterogeneous devices that are integrated by very different subsystems.

Moreover, current architecture optimization strategies tend to distribute the computation resources among several processing units or cores. We can see for instance very specific cores optimized and dedicated to execute certain tasks, such as the protocol stack. When their functionality is not needed they can be switched off, resulting in more efficient energy consumption than a more flexible processor running for longer time intervals.

Being able to group power models into higher level hardware structures is therefore very useful for architecture exploration.

In the implementation this is done by exploiting the hierarchy among the SystemC modules. Through the framework, when the user asks for power consumption of a module, it sums up the power consumption values of all its submodules.

7.2.2 Software Profiles

Software developers do typically know how to increase the efficiency of their code. However, they do not have information about how much energy is required to perform the programmed tasks.

The software profiles proposed here give the designer the flexibility to define its own software activities while the simulation framework accounts the energy consumption this activity can be liable for. Implementation in SICYPHOS is very simple and places responsibility in the software designer, who has freedom to define his own activities but must then explicitly switch on and off the activities.

The implemented activities have shown their effectiveness in showing that software tasks do have a greater impact in power consumption. Fine comparison is only possible when using accurate processor models, but they are actually required anyhow for accurate power estimation as well, in order to find out the execution times. If accurate models are not available, coarse-grained software profiles do also enable identifying which tasks are more power consuming and therefore should receive special attention from the designer.

However, the implemented activities have limited value for multi-tasking environments, where several simultaneous activities can be executed at the same time. At the current state, this is barely the case for ultra-low power sensor nodes, but this might change in the future and requires further research.

7.2.3 Communication Profiles

Hardware and software profiles can be found in literature in different forms and applications. They have been adapted to the specific application in distributed embedded systems.

However, for estimating the energy consumption of communication, no previous approach could be found in literature. Therefore, the communication profiling approach presented here is a novel approach and most likely the major contribution of this thesis to the high level energy optimization problem.

Implementation was made in combination with an abstraction of the communication model inspired in Transaction-Level Modelling for bus communication. The result is that the simulation does not just run based on the hardware nodes, but can also be seen from a communication perspective, where the end-to-end communication transactions are the simulation central structures.

Hence, new metrics can be used and shown after simulation, such as the energy consumption of a transaction all over the network. Transactions can therefore be optimized by configuring the routing algorithms and the topology in order to reduce hops, retransmissions and any other operation that might result in very inefficient end-to-end communications.

Moreover, comparisons between communication profiles and hardware and software profiles enable an energy efficiency analysis of decentralised and centralised approaches, which are a crucial design decision in distributed systems, that has to be made very early and has great impact in system energy consumption and performance. Purely centralised approaches require more communication to blindly deliver data to the central unit that will process it. Decentralised approaches use more intelligent (and more expensive and energy demanding) end devices that are able to process the data at its origin and make decisions that lead to less traffic and less communication effort.

From the energy point of view, decentralised approaches require less computation and processing energy consumption but more communication energy consumption. Profiling enables assessing the contribution of each part in order to optimize the energy consumption distribution to extend the energy lifetime of the distributed application.

7.3 Simulation Framework for Energy Optimization

Apart from the structures and requirements to model the power consumption of the hardware subsystems, it is crucial to consider the effects of communication and physical processes that are part of the feedback closed loop of Cyber-Physical Systems in order to optimize the overall energy consumption of the system.

In literature, some multi-domain simulators have been introduced, such as Ptolemy II and Modelica. This work proposed a multi-domain modelling approach based on SystemC simulation libraries, which is already widely used for virtual prototyping. This way, there is no need for coupling different simulators, but all hardware, software, communication and physical processes can be simulated in the same environment. There is also no need to learn a different language, as the approach is built on top of a hardware software co-design standard.

The idea of virtual prototyping is to start developing software before the hardware platform is available. However, in distributed embedded systems with cyber-physical interaction, the application development not only needs to consider the hardware where it will be executed. It also requires some knowledge about network topology and architecture and about the physical process that will be part of the system. The proposed platform enhances the current virtual prototyping concept by extending it with communication and physical environment models. All these extensions are made using the same virtual prototyping tools, with no need of co-simulation with external tools.

7.3.1 Energy optimization of hardware architecture

Energy optimization of the hardware architecture was already possible with the traditional virtual prototype approach. However, the simulation approach and implementation proposed here enhances its possibilities from the distribution point of view.

The simulation platform enables the exploration of how to distribute the computation resources in a distributed system. In distributed systems, there is a basic trade-off between communication and resources distribution.

- Simple systems with less intelligence and lower price and power consumption need to communicate more frequently to have the information processed.
- More intelligent systems can partially process the information and reduce communication, but will have higher cost and consume more power.

The exploration of different distribution options, together with the energy profiling tools, can help the designer to find out which computation resources distribution results in the most energy efficient overall system.

7.3.2 Energy optimization of network architecture

The virtual prototype of the whole distributed system enables comparing the energy consumption of different network architectures and configurations. The interfaces defined in SICYPHOS based on TLM interoperability, permit rapidly configuring and customizing the communication protocol stack and very easily exchange the algorithms on each layer without affecting the rest of the stack.

The flexibility and interoperability of TLM interfaces applied to network communication can be used together with the network wide energy profiling approach, which is also implemented in SICYPHOS, to estimate and compare the energy efficiency of different protocols and parameterizations. The protocols used definitely have an impact in system performance and in energy consumption. However, it is very complicated to account the energy cost of communication. Network energy profiles make this possible and easy to achieve.

Furthermore, all aspects must be combined in order to obtain the most effective results. For instance, depending on the resources distribution, different network architectures might be possible. Best decisions can only be made if all aspects are considered. The virtual prototype enables considering all these aspects at an early development stage.

7.3.3 Software energy optimization

The hardware virtual prototype permits developing the code of the software that will later run on the platform. However, with a virtual prototype of an isolated node, the software of a distributed system cannot be properly validated. Thus, although the virtual prototype might still help to ensure that the machine code will execute with no compilation or runtime errors, it is not possible to validate the network and physical interaction aspects of the application, which is actually the most critical part of many distributed embedded systems applications.

The SICYPHOS framework extends the hardware/software co-design approach by adding network and physical models that enable the validation and refinement of distributed software applications before the hardware is manufactured and installed and the distributed application is deployed.

This prototype also enables testing the software under different environmental conditions, in order to find corner-cases and weaknesses.

The framework adds software energy profiles in order to identify which software tasks are more critical for energy consumption. Using the prototype in combination with the profiles, the software application can be optimized for most efficient energy consumption. Furthermore, considering different effects that the network and the physical environment has on the application, several strategies or presets can be defined, to conform a basic energy management that could be implemented in simple devices that have no intelligence to create them themselves.

Literature

[Ad] Adam Dunkel . *Contiki Operating System*

[Acca] Accellera Systems Initiative. *SystemC*. Accellera Systems Initiative

[Accb] Accellera Systems Initiative. *SystemC-AMS*. SystemC-AMS Working Group

[Agr11] Agrawal, Dharma: Designing Wireless Sensor Networks: from theory to applications. In: *Central European Journal of Computer Science* 1 (2011), S. 2–18. – 10.2478/s13537-011-0007-z. – ISSN 1896–1533

[Ber] Berkeley, EECS Dept. U. *Ptolemy II*. UC Berkeley EECS Dept.

[BHS98] Benini, L. ; Hodgson, R. ; Siegel, P.: System-level power estimation and optimization. In: *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on*, 1998, S. 173–178

[BKL+04] Baldwin, Philip ; Kohli, Sanjeev ; Lee, Edward A. ; Liu, Xiaojun ; Zhao, Yang ; Ee, Contributors C. T. ; Brooks, Christopher ; Krishnan, N. V. ; Neuendorffer, Stephen ; Zhong, Charlie ; Zhou, Rachel: VisualSense: Visual modeling for wireless and sensor network systems / UCB ERL Memorandum UCB/ERL M04/8. 2004. – Forschungsbericht. Technical Memorandum

[BLT10] Brooks, Christopher X. ; Lee, Edward A. ; Tripakis, Stavros: Exploring models of computation with ptolemy II. In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA : ACM, 2010 (CODES/ISSS '10). – ISBN 978–1–60558–905–3, S. 331–332

[BOA+11] Blochwitz, Torsten ; Otter, M ; Arnold, M ; Bausch, C ; Clauss, C ; Elmqvist, H ; Junghanns, A ; Mauss, J ; Monteiro, M ; Neidhold, T [u. a.]: The functional mockup interface for tool independent exchange of simulation models. In: *Modelica'2011 Conference, March*, 2011, S. 20–22

[BROV11] Boano, Carlo A. ; Römer, Kay ; Österlind, Frederik ; Voigt, Thiemo: Demo Abstract: Realistic Simulation of Radio Interference in COOJA. In: *European Conference on Wireless Sensor Networks (EWSN 2011)*, 2011

[CBP+05] Chen, Gilbert ; Branch, Joel ; Pflug, Michael ; Zhu, Lijuan ; Szymanski, Boleslaw: SENSE: A Wireless Sensor Network Simulator. In: Szymanski, Boleslaw K. (Hrsg.) ; Yener, BÃijlent (Hrsg.): *Advances in Pervasive Computing and Networking*. Springer US, 2005. – 10.1007/0-387-23466-7_13. – ISBN 978–0–387–23466–3, S. 249–267

[Cen11] Center for Adaptive Wireless Systems. *AVRORAz: enabling ieee 802.15.4 compliant emulations*. 2011

[Chi07] Chipcon. *Chipcon CC2420 Datasheet. Texas Instruments*. 2007

[CLZ06] Cheong, Elaine ; Lee, Edward A. ; Zhao, Yang: Viptos: A Graphical Development and Simulation Environment for TinyOS-based Wireless Sensor Networks / EECS Department, University of California, Berkeley. 2006 (UCB/EECS-2006-15). – Forschungsbericht. Technical Report

[CMTG04] Conti, M. ; Maselli, G. ; Turi, G. ; Giordano, S.: Cross-layering in mobile ad hoc network design. In: *Computer* 37 (2004), Nr. 2, S. 48–51. – ISSN 0018–9162

[CS02] Chen, G. ; Szymanski, B.K.: COST: a component-oriented discrete event simulator. In: *Winter Simulation Conference* 1 (2002), S. 776–782. ISBN 0–7803–7614–5

[DLRJ00] Dick, Robert P. ; Lakshminarayana, Ganesh ; Raghunathan, Anand ; Jha, Niraj K.: Power analysis of embedded operating systems. In: *Proceedings of the 37th Annual Design Automation Conference*. New York, NY, USA : ACM, 2000 (DAC '00). – ISBN 1–58113–187–9, S. 312–315

[DLV12] Derler, P. ; Lee, E.A. ; Vincentelli, A.-S.: Modeling Cyber-Physical Systems. In: *Proceedings of the IEEE* 100 (2012), Nr. 1, S. 13–28. – ISSN 0018–9219

[DMHG10] Damm, M. ; Moreno, J. ; Haase, J. ; Grimm, C.: Using Transaction Level Modeling techniques for wireless sensor network simulation. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010. – ISSN 1530–1591, S. 1047 –1052

[DMN10a] Du, Wan ; Mieyeville, Fabien ; Navarro, David: IDEA1: A SystemC-based system-level simulator for wireless sensor networks. In: *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, 2010, S. 618 –622

[DMN10b] Du, Wan ; Mieyeville, Fabien ; Navarro, David: Modeling Energy Consumption of Wireless Sensor Networks by SystemC. In: *Systems and Networks Communication, International Conference on* 0 (2010), S. 94–98. ISBN 978–0–7695–4145–7

[DOTH07] Dunkels, Adam ; Osterlind, Fredrik ; Tsiftes, Nicolas ; He, Zhitao: Software-based on-line energy estimation for sensor nodes. In: *Proceedings of the 4th workshop on Embedded networked sensors*. New York, NY, USA : ACM, 2007 (EmNets '07). – ISBN 978–1–59593–694–3, S. 28–32

[DQ] Davide Quaglia, Francesco S. *SystemC Network Simulation Library*

[ELVAMS+06] Egea-Lopez, E. ; Vales-Alonso, J. ; Martinez-Sala, A. ; Pavon-Mario, P. ; Garcia-Haro, J.: Simulation scalability issues in wireless sensor networks. In: *Communications Magazine, IEEE* 44 (2006), Juli, Nr. 7, S. 64 – 73. – ISSN 0163–6804

[EOF+09] Eriksson, Joakim ; Osterlind, Fredrik ; Finne, Niclas ; Dunkels, Adam ; Tsiftes, Nicolas ; Voigt, Thiemo: Accurate Network-Scale Power Profiling for Sensor Network Simulators. In: Roedig, Utz (Hrsg.) ; Sreenan, Cormac (Hrsg.): *Wireless Sensor Networks* Bd. 5432. Springer Berlin / Heidelberg, 2009. – 10.1007/978-3-642-00224-3_20, S. 312–326

[EOH+09] Elmqvist, Hilding ; Otter, Martin ; Henriksson, Dan ; Thiele, Bernhard ; Mattsson, Sven E. ; Systèmes, Dassault ; Lund, Sweden D.: Modelica for embedded systems. In: *Proc. of the 7-th International Modelica Conference*, 2009, S. 354–363

[Ern98] Ernst, R.: Codesign of embedded systems: status and trends. In: *Design Test of Computers, IEEE* 15 (1998), Nr. 2, S. 45–54. – ISSN 0740–7475

[EWP12] Elsheikh, A ; Widl, E. ; Palensky, P.: Simulating complex energy systems with Modelica: A primary evaluation. In: *Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on*, 2012. – ISSN 2150–4938, S. 1–6

[FDLS08] Fonseca, Rodrigo ; Dutta, Prabal ; Levis, Philip ; Stoica, Ion: Quanto: tracking energy in networked embedded systems. In: *Proceedings of the 8th USENIX conference on Operating systems design and implementation.* Berkeley, CA, USA : USENIX Association, 2008 (OSDI'08), S. 323–338

[FGSS98] Fornaciari, William ; Gubian, Paolo ; Sciuto, Donatella ; Silvano, Cristina: Power estimation of embedded systems: a hardware/software codesign approach. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 6 (1998), Nr. 2, S. 266–275

[FQS08] Fummi, F. ; Quaglia, D. ; Stefanni, F.: A SystemC-based framework for modeling and simulation of networked embedded systems. In: *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, 2008, S. 49–54

[GEC+04] Girod, Lewis ; Elson, Jeremy ; Cerpa, Alberto ; Stathopoulos, Thanos ; Ramanathan, Nithya ; Estrin, Deborah: EmStar: a software environment for developing and deploying wireless sensor networks. In: *Proceedings of the annual conference on USENIX Annual Technical Conference.* Berkeley, CA, USA : USENIX Association, 2004 (ATEC '04), S. 24–24

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-oriented Software.* 1. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0–201–63361–2

[GKM82] Graham, Susan L. ; Kessler, Peter B. ; Mckusick, Marshall K.: Gprof: A call graph execution profiler. In: *Proceedings of the 1982 SIGPLAN symposium on Compiler construction.* New York, NY, USA : ACM, 1982 (SIGPLAN '82). – ISBN 0–89791–074–5, S. 120–126

[GLB+03] Gay, David ; Levis, Philip ; von Behren, Robert ; Welsh, Matt ; Brewer, Eric ; Culler, David: The nesC language: A holistic approach to networked embedded systems. In: *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. New York, NY, USA : ACM, 2003 (PLDI '03). – ISBN 1–58113–662–5, S. 1–11

[GNMO12] Galos, Mihai ; Navarro, David ; Mieyeville, Fabien ; O'Connor, Ian: A cycle-accurate transaction-level modelled energy simulation approach for heterogeneous Wireless Sensor Networks. In: *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International* IEEE, 2012, S. 209–212

[GRE+07] Girod, Lewis ; Ramanathan, Nithya ; Elson, Jeremy ; Stathopoulos, Thanos ; Lukac, Martin ; Estrin, Deborah: EmStar: A software environment for developing and deploying heterogeneous sensor-actuator networks. In: *ACM Trans. Sen. Netw.* 3 (2007), August. – ISSN 1550–4859

[GSR+04] Girod, Lewis ; Stathopoulos, Thanos ; Ramanathan, Nithya ; Osterweil, Eric ; Schoellhammer, Tom ; Kapur, R. *EmTOS: A Development Tool for Heterogeneous Sensor Networks*. 2004

[GZD+00] Gajski, Daniel D. ; Zhu, Jianwen ; Domer, Rainer ; Gerstlauer, Andreas ; Zhao, Shuqing: *SpecC: Specification Language and Methodology*. 1. Springer, März 2000. – ISBN 0792378229

[HMD11] Haase, J ; Moreno, J ; Dietrich, D: Power-Aware System Design of Wireless Sensor Networks: Power Estimation and Power Profiling Strategies. In: *Industrial Informatics, IEEE Transactions on* (2011), Nr. 99, S. 1–1

[HRFR06] Henderson, Thomas R. ; Roy, Sumit ; Floyd, Sally ; Riley, George F.: ns-3 project goals. In: *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. New York, NY, USA : ACM, 2006 (WNS2 '06). – ISBN 1–59593–508–8

[HSW+00] Hill, Jason ; Szewczyk, Robert ; Woo, Alec ; Hollar, Seth ; Culler, David ; Pister, Kristofer: System architecture directions for networked sensors. In: *SIGPLAN Not.* 35 (2000), November, S. 93–104. – ISSN 0362–1340

[IEE06] IEEE standard 802.15.4-2006: *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*. . Piscataway, USA: IEEE Standards Association, September 2006

[IEE12] IEEE standard 1666-2011: *IEEE Standard for Standard SystemC Language Reference Manual* . 2. Piscataway, USA: IEEE Standards Association, Januar 2012

[IFF96] Ierusalimschy, Roberto ; de Figueiredo, Luiz H. ; Filho, Waldemar C.: Lua: an extensible extension language. In: *Softw. Pract. Exper.* 26 (1996), June, S. 635–652. – ISSN 0038–0644

[Ini13] Initiative, Accellera S.: *Standard SystemC AMS extensions 2.0 Language Reference Manual*. . : Accellera Systems Initiative, März 2013

[Ins] Institute, Information S. *The Network Simulator - ns-2*

[JMF+96] Jones, Michael B. ; McCulley, Daniel L. ; Forin, Alessandro ; Leach, Paul J. ; Roşu, Daniela ; Roberts, Daniel L.: An overview of the Rialto real-time architecture. In: *Proceedings of the 7th workshop on ACM SIGOPS European workshop: Systems support for worldwide applications*. New York, NY, USA : ACM, 1996 (EW 7), S. 249–256

[Joh99] Johnson, David B.: Validation of Wireless and Mobile Network Models and Simulation. In: *In Proceedings of the DARPA/NIST Network Simulation Validation Workshop*, 1999

[KAB+03] Kim, Nam S. ; Austin, Todd ; Baauw, D ; Mudge, Trevor ; Flautner, Krisztián ; Hu, Jie S. ; Irwin, Mary J. ; Kandemir, Mahmut ; Narayanan, Vijaykrishnan: Leakage current: Moore's law meets static power. In: *Computer* 36 (2003), Nr. 12, S. 68–75

[Kal09] Kalicinski, Marcin: *RAPIDXML Manual*. 1.13. : , 2009. – Version 1.13

[Kes88] Keshav, Srinivasan: REAL: A Network Simulator / University of California at Berkeley. 1988. – Forschungsbericht.

[Kop11] Kopetz, Hermann: Internet of Things. In: *Real-Time Systems*. Springer US, 2011 (Real-Time Systems Series). – ISBN 978–1–4419–8236–0, S. 307–323

[KR07] Kupzog, F. ; Roesener, C.: A closer Look on Load Management. In: *Industrial Informatics, 2007 5th IEEE International Conference on* Bd. 2, 2007. – ISSN 1935–4576, S. 1151–1156

[Lee08a] Lee, E.A.: Cyber Physical Systems: Design Challenges. In: *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 2008, S. 363 –369

[Lee08b] Lee, E.A.: Cyber Physical Systems: Design Challenges. In: *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 2008, S. 363–369

[Lee09] Lee, Edward A.: Finite State Machines and Modal Models in Ptolemy II / EECS Department, University of California, Berkeley. 2009 (UCB/EECS-2009-151). – Forschungsbericht.

[LLWC03] Levis, Philip ; Lee, Nelson ; Welsh, Matt ; Culler, David: TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: *Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA : ACM, 2003 (SenSys '03). – ISBN 1–58113–707–9, S. 126–137

[LN01] Liu, Jason ; Nicol, David M. *DaSSF 3.1 User's Manual*. `http://users.cis.fiu.edu/~liux/research/projects/dassf/papers/dassf-manual-3.1.ps`. August 2001

[LPN+01] Liu, Jason ; Perrone, L. F. ; Nicol, David M. ; Liljenstam, Michael ; Elliott, Chip ; Pearson, David: Simulation Modeling of Large-Scale Ad-hoc Sensor Networks. In: *European Simulation Interoperability Workshop*, 2001

[LS11] Lee, E.A. ; Seshia, S.A.: *Introduction to Embedded Systems: A Cyber-physical Systems Approach.* . Lulu.com, 2011. – ISBN 9780557708574

[LW04] Landsiedel, Olaf ; Wehrle, Klaus: AEON: Accurate Prediction of Power Consumption in Sensor Networks. In: *In Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II*, 2004

[LWTP05] Landsiedel, Olaf ; Wehrle, Klaus ; Titzer, Ben L. ; Palsberg, Jens: Enabling detailed modeling and analysis of sensor networks. In: *Praxis der Informationsverarbeitung und Kommunikation* 28 (2005), Nr. 2, S. 101–106

[MB04] Mahlknecht, S. ; Bock, M.: CSMA-MPS: a minimum preamble sampling MAC protocol for low power wireless sensor networks. In: *Factory Communication Systems, 2004. Proceedings. 2004 IEEE International Workshop on*, 2004, S. 73 – 80

[MDG10] Mahlknecht, S. ; Damm, M. ; Grimm, C.: A Smartcard based approach for a secure energy management node architecture. In: *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, 2010, S. 769–773

[MDH+12] Moreno, Javier ; Damm, Markus ; Haase, Jan ; Grimm, Christoph ; Holleis, Edgar: Unified and comprehensive electronic system level, network and physics simulation for wirelessly networked cyber physical systems. In: *Specification and Design Languages (FDL), 2012 Forum on* IEEE, 2012, S. 68–74

[MEO98] Mattsson, Sven E. ; Elmqvist, Hilding ; Otter, Martin: Physical system modeling with Modelica. In: *Control Engineering Practice* 6 (1998), Nr. 4, S. 501–510

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

[MHSM10] Möstl, Georg ; Hagelauer, Richard ; Springer, Andreas ; Müller, Gerhard: Accurate power-aware simulation of wireless sensor networks considering real-life application code. In: *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems.* New York, NY, USA : ACM, 2010 (MSWIM '10). – ISBN 978–1–4503–0274–6, S. 31–38

[Mod13] Modelica Association Project: Functional Mock-up Interface for Model Exchange and Co-Simulation. In: *Functional Mock-up Interface Standard* (2013)

[MPGD13] Molina, J.M. ; Pan, Xiao ; Grimm, C. ; Damm, M.: A framework for model-based design of embedded systems for energy management. In: *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*, 2013, S. 1–6

[NC11] Nygren, Johannes ; Carlsson, Bengt: Benchmark simulation model no. 1 with a wireless sensor network for monitoring and control. In: *Uppsala University* (2011)

[NG03] Naoumov, Valeri ; Gross, Thomas: Simulation of large ad hoc networks. In: *Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*. New York, NY, USA : ACM, 2003 (MSWIM '03). – ISBN 1–58113–766–4, S. 50–57

[Ö06] Österlind, Fredrik. *A Ray-Tracing Based Radio Medium in COOJA*. Dezember 2006

[ODE+06] Osterlind, F. ; Dunkels, A. ; Eriksson, J. ; Finne, N. ; Voigt, T.: Cross-Level Sensor Network Simulation with COOJA. In: *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006. – ISSN 0742–1303, S. 641 –648

[OED10] Österlind, Fredrik ; Eriksson, Joakim ; Dunkels, Adam: Cooja TimeLine: a power visualizer for sensor network simulation. In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. New York, NY, USA : ACM, 2010 (SenSys '10). – ISBN 978–1–4503–0344–6, S. 385–386

[PAP08] de Paz Alberola, Rodolfo ; Pesch, Dirk: AvroraZ: extending Avrora with an IEEE 802.15.4 compliant radio chip model. In: *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. New York, NY, USA : ACM, 2008 (PM2HW2N '08). – ISBN 978–1–60558–239–9, S. 43–50

[PBM+04] Polley, J. ; Blazakis, D. ; McGee, J. ; Rusk, D. ; Baras, J.S.: ATEMU: a fine-grained sensor network simulator. In: *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, 2004, S. 145 – 152

[PCC+08] Perla, Enrico ; Catháin, Art ¦ Carbajo, Ricardo S. ; Huggard, Meriel ; Mc Goldrick, Ciarán: PowerTOSSIM z: realistic energy modelling for wireless sensor network environments. In: *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. New York, NY, USA : ACM, 2008 (PM2HW2N '08). – ISBN 978–1–60558–239–9, S. 35–42

[PCN] of Pervasive Computing, Center ; Networking. *SENSE: Sensor Network Simulator and Emulator*

[PN02] Perrone, L.F. ; Nicol, D.M.: A scalable simulator for TinyOS applications. In: *Simulation Conference, 2002. Proceedings of the Winter* Bd. 1, 2002, S. 679 – 687 vol.1

[PR02] Kap. 1 In: Pedram, M. ; Rabaey, J.M.: *Power aware design methodologies*. Kluwer Academic, 2002, S. 2–3. – ISBN 9781402071522

[PSS00] Park, Sung ; Savvides, Andreas ; Srivastava, Mani B.: SensorSim: a simulation framework for sensor networks. In: *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*. New York, NY, USA : ACM, 2000 (MSWIM '00). – ISBN 1–58113–304–9, S. 104–111

[PVS+08] Prabhakar, T.V. ; Venkatesh, S. ; Sujay, M.S. ; Kuri, J. ; Praveen, K.: Simulation blocks for TOSSIM-T2. In: *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, 2008, S. 17 –23

[RHWG95] Rosenblum, M. ; Herrod, S.A. ; Witchel, E. ; Gupta, A.: Complete computer system simulation: the SimOS approach. In: *Parallel Distributed Technology: Systems Applications, IEEE* 3 (1995), Nr. 4, S. 34 –43. – ISSN 1063–6552

[RSZ04] Raghavendra, C.S. ; Sivalingam, K.M. ; Znati, T.: *Wireless Sensor Networks.* . Springer, 2004 (Ercoftac Series). – ISBN 9781402078835

[Sav98] Savage, John E.: *Models of computation.* Bd. 136. . Addison-Wesley Reading, MA, 1998

[SHC+04] Shnayder, Victor ; Hempstead, Mark ; Chen, Bor-rong ; Allen, Geoff W. ; Welsh, Matt: Simulating the power consumption of large-scale sensor network applications. In: *Proceedings of the 2nd international conference on Embedded networked sensor systems.* New York, NY, USA : ACM, 2004 (SenSys '04). – ISBN 1–58113–879–2, S. 188–200

[SHK+06] Sobeih, A. ; Hou, J.C. ; Kung, Lu-Chuan ; Li, Ning ; Zhang, Honghai ; Chen, Wei-Peng ; Tyan, Hung-Ying ; Lim, Hyuk: J-Sim: a simulation and emulation environment for wireless sensor networks. In: *Wireless Communications, IEEE* 13 (2006), August, Nr. 4, S. 104 –119. – ISSN 1536–1284

[SIS] for Software Integrated Systems, Institute. *JProwler*

[SRMB98] Schulz, S. ; Rozenblit, Jerzy W. ; Mrva, M. ; Buchenriede, K.: Model-based codesign. In: *Computer* 31 (1998), Nr. 8, S. 60–67. – ISSN 0018–9162

[SVML03] Simon, G. ; Volgyesi, P. ; Maroti, M. ; Ledeczi, A.: Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In: *Aerospace Conference, 2003. Proceedings. 2003 IEEE* Bd. 3, 2003, S. 1339–1346

[Teca] Technologies, Scalable N. *Qualnet*

[Tecb] Technology, Riverbed. *OPNET*

[TLP05] Titzer, Ben L. ; Lee, Daniel K. ; Palsberg, Jens: Avrora: scalable sensor network simulation with precise timing. In: *Proceedings of the 4th international symposium on Information processing in sensor networks.* Piscataway, NJ, USA : IEEE Press, 2005 (IPSN '05). – ISBN 0–7803–9202–7

[TMW94] Tiwari, V. ; Malik, S. ; Wolfe, A.: Power analysis of embedded software: a first step towards software power minimization. In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 2 (1994), Dezember, Nr. 4, S. 437 –445. – ISSN 1063–8210

[UCL11] UCLA Compilers Group. *AVRORA: The AVR Simulation and Analysis Framework.* 2011

[VGE03] Vachoux, A. ; Grimm, C. ; Einwich, K.: Analog and mixed signal modelling with SystemC-AMS. In: *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on* Bd. 3, 2003, S. III–914–III–917 vol.3

[VNPJ96] Valderrama, C.A. ; Nacabal, F. ; Paulin, P. ; Jerraya, A.A.: Automatic generation of interfaces for distributed C-VHDL cosimulation of embedded systems: an industrial experience. In: *Rapid System Prototyping, 1996. Proceedings., Seventh IEEE International Workshop on*, 1996, S. 72–77

[WB13] Wang, Baobing ; Baras, J.S.: HybridSim: A Modeling and Co-simulation Toolchain for Cyber-physical Systems. In: *Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on*, 2013. – ISSN 1550–6525, S. 33–40

[WNP11] Wu, He ; Nabar, Sidharth ; Poovendran, Radha: An Energy Framework for the Network Simulator 3 (ns-3). In: *Proceedings of SIMUTools 2011*, 2011

[WPW00] Wu, Qing ; Pedram, M. ; Wu, Xunwei: Clock-gating and its application to low power design of sequential circuits. In: *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on* 47 (2000), März, Nr. 3, S. 415 –420. – ISSN 1057–7122

[WR96] Witchel, Emmett ; Rosenblum, Mendel: Embra: fast and flexible machine simulation. In: *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems.* New York, NY, USA : ACM, 1996 (SIGMETRICS '96). – ISBN 0–89791–793–6, S. 68–79

[YHE02] Ye, Wei ; Heidemann, J. ; Estrin, D.: An energy-efficient MAC protocol for wireless sensor networks. In: *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* Bd. 3, 2002. – ISSN 0743–166X, S. 1567 – 1576 vol.3

[ZBG98] Zeng, Xiang ; Bagrodia, Rajive ; Gerla, Mario: GloMoSim: a library for parallel simulation of large-scale wireless networks. In: *Proceedings of the twelfth workshop on Parallel and distributed simulation.* Washington, DC, USA : IEEE Computer Society, 1998 (PADS '98). – ISBN 0–8186–8457–7, S. 154–161

Javier Moreno Molina

Curriculum Vitae

Peter-Bardens-Str. 9
67661 Kaiserslautern
+49 (0) 176 565 94137
javiermorenomolina@gmail.com
Madrid, 10th July, 1984

Education

2002-2007 **Master of Science in Telecommunication Engineering**, *Madrid Technical University*, 2 Semesters + Master Thesis in Vienna University of Technology.

Experience

2008-2012 **Project Assistant**, *Vienna University of Technology*, Vienna, Institute for Computer Technology.

Projects

PAWiS:, *Implementation of novel MAC and routing protocols for wireless sensor networks within a framework, developed with OMNeT++, to simulate WSNs.*

SNOPS:, *SystemC and Transaction Level Modelling (TLM) based framework development for simulating hardware, network and system level of wireless sensor networks..*

SmartCoDe:, *Virtual prototype of smart metering systems capable to sense, actuate and wireless communicate among them and with an energy management system. Implementation of the embedded application and ZigBee custom profile..*

2012-2015 **University Assistant**, *University of Kaiserslautern*, Kaiserslautern, Workgroup on Design of Cyber-Physical Systems.

Master Thesis

title *Routing Algorithms for Air Traffic Services Communication Networks*

supervisors O. Univ. Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich and Univ.Ass. Dipl.-Ing. Thomas Rausch

grade 1 (Excellent)

description Adaptation of routing algorithms from both packet and circuit switching networks to a circuit switching network based on Air Traffic Services communication networks. Simulation and performance evaluation of the algorithms adapted. Combination of the most advantageous features into a single routing algorithm.

Selected Publications

J. Moreno Molina, M. Damm, J. Haase, E. Holleis, and C. Grimm, "Model based design of distributed embedded cyber physical systems," in *Models, Methods, and Tools for Complex Chip Design*. Springer, 2014, pp. 127–143.

J. Moreno Molina, X. Pan, C. Grimm, and M. Damm, "A framework for model-based design of embedded systems for energy management," in *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*. IEEE, 2013, pp. 1–6.

J. Moreno Molina, M. Damm, J. Haase, C. Grimm, and E. Holleis, "Unified and comprehensive electronic system level, network and physics simulation for wirelessly networked cyber physical systems," in *Specification and Design Languages (FDL), 2012 Forum on*. IEEE, 2012, pp. 68–74.

J. Moreno Molina, J. Wenninger, J. Haase, and C. Grimm, "Energy profiling technique for network-level energy optimization," in *AFRICON, 2011*. IEEE, 2011, pp. 1–6.

J. Moreno Molina, J. Haase, and C. Grimm, "Energy consumption estimation and profiling in wireless sensor networks," in *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*. VDE, 2010, pp. 1–6.

J. Haase, J. M. Molina, and D. Dietrich, "Power-aware system design of wireless sensor networks: Power estimation and power profiling strategies," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 601–613, 2011.

M. Damm, J. Moreno Molina, J. Haase, and C. Grimm, "Using transaction level modeling techniques for wireless sensor network simulation," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 1047–1052.

Languages

Spanish	**C2**	*Mother tongue*
English	**C2**	*High speaking, writing, and reading comprehension level.*
German	**C1**	*Good speaking, writing and reading skills. Courses until B2 level at Goethe Institute, Deutsch Academy and Sprachenzentrum (Universität Wien). +8 years studying and working in german-speaking countries.*
French	**A2**	*600 hours at high school.* *Good understanding and writing skills and basic oral skills.*