Energy Consumption Analysis and Design of Energy-Aware WSN Agents in fUML

Luca Berardinelli[1,3], Antinisca Di Marco[1,2], Stefano Pace[1,2(✉)], Luigi Pomante[1,2], and Walter Tiberti[1,2]

[1] Dipartimento DISIM, University of L'Aquila, L'Aquila, Italy
[2] Center of Excellence DEWS, University of L'Aquila, L'Aquila, Italy
{berardinelli,dimarco,pace,pomante}@univaq.it
[3] Business Informatics Group, Vienna University of Technology, Wien, Austria
wtuniv@gmail.com

Abstract. Wireless Sensor Networks (WSN) are nowadays applied to a wide set of domains (e.g., security, health). WSN are networks of spatially distributed, radio-communicating, battery-powered, autonomous sensor nodes. WSN are characterized by scarcity of resources, hence an application running on them should carefully manage its resources. The most critical resource in WSN is the nodes' battery.

In this paper, we propose model-based engineering facilities to analyze the energy consumption and to develop energy-aware applications for WSN that are based on Agilla Middleware. For this aim i) we extend the Agilla Instruction Set with the new `battery` instruction able to retrieve the battery Voltage of a WSN node at run-time; ii) we measure the energy that the execution of each Agilla instruction consumes on a target platform; and iii) we extend the Agilla Modeling Framework with a new analysis that, leveraging the conducted energy consumption measurements, predicts the energy required by the Agilla agents running on the WSN. Such analysis, implemented in fUML, is based on simulation and it guides the design of WSN applications that guarantee low energy consumption. The approach is showed on the Reader agent used in the WildFire Tracker Application.

Keywords: fUML · Model-driven analysis · Tool support · WSN

1 Introduction

A Wireless Sensor Network (WSN) consists of spatially distributed autonomous sensors that cooperate in order to accomplish a task. Sensor nodes are small, low-cost, wireless and battery-powered devices. They can be easily deployed to monitor several environmental parameters and they create large-scale flexible architectures. Sensors can be distributed everywhere and they enable different applications such as domotics, disaster relief, alternate reality game.

This work is supported by the EU-funded VISION ERC project (ERC-240555), the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

G. Taentzer and F. Bordeleau (Eds.): ECMFA 2015, LNCS 9153, pp. 1–17, 2015.
DOI: 10.1007/978-3-319-21151-0_1

The growing request for applications running on WSN showing high quality, demands for suitable design and analysis approaches, that consider non-functional properties already in an early development stage to guarantee the fulfillment of non-functional requirements. Model-based engineering facilitates an early analysis of non-functional properties based on design models. In this respect, UML and its profiles can be chosen as the primary design notation and analysis-specific languages suitable for model-based analysis. The complexity brought by the required set of model-based methodologies, notations and tools may hinder the adoption of UML-based approaches in the WSN domain. The specific nature of sensors complicates the development of applications, because the quality of the services they provide is influenced by several factors like network availability, battery levels. Despite this, a WSN must continue providing its services as long as possible, and with the best effort trying to guarantee network longevity. Traditionally, WSN applications have been developed by with a code-and-fix approach, that is by directly programming nodes with the use of low level primitives. This approach, neglecting design and quality validation phases, results in not structured, hard to maintain code with the risk of missing non-functional requirements and compromising the system usability. Indeed, the system's non-functional properties must be considered as earlier as possible in the system life-cycle to guarantee their fulfillment.

In this work, we propose model-based engineering facilities to analyse the energy consumption and to develop energy-aware applications for WSN that are based on Agilla Middleware. For this aim i) we extend the Agilla Instruction Set with the new `battery` instruction able to sense at run-time the battery of a WSN node; ii) we measure the energy that the execution of each Agilla instruction consumes on a target platform; and iii) we extend the Agilla Modeling Framework (AMF)[1,2] with a new analysis that, leveraging on the conducted energy consumption measurements, predicts the energy required by the Agilla agents running on the WSN. AMF is an executable model library we implemented to model and analyze Agilla based applications running on WSN. It permits to conveniently design agent-based software applications and to carry out its analysis upon the execution of the corresponding UML model. It exploits the recently introduced Foundational UML (fUML) standard that provides a formal semantics of a subset of UML enabling the execution of UML models.

The paper is organized as follows: Section 2 describes the new version of Agilla middleware we implemented; Section 3 shows the process of measurement of the energy consumption of Agilla instructions; Section 4 introduces the used case study; Section 5 illustrates the Agilla Modeling Framework we implemented and its new energy consumption analysis; Section 6 shows the application of the new energy; Section 7 illustrates related work and Section 8 comments the main advantages of our approach; Finally, Section 9 concludes the paper.

2 Agilla v2.0: Energy-Aware Middleware

With the introduction of TinyOS 2.x (TOS2, 2005), that is not compatible with the older TinyOS versions (TinyOS 1.x, TOS1), there is no possibility to exploit

some of the old reference applications. Agilla [3] is one of these applications, that allows creating, migrating and destroying software agents on WSN nodes at run-time, without service interruption. To continue working with Agilla we needed to perform a porting of the original Agilla to TinyOS 2.x. Such a porting, called Agilla2.0, was also released to the TinyOS community and it allows to exploit increased reliability of TinyOS2.x and new sensor node technologies (i.e. those not supported by TinyOS1.x), in particular the Memsic IRIS node[1].

Agilla2.0 is fully ISA-compatible with the old version, hence it is possible to execute all the old Agilla applications. To re-use the timing and performance analysis defined in [1,2] also for Agilla2.0/TOS2.x, we performed once again the measurement of the execution time of each single Agilla2.0 instruction. The techniques used in [1] were partially adapted by exploiting a free-running HW counter to timestamp start and end times of each instruction. Such timestamps have been then collected and used to evaluate offline (in order to be as less as possible intrusive in the code behavior) the average execution time for each instruction and other statistical information. The AMF framework hence contains all the information needed to execute timing and performance analysis both in case of Agilla1/TinyOS1 and in case of Agilla2.0/TOS2.x.

In order to also provide an effective run-time support to energy analysis and WSN lifetime estimation, the Agilla2 ISA has been extended by means of a new instruction (i.e. battery) that is able to provide information about the current battery voltage of a node to an agent. This made Agilla2.0 energy-aware.

The `battery` instruction provides voltage information with a precision of 100 mV. Technically, the new instruction reads data from the ADC and puts it on the top of the agent stack after some processing. Final measurement unit is 100 mV for each unit of the stacked integer value (e.g. a value 33 represents 3.3V).

The ISA extension has followed the procedure reported in the original Agilla website[2]. In the following the main issues are described with some details.

VoltageC Module - Memsic IRIS nodes can obtain information about the voltage by means of the VoltageC TOS2 module. Since the power supply voltage is attached to a dedicated ADC channel, this module is able to retrieve it easily. It uses a split-phase command called read() that retrieves voltage data in term of ADC divisions. Such data is then converted by means of the formula 1. It is a conversion formula from ADC divisions to the real voltage value, where we just added the 10 factor and -36 in order to adjust the offset. The formula takes into account the features of the IRIS ADC[3]:

$$V = 10 * [(1100 * 1024)/read()] - 36 \quad (1)$$

where: i) the 10 factor is used to convert from Volt to Volt/10 (decivolt); ii) the 1100 value refers the internal reference voltge of iris motes; iii) the 1024 value

[1] http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf
[2] http://mobilab.wustl.edu/projects/agilla/docs/tutorials/10_add_instr.html
[3] http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

is the number of ADC steps (10 bit resolution); iv) read() is the actual value read via VoltageC module; v) The 36 (360mV) value is an offset adjustment needed to obtain a correct value.

OPBatteryM Module. By following the standard procedure to extend the Agilla2 ISA, we created a module called OPBatteryM, that contains all the instructions needed to read and process the voltage value (as described above). Such a module is also related to a configuration called OPBatteryC that we inserted in the *opcodes* Agilla2.0 directory. Moreover, it makes use of all the interfaces needed to perform its work: previously cited VoltageC and also all the Agilla2.0 modules that allow to interact with Agilla2.0 main elements (e.g. stack, context, tuple space, etc.). Finally, to be fully usable in the Agilla2.0 MW, the module provides an implementation of the interface BytecodeI. The main code related to the execution of a new instruction in the Agilla2.0 MW context can be found in the execute command of such an interface. In fact, such a command stops the Agilla2 agent execution and starts the split-phase operation needed to read the voltage. When the read is done, the value is processed and then passed to the Agilla2.0 code that writes it on the top the stack (interface OPStackI). Finally, the agent execution is resumed. To conclude ISA extension, the Agilla2.0 Agent Injector should be able to identify the new instruction. For this, we modified the related Java code by inserting `battery` and associating to it a unique opcode. In this way, the OPBatteryM module can be properly exploited when the `battery` instruction is detected.

Time Measurements of the `Battery` Instruction. The code related to `battery` instruction is nothing more than an ADC read, same as the `sense` instruction, for example. So, its execution time can be estimated as equal to the one of the `sense` instruction. Code debugging and time measurements have confirmed the validity of this assumption.

3 Energy Consumption Measurements of Agilla Instructions

The concept of low-power is strictly related to WSN since their invention. For this, one of the main goals of this work is to improve the power-awareness of adopted devices and related software. Considering the previously described Agilla2.0, we needed an analysis to consider the energy consumption issues related to the adoption of such a middleware. In particular, starting from Agilla2.0 instructions' execution times, we estimated their energy consumption. For this, we decomposed each instruction into several basic operations and, for each of them, we measured the power involved on IRIS nodes as detailed below. Finally, by combining the energy consumption of basic operations and the timing information, we estimated the consumption of each Agilla2.0 instruction.

In order to estimate the energy consumption, we adopted the milli Ampere per hour (mAh) metric. Such metric is very useful, since it is directly related to information provided for some commercial batteries. So, by knowing timing

information for each instruction and decomposing each of them in basic operations, we have measured the power related to each basic operation and we have combined them with their duration to obtain mAh estimations. In particular, the considered basic operations are the following ones: i) microprocessor processing, ii) radio RX/TX, and iii) leds ON/OFF. To measure power consumption associated to basic operation, we exploited the experimental setup shown in Figure 1. By means of the voltage measured on R_TEST ($1.2ohm$) we evaluated the power by using an oscilloscope.

To associate measured current to the basic operations, we created a proper Agilla2.0 agent with the following 5-phases behavior (10 seconds for phase): *i)* microprocessor activity only, *ii)* microprocessor and led activities, *iii)* radio ON in reception mode, *iv)* radio ON/OFF switching, and *v)* periodic radio transmission. Finally, we could estimate mAh consumption for each Agilla2.0 instruction.

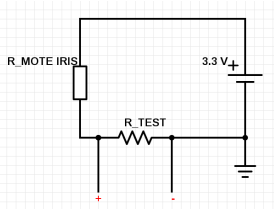


Fig. 1. Energy measurement experimental setup

3.1 Battery Behavior Analysis

Once estimated Agilla2.0/IRIS energy consumption, some considerations about commercial batteries behaviours have to be made in order to be able to use consumption data in a correct way. In particular, nominal mAh could be affected by operating conditions (e.g. temperature, humidity, etc.). So, in order to be able to estimate the energy status of a node, we analyzed some info about AA batteries normally used to power WSN nodes [4]. In particular, for IRIS nodes, we need two AA batteries, for a total of about 3V. Then, by means of some stress tests with different reference voltages, we identified a critical threshold of 1.7 V under which the node is no more active. Once under such voltage, only a nominal voltage (3V-3.3V) followed by a reset allows the node to start working again. Moreover, we identified four operational zones:

- more than 3V (up to 3.3V): ideal conditions.
- 3V-2.8V: good conditions.
- 2.8V-2.4V: still working but with some warning.
- Under 2.4V: serious warning.

The last zone, considering the information reported in [4], represents a point in which energy consumption analysis could no more be representative of a real operative condition due to a possible very fast batteries failure. It is worth noting that this means that, also if the available mAh have not been totally used, it is not possible to make reliable assumptions about their availability in a short period. Maybe it could be better to change the batteries, if possible, or to consider using the node only for non critical activities.

4 Case Study

In this paper we reuse and extend the Wildfire Tracking Application (WTA), an existing case study originally described in [3] and already adopted in our previous work [1,2]. The WTA software is deployed on a WSN distributed into a region that is prone to fires. It must detect a fire and determine its perimeter. Figure 2 shows the high-level behavior of the application.

The original WTA is composed by three Agilla agents. The *Reader* agent runs on all the WSN nodes and is programmed to sense the temperature at regular time intervals of 1/8 of second. The readings are sent to the Base Station (BS). A *Forwarder* agent, running on the BS, forwards the sensed values up to the PC, where the temperature level is evaluated. Once a fire has been detected, a *Tracker* agent is injected from the PC into the WSN, through the BS, in order to dynamically determine the perimeter of the fire.

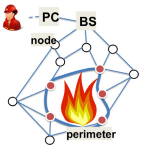


Fig. 2. The WTA App

In this paper, we carry out the energy analysis of two variants of the `Reader` agent by extending the original version (Figure 3) with battery awareness capability through our new `battery` instruction (battery-aware reader, `baReader`, Figure 4). The Agilla code of both agents, `Reader` and `baReader`, consists of Agilla instructions (e.g., `pushc`) that are grouped in tasks (e.g., `BEGIN`, depicted as small gray nodes in the control flow graph. By default, the Agilla middleware executes tasks sequentially by scheduling the next tasks after the latest

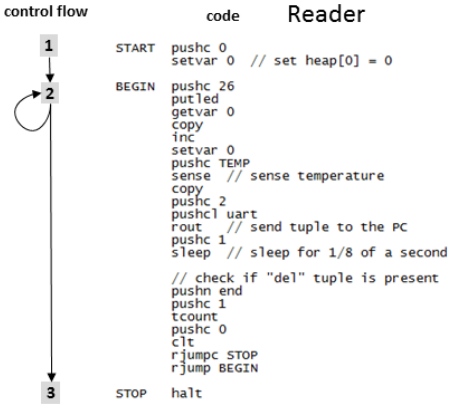


Fig. 3. The standard Reader (`Reader`)

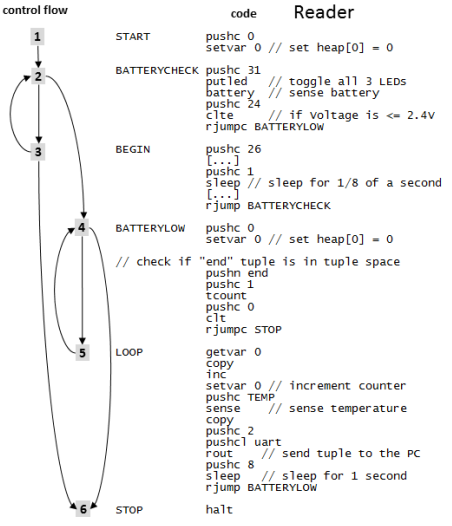


Fig. 4. The `battery`-aware Reader (`baReader`)

instruction of the previous one. However, goto-like instructions such as `rjump` and `rjumpc` allow (un)conditional relative jumps [3].

This `Reader` version is programmed to `sense` the temperature at regular time intervals of 125 ms (default waiting time of `sleep` in task `BEGIN`) and to send it (`rout`) to the BS. Since `Reader` is not `battery`-aware, no further actions are taken in case of low battery level.

The `baReader` version is extended with a battery-level check (`BATTERYCHECK` task) before sensing the temperature. In case of low voltage (lower or equal to 2.4 Volts), the execution jumps to the `BATTERYLOW` task, from where a `LOOP` task is entered, where the same actions as the `BEGIN` task are executed (a counter is incremented, and the temperature is sensed and sent to the PC) but with a lower frequency.

Finally, both `Reader` and `baReader` agents can stop their execution by checking the presence of a *del* entry in the Tuple Space (in `BEGIN` and `BATTERYLOW` tasks, respectively). If found, both `Reader` and `baReader` agents jump to the `STOP` task where the `halt` instruction is executed.

5 The Agilla Modeling Framework

The Agilla Modeling Framework (AMF)[4] is part of a model-driven, tool-supported approach (see Figure 5) to design and analyze Agilla applications [3] suitably modeled through the Foundational UML (fUML) [5], a new standard of the Object Management Group (OMG) that defines the operational semantics of a (strict) UML subset.

Figure 5 sketches the fUML-driven approach supported by AMF. Artifacts (*fUML Model* and *Analysis Results*) and functionalities (*Parsing*, *Instruction Semantics Simulation*, *Trace Generation*, *Timing Analysis*, *Performance Analysis*, *Energy Analysis*) are depicted as rectangles and rounded boxes, respectively, while dashed arrows connect functionalities and related artifacts with labels detailing their relationships.

The AMF nature is two-fold. On the one hand AMF is a *fUML model library*, i.e., a model consisting of reusable classes and activities for fUML models representing Agilla applications as created by the *User*s. On the other hand, AMF is also a *tool* whose analysis algorithms are *implemented* (i.e., modeled) through executable UML activities and *Analysis Results* are saved as slot values [6] within the same executable *fUML Model*.

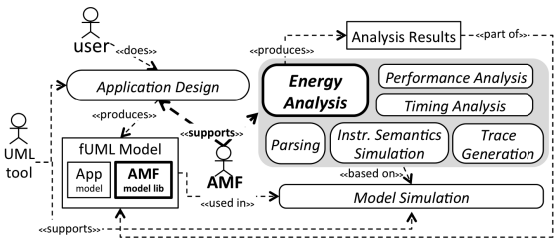


Fig. 5. Actors, functionalities and artifacts in AMF

[4] http://sealabtools.di.univaq.it/tools.php

The remainder of this section provides a brief background on fUML and AMF design choices, then describes the new *Energy Consumption Analysis* capability. The other AMF capabilities (parsing, instruction semantics simulation, trace generation, timing and performance analyses) are described in our previous works [1,2].

5.1 The Foundational UML and AMF Design Choices

fUML[5] defines the operational semantics of a strict UML subset (cf. Figure 7a) that includes Classes, Common Behaviors, Activities, and Actions. Neither heavy (e.g., metamodel changes) nor lightweight extensions (e.g., UML profiles) are required. fUML enables the execution of UML models including Classes with their own Structural (i.e., attributes) and BehavioralFeatures (i.e., operations). Behavioral specifications (i.e., operations' body) are modeled through Activities.

The fUML standard goes along with a Java-based reference implementation of an fUML virtual machine (fUML VM)[5], to simulate fUML models. Free open source and commercial UML modeling tools exist that embed this reference implementation within their modeling environments, like Papyrus[6] and MagicDraw[7]. We adopted the latter, in conjunction with its plug-in Cameo Simulation Toolkit, as modeling and simulation environment (i.e., UML tool in Figure 5).

Figure 7b shows an excerpt of the user-defined fUML Model for the Reader agent. Its structure and behavior are modeled through composition of Classes and hierarchical Activities, respectively. The AMF Model Library helps the structural modeling of Agilla applications. It includes:

- An abstract, hierarchical structure of classes (`AppComp`, `AgentComp`, `TaskComp`, `InstrComp`). User-defined fUML Model of Agilla applications are modeled with classes and proper *Generalization* relationships. (e.g., the `Reader` agent and its own `START`, `BEGIN`, and `STOP` tasks).
- A set of 74 concrete `InstrComp` classes (e.g., `pushc`, cf. Figure 7c) covering the whole Agilla instruction set [3], including a new class for the `battery` instruction.

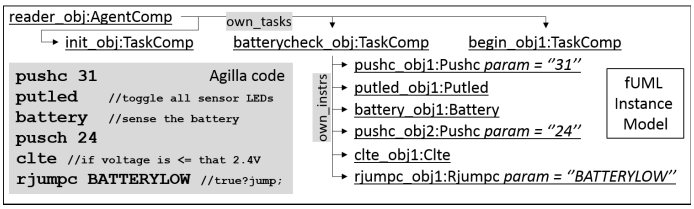


Fig. 6. `baReader` agent instance model at simulation time

[5] http://fuml.modeldriven.org
[6] www.papyrusuml.org/
[7] www.nomagic.com/products/magicdraw.html

At simulation time, the fUML VM generates a so-called *instance model* and ignores the non-executable part (e.g., Sequences, Statemachines, Deployments or annotations [7]). InstanceSpecifications, Links, and Slots elements are generated within the instance model as counterparts of Classes, Associations and Properties, respectively.

The execution of fUML Activities then reads, adds, deletes, and modifies elements of the instance model. Figure 6 shows a graph-like excerpt of the Reader agent instance where InstanceSpecification (e.g., reader_obj) and Links (e.g., own_tasks) are depicted as nodes and arrows, respectively.

The behavioral specification of an `AgentComp` comprises a layered set of fUML Activities representing i) the flow of tasks (e.g., `START`, `BEGIN`, and `STOP`, cf. Activity Level 1 in Figure 7b), ii) the flow of instructions for each task (cf. Level 2 in Figure 7b) and, iii) AMF algorithms like the Energy Analysis (cf. Level 3 in Figure 7b). The AML user is in charge of modeling the task flow from scratch. On the contrary, the instruction-level flow is built by dragging and dropping instruction actions (e.g., `halt`) from the library and, if needed, manually add parameters through typed pins (e.g., `26:String` for `pushc`)[6]. Such actions transparently invokes further nested fUML activities realizing instruction semantics and AMF analysis algorithms (cf. Level 3 in Figure 7c).

The AMF Instruction Semantics Simulation is in charge of traversing an agent's instance model and collecting instruction-specific slot values for the sake of Timing, Performance and Energy Consumption Analyses.

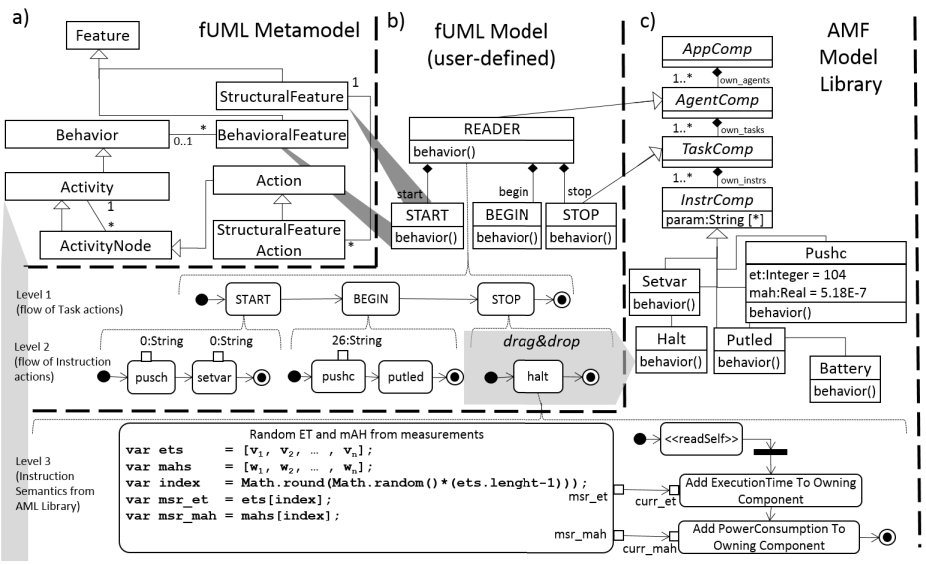


Fig. 7. Modeling with fUML and the AML Library

In [1] we modeled and implemented[8] the timing analysis algorithm which collects and sums slot values corresponding to measured instruction execution times (e.g., the *et* default value of `pushc` in Figure 7c). In [2] we further enhanced AMF with a First Come First Served (FCFS) scheduling policy required by the Performance Analysis capability to generate additional timed properties like waiting (*wt*), and completion times (*ct*). Similar to the timing analysis algorithm, all these timed properties (*et*s, *wt*s, *ct*s), stored as slot values in the Agent's instance model, are collected during the instruction semantics simulation to finally obtain performance indices (e.g., response time) from fUML models of Agilla agents.

In this paper, we enrich AMF with the Energy Consumption Analysis capability by reusing the same algorithm design. AMF collects the measured energy consumption of each Agilla Instruction while traversing the instance model during the fUML model simulation (see Figure8a).

Figure 7c shows a generic energy consumption analysis algorithm as executed during the `behavior()` operations of each InstrComp instance. In particular, a `Random ET and mAh from measurements` JavaScript snippet [9] randomly selects a couple of values v_i and w_i for et and mah, respectively, from two arrays of n measurements and adds them to the corresponding partial sums saved as slots of the owning `TaskComp` instances. A similar activity calculate the execution time and power consumption of an `AgentComp` instance from partial sums stored in the owned `TaskComp` ones.

6 Energy Consumption Analysis

The analysis work flow is shown in Figure 8a. We analyzed the energy consumption of `Reader` and `baReader` agents in three different scenarios (Sc). In each scenario, we simulate the fUML model of both `Reader` and `baReader` agents, in isolation, from the beginning (i.e., the execution of the first Agilla instruction, `pushc`, see Figures 3 and 4) up to the end (i.e., the first execution of the `halt` instruction of the `STOP` task). Each simulation run corresponds to the execution occurrence shown in the sequence diagram in Figure 8b[10]. The three different simulation scenarios Sc1, Sc2, and Sc3, differs from the agent's `sleep` time between two consecutive `sense` and dispatch (`rout`) of temperature to the base station (BS). By default, a `sense 1` instruction hibernates an agent for 125 milliseconds. In our scenarios Sc1, Sc2, and Sc3, we hibernate the agent for 30, 60, and 120 seconds. This behaviors are implemented by setting the `sleep` parameter to 240, 480, and 960, respectively. At the modeling level, setting such scenarios consists in i) saving distinct fUML models for both `Reader` and `baReader` for a total of six artifacts (e.g., .mdzip files using MagicDraw), ii) opening each model and updating the integer *value specification* [6] of the `sleep` action (imported from the Agilla InstructionSet in the AMF model library) within the containing

[8] AMF is a tool directly designed in fUML realizing the motto "the model is the code".
[9] Set as body of a UML Opaque action.
[10] Both state machine and sequence diagram are not part of the fUML model and it is used for explanation purposes.

`BEGIN` task action, and iii) starting the model simulation which repeat the agent simulation for a user-defined number of runs (see Figure 8).

Figure 9 shows the analysis results. Minimums, maximums, and averages of the energy consumption for both `Reader` and `baReader` agents have been calculated through AMF by executing the corresponding fUML model. Each scenario has been executed 100 times.

As expected, the `baReader` agent saves energy (from 38,37% in Sc1 up to 43,22% in Sc3, see Figure 9) w.r.t. the battery unaware `Reader` one. However, focusing on analysis results of the same reader version in different scenarios, we observe that the energy consumption is almost the same, i.e., it is invariant w.r.t.

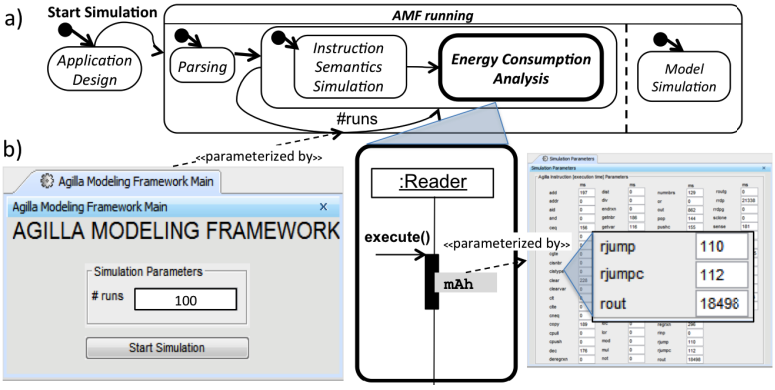
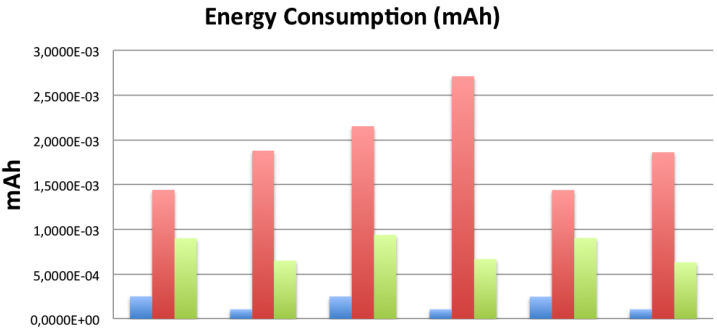


Fig. 8. Energy consumption analysis work flow



Reader	R	baR	R	baR	R	baR
Scenario	Sc1	Sc1	Sc2	Sc2	Sc3	Sc3
Sleep Time (ms)	30000	30000	60000	60000	120000	120000
Minimum (mAh)	2,5139E-04	1,0679E-04	2,5015E-04	1,0679E-04	2,4749E-04	1,0678E-04
Maximum (mAh)	1,4410E-03	1,8793E-03	2,1524E-03	2,7106E-03	1,4390E-03	1,8612E-03
Average (mAh)	9,0032E-04	6,5067E-04	9,3812E-04	6,6762E-04	9,0379E-04	6,3105E-04
Energy Saving on Avg (%)	38,37%		40,52%		43,22%	

Fig. 9. Energy consumption analysis results

to the sensing frequency. It may be caused by the busy wait of the `sleep` instruction which does not turn off but only hibernates the agent then causing a lower, but not null, energy consumption while waiting for resuming the agent behavior. As a consequence, the longer is the busy wait (i.e., greater is the `sleep` integer parameter), the higher is the energy required to complete the execution of the `sleep` instruction. In addition, we also observe a lower difference among minimums and maximums of the `baReader` version w.r.t. the corresponding results for the `Reader` version in all scenarios. These results are caused, on the one hand, by the greater complexity of the `baReader` version whose execution may generate longer control flows (e.g., more loops) thus requiring more energy to their completion. On the other hand, it may be caused by the limited number of simulation runs (100) for the proposed scenarios. Even if, as already experienced in our previous works [1,2,8], we are limited by the current scalability problem of the fUML virtual machine which it is not optimized for simulation-based analysis [9], our approach is promising since we are able to integrate several non-functional analyses in a single tool. This will open to an easy and consistent multi-dimensional analyses environment where trade-off analysis are possible. Moreover, since we do not move from different modeling notations and analysis tools, we are sure that the results of the analyses refer to the same software system. Instead when we move to different modeling notations and tools, this warranty is obtained if it is proved that the provided M2M transformation does not change the modelled system. Up to day this proof is not provided for most of the transformation in literature, but it is leaved to intuition. Finally, in literature, we can find some example of multi-dimensional analysis environment, such as Palladio [10], using proprietary modeling notations. Differently from them, we do not use own notations, but UML that is a standard de-facto modeling language hence the impact of AMF is wider and more general.

7 Related Work

fUML - In this work, that directly stems from [1] as well as in [7] and [11], we pursue a similar goal but aiming at a tighter integration between fUML and analysis methodologies. In particular, we showed in [7] and [11] how performance analysis can be conducted on annotated fUML models by generating and analyzing traces compliant with a fUML runtime metamodel [12]. Tool support is provided through a Java-based Eclipse plug-in that suitably interacts with the fUML VM during its execution. In [1] and in this subsequent work, we further emphasize the role and importance of fUML by directly designing (that is, implementing) the analysis tool as a fUML model library.

In [1,7,11] and in this work as well, the expected benefits of directly utilizing the execution of UML models for carrying out model-based analysis are twofold: *(i)* the costly translation of UML models into formal languages dedicated to specific analysis purposes is avoided and, hence, *(ii)* the implementation and maintenance of supporting analysis tool sets is eased significantly. With AMF approach presented in this paper, we offer both of these benefits and showcased

them by developing a performance analyzer that implements an analysis method directly on UML models for WSN applications. A translation of UML models into an analysis model can now be omitted and it is not necessary to use additional external tools for analysis purposes.

In [13], the authors present a behavioural design solution for WSN, abstracting the complicated dynamic aspects of WSN software systems through the concept of activity-driven states. This provides the programmer with concrete design elements that can be directly mapped to the constructs of target programming languages opening towards more accurate verification and validation of software systems for WSNs. Differently from [13], our approach gives the possibility to design software for WSN with fUML and to map it to the Agilla target language and provides functional, timing, performance and energy analyses on the UML models via simulation.

Romero and Ferreira propose an MDA-based approach applied to the domain of space real-time software for sake of code generation and schedulability analysis [14]. In their approach platform independent models and test cases are specified using fUML activities. Non-functional properties are annotated on the activities using the UML profile MARTE. However, for carrying out the schedulability analysis, fUML activities are translated into AADL models [15].

Benyahia et al. [16] evaluate how well the current fUML semantics supports the formalization of concurrent and temporal semantic aspects which is required for the design and analysis of real-time embedded systems. They illustrate how the standard fUML execution model, as well as the fUML VM, have to be extended for this purpose to explicitly incorporate a *scheduler* into fUML that, at each step of the model execution, determines the activity node to be executed next according to certain scheduling policies (e.g., first-in-first-out (FIFO)). The same limitation has been addressed by Abdelhalim et al. [17]. In contrast to Benyahia et al. [16], they do not propose an extension of the standard fUML execution model but rather present a model-based framework that translates fUML activities into communicating sequential processes (CSP) for performing a deadlock analysis detecting possible scenarios leading to deadlocks which are provided as UML sequence diagrams.

Energy Analysis - In [18], a Model-Driven approach is used to separately model the software architecture of a WSN, the low-level hardware specification of the WSN nodes and the physical environment where nodes are deployed in. The framework can use these models to generate executable code to analyze the energy consumption of the modeled application. The last three approaches aim at evaluating the quality of the WSN application (that is, its performance for the first two and the energy consumption for the last one). Instead, our approach aims at performing non-functional analysis, including energy analysis, on the models with no model-weaving processes. We also generate the executable code, ready to be deployed and run on a node.

In [19], the authors first break down the energy consumption for the components of a typical sensor node, and they discuss the main directions to energy conservation in WSNs. Then, they present a systematic and comprehensive

taxonomy of the energy conservation schemes. This work guided us through the step of integrating in the framework an energy consumption analysis for Agilla agents. In [20], the authors propose an energy-efficient MAC protocol for WSNs, so they save energy at lower level, while we save energy at the application level.

The authors of [21] formulate the energy consumption and study their estimated lifetime based on a clustering mechanism with varying parameters related to the sensing field (e.g., size and distance). They provide numerical analysis and results of the energy consumed by the WSN and the WSN's lifetime, but they only consider energy consumed in communication, without taking into account energy consumed in data processing, etc. Further, their analysis is generic and does not take into account differences between different hardware platforms. With our approach, instead, we take into account the energy consumed by every single instruction of an agent, and the energy consumption of each instruction is evaluated in the specific hardware platform where it is executed. So, when changing the platform, the energy consumption value may vary also, if the hardware has got different characteristics. In general, great research effort is focused on optimizing protocols and clustering schemes for performance and energy saving, but there's a very poor work on designing and generating good applications for WSNs, from performance and energy viewpoint. This is instead our research direction: We design, simulate and analyze Agilla-based WSN applications, in order to automatically obtain application code that meets the non-functional constraints.

8 Discussion

It is worth noting that, in this paper, we are extending the analysis capabilities of AMF with the intent of showing the suitability of fUML for i) the design and analysis of WSN applications, and ii) the design of more and more complex analysis tools as fUML model libraries, at the same time. In accordance with our background and research goals, we primarily focus on assessing the exploitation of fUML and related technologies in the WSN and extra-functional analysis domains. We consider AMF and, more in general, the underlying fUML-driven approach proposed in this paper, an initial as well as a first practical evaluation of the impact that fUML and its related technologies may have i) on expectations from UML practitioners and ii) on future research directions in MDE [22,23]. With this work, we show that both the design and analysis of WSN as well as tool development are feasible activities with fUML. While pursuing our goals, we experienced both opportunities and limitations related to the usage of fUML.

fUML is a young OMG standard, published on February 2011. It makes a strict (and then easier to learn) subset of UML executable. By leveraging their current background in UML, both researchers and practitioners can already adopt it for their specific purposes. At the time of writing, the (positive or negative) impact of fUML on daily modeling activities still have to be assessed (e.g., [23] was concluded in July 2011). fUML promotes model reuse through executable model libraries, like AMF, and it may be compared to a new programming language that, however, still suffers from the lack of an adequate

support in term of built-in libraries. In AMF, for example, we had to model from scratch common auxiliary data structures like queue and stack. The modeling effort required to create executable model libraries may then be high. For example, fUML activities are much more detailed than non-executable ones and so far usually disregarded details, like input/output pins, have to be systematically modeled to allow a correct execution. Being aware of this, we worked on AMF for its users to simplify the modeling of agents' control flows.

In addition, still few UML modeling tools exist that provide plug-ins that support the simulation of their models. We choose MagicDraw and its plug-in Cameo Simulation Toolkit to support the modeling and simulation tasks in AMF. However, being fUML models also valid UML models, such artifacts may be exchanged among any UML modeling tools supporting common serialization formats (e.g., XMI and Eclipse UML).

In this work, we are adopting fUML to design, from scratch, an analysis tool. From [1] on, the AMF executable model is growing fast to support its new functionalities, including also possibly heavy computational tasks, like performance analysis is. It is worth noting that AMF can be seen as a layered tool infrastructure and all the AMF functionalities run within an hosting UML Modeling environment which, in turn, run on atop a Java Virtual Machine. This layered infrastructure may cause scalability issues for analysis tools, like AMF, running on the topmost layer. In this work, we mainly focused on promoting MDE approaches in the WSN domain using fUML as the only modeling and simulation notation. Assessing the maturity level of fUML and its underlying technology for such a challenge task is out of scope of this paper and left for future work.

9 Conclusion

AMF is an ongoing work that spread MDE methodologies and tools in the Wireless Sensor Network domain. We developed a model-driven approach that allows both modeling and multiple analysis (timing, performance and energy consumption) of software for WSN nodes running the Agilla mobile agents-based middleware. We adopted fUML, a strict executable UML subset, as design notation for AMF users and as well as development language for AMF itself. We provided modeling guidelines to AMF users in order to obtain an executable specification directly in UML, without the need of learning ad-hoc notations and tools. In this respect, thanks to its fUML native compatibility with UML, our approach is tool-supported by construction and can leverage many existing, industrial-strength UML-based tools. We simulated different fUML models representing two (un)aware variants of a reader agent in three different scenarios by feeding the analysis with measured data obtained from a real Agilla execution platform. For this purpose, we extended both the Agilla middleware and the corresponding AMF model library with a new `battery` instruction.

As future research goals, we plan to improve the design and scalability of existing analysis algorithms in fUML and to combine them in trade off analyses.

References

1. Berardinelli, L., Di Marco, A., Pace, S., Marchesani, S., Pomante, L.: Modeling and timing simulation of agilla agents for WSN applications in executable UML. In: Balsamo, M.S., Knottenbelt, W.J., Marin, A. (eds.) EPEW 2013. LNCS, vol. 8168, pp. 300–311. Springer, Heidelberg (2013)
2. Berardinelli, L., Di Marco, A., Pace, S.: fUML-driven design and performance analysis of software agents for wireless sensor network. In: Avgeriou, P., Zdun, U. (eds.) ECSA 2014. LNCS, vol. 8627, pp. 324–339. Springer, Heidelberg (2014)
3. Fok, C.L., Roman, G.C., Lu, C.: Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Trans. Auton. Adap. **4**(3), 16 (2009)
4. A Comparison of Primary Battery Performance using a Solartron 7150plus Multimeter. http://www.drkfs.net/bestbattery.htm
5. OMG. Semantics of a foundational subset for executable UML models (2011)
6. OMG. UML, Superstructure, Version 2.4.1 (2011)
7. Berardinelli, L., Langer, P., Mayerhofer, T.: Combining fUML and profiles for non-functional analysis based on model execution traces. In: QoSA (2013)
8. Berardinelli, L., Cortellessa, V.: fUML-driven performance analysis through the moses model library. In: ACES-MB, MoDELS, pp. 34–43 (2014)
9. Tatibouet, J., Cuccuru, A., Gérard, S., Terrier, F.: Principles for the realization of an open simulation framework based on fuml (wip). In: Proc. of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium, p. 4. Society for Computer Simulation International (2013)
10. Brosig, F., Meier, P., Becker, S., Koziolek, A., Koziolek, H., Kounev, S.: Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. IEEE Trans. Software Eng. **41**(2), 157–175 (2015)
11. Fleck, M., Berardinelli, L., Langer, P., Mayerhofer, T., Cortellessa, V.: Resource contention analysis of service-based systems through fUML-driven model execution. In: Proc. of NiM-ALP, p. 6 (2013)
12. Mayerhofer, T., Langer, P., Kappel, G.: A runtime model for fUML. In: Proc. of the Int'l Workshop on Models@run.time (MRT 2012) at MODELS (2012)
13. Taherkordi, A., Eliassen, F., Johnsen, E.B.: Behavioural design of sensor network applications using activity-driven states. In: Int'l Workshop on Soft. Eng. Sensor Network App. (SESENA), pp. 13–18 (2013)
14. Romero, A.G., Ferreira, M.G.V.: An approach to model-driven architecture applied to space real-time software. In: Proc. of the Int'l Conf. on Space Op. (2012)
15. Feiler, P.H., Gluch, D.P.: Model-based engineering with AADL: An introduction to the sae architecture analysis & design language. Addison-Wesley (2012)
16. Benyahia, A., Cuccuru, A., Taha, S., Terrier, F., Boulanger, F., Gérard, S.: Extending the standard execution model of UML for real-time systems. In: Hinchey, M., Kleinjohann, B., Kleinjohann, L., Lindsay, P.A., Rammig, F.J., Timmis, J., Wolf, M. (eds.) DIPES 2010. IFIP AICT, vol. 329, pp. 43–54. Springer, Heidelberg (2010)
17. Abdelhalim, I., Schneider, S., Treharne, H.: An integrated framework for checking the behaviour of fUML models using CSP. Int'l Journal on Software Tools for Technology Transfer, 1–22 (2012)
18. Doddapaneni, K., Ever, E., Gemikonakli, O., Malavolta, I., Mostarda, L., Muccini, H.: A model-driven engineering framework for architecting and analysing wireless sensor networks. In: Int'l Workshop on Soft. Eng. Sensor Network App. (SESENA), pp. 1–7 (2012)

19. Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy conservation in wireless sensor networks: A survey. Ad Hoc Networks **7**(3), 537–568 (2009)
20. Ye, W., Heidemann, J., Estrin, D.: An energy-efficient mac protocol for wireless sensor networks. In: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2002, vol. 3, pp. 1567–1576. IEEE (2002)
21. Duarte-Melo, E.J., Liu, M.: Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks. In: Global Telecommunications Conference, GLOBECOM 2002, vol. 1, pp. 21–25. IEEE (2002)
22. France, R.B., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: Future of Software Engineering, pp. 37–54 (2007)
23. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: A survey. IEEE Trans. Softw. Eng. **39**(6) (2013)