Daniel-Jesus Munoz, José A. Montenegro, Mónica Pinto, Lidia Fuentes,

Energy-aware environments for the development of green applications for cyber–physical systems,

Future Generation Computer Systems,

Volume 91,

2019,

Pages 536-554,

ISSN 0167-739X,

https://doi.org/10.1016/j.future.2018.09.006.

(https://www.sciencedirect.com/science/article/pii/S0167739X18307295)

Abstract: Cyber–physical Systems are usually composed by a myriad of battery-powered devices. Therefore, developers should pay attention to the energy consumption of the global system so as not to compromise the system lifetime. There are plenty of experimental studies that give hints about how to reduce the energy consumption. However, this knowledge is not readily available for the software developers of cyber–physical systems. They normally use software development environments that do not provide useful advice about the energy consumption of the software solutions being implemented. In this paper, we propose a Developer Eco-Assistant to integrate the experimental results obtained by researchers into the software development environments, so as to increase the energy-awareness of cyber–physical systems developers. In our solution, the energy information is obtained in real-time from a repository of energy consuming concerns, where researchers store their experimental measurements. Developers use the repository to perform sustainability analyses, which, in turn, will lead to greener design/implementation decisions. In this paper, we illustrate the use of our approach in the context of cyber–physical systems development using both open source environments (e.g. JetBrains IDEs) and proprietary environments (e.g. Waspmote development environment). We experimentally demonstrate that cyber–physical systems can reduce more than 40% of its energy consumption depending on the scenario, reaching approximately 90% in some certain cases.

Keywords: Energy consumption; Cyber–physical systems; Green plugin; HADAS eco-assistant

Energy-Aware Environments for the Development of Green Applications for Cyber-Physical Systems

Daniel-Jesus Munoz, José A. Montenegro, Mónica Pinto, Lidia Fuentes

CAOSD Group, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Andalucía Tech, Málaga, 29071, SPAIN
Email: {danimg,monte,pinto,lff}@lcc.uma.es

Abstract

Cyber-Physical Systems are usually composed by a myriad of battery-powered devices. Therefore, developers should pay attention to the energy consumption of the global system so as not to compromise the system lifetime. There are plenty of experimental studies that give hints about how to reduce the energy consumption. However, this knowledge is not readily available for the software developers of cyber-physical systems. They normally use software development environments that do not provide useful advice about the energy consumption of the software solutions being implemented. In this paper, we propose a *Developer Eco-Assistant* to integrate the experimental results obtained by researchers into the software development environments, so as to increase the energy-awareness of cyber-physical systems developers. In our solution, the energy information is obtained in real-time from a repository of energy consuming concerns, where researchers store their experimental measurements. Developers use the repository to perform sustainability analyses, which, in turn, will lead to greener design/implementation decisions. In this paper, we illustrate the use of our approach in the context of cyber-physical systems development using both open source environments (e.g. JetBrains IDEs) and proprietary environments (e.g. Waspmote development environment). We experimentally demonstrate that cyber-physical systems can reduce more than 40% of its energy consumption depending on the scenario, reaching approximately 90% in some certain cases.

Keywords: Energy Consumption, Cyber-Physical Systems, Green Plugin,

1. Introduction

Energy dissipation is a critical topic, especially in systems with scarce hardware resources [1]. Software systems in themselves do not directly consume energy, rather they affect the energy consumption of the hardware (e.g., microprocessor, memory). As a solution, Green Computing [2] aims to promote energy-aware designs and implementations ensuring the efficient use of the hardware by the software.

Software developers need to be aware of the impact that their design and implementation decisions have on the energy expenditure of their systems [3, 4]. Recently, several experimental studies [5, 6, 7, 8, 9, 10, 11, 12] have been developed with the aim of providing information about the energy consumed by different APIs, programming language frameworks, etc. However, this kind of data is rarely accessible through existing *Integrated Development Environments* (IDEs). As a result, energy information goes unnoticed by software developers, hindering the reuse of this energy efficiency knowledge in their applications [3, 4].

This problem is especially crucial in *Cyber-Physical Systems* (CPSs) , which are composed of a myriad of battery-powered devices, so any energy saving solution can lengthen the CPS lifetime. The motivating scenario is a software developer who is programming a Smart Building system provisioned with different sensors (temperature, CO2, presence sensors), sensor boards (e.g. Waspmote, STM32F4 Discovery Base Board) and computational nodes (e.g. Raspberry Pi, mobile phone, personal computer). The environmental information is collected by the sensors on each floor of the building and sent to a mobile phone for processing. The sensor data communication can be done either using Bluetooth (the floor's operator is in charge of collecting the information) or WiFi (the collected information is sent to the operator's mobile phone via WiFi). Considering that it is sensitive data, communication must be secure. Different types of security are applied in different parts of the system, including sensors, so security can

be classified as a large consuming concern that may have a heavy impact on the global energy expenditure of the system over its lifetime. However, depending on the hardware chosen to deploy each part of the CPS, developers will use different APIs or application frameworks to implement the functionality of the system, including the security part. For instance, there are different encryption libraries available depending on the sensor board, and not every algorithm is offered by all the libraries. Moreover, even if running the same operating system (e.g., Linux), programming language (e.g., C) and encryption library (e.g. WolfSSL [1]), the energy dissipation varies for different hardware (e.g., different versions of Raspberry Pi). This is even more important in CPSs due to their large heterogeneity in hardware, and the implications of that heterogeneity in the energy consumption derived from the software execution in battery-powered devices [13]. Thus, both the hardware and the software selected influence the CPS' energy consumption.

If software developers had appropriate sustainability analysis tools, they would be able to reduce the overall energy consumption of a CPS by selecting the most appropriate software solution for each hardware element present in that CPS. A straightforward solution is to integrate energy information into the development environments. However, due to the high-level of heterogeneity that exists, in both hardware and software, identifying the most energy-efficient configurations is very challenging. So, to address the programming of energy efficient CPSs we should answer the following questions: (1) Which are the large energy consumers and their degree of variability (e.g., security); (2) How many different configurations need to be measured, considering the heterogeneity of CPSs in hardware and software; (3) How should the power information be organised and stored so that it is possible to automatically reason about it, and (4) How can software developers obtain this information during CPSs programming in their preferred IDE?

HADAS [14] is a general-purpose green repository that partially answers

[1] `https://www.wolfssl.com/`

these questions. HADAS is based on the identification and modelling of the variability of functional concerns (e.g., compression, distribution), and also on measuring and reasoning about the energy consumption of different software solutions for those concerns. *Concretely, we focus on those concerns with a huge impact on the energy consumption of a software system, and for which the energy expenditure can be measured and analysed independently of a concrete application. We call these concerns, Energy-Consuming Concerns (ECCs)*

However, HADAS cannot be directly applied in the context of CPSs. In this paper, we extend HADAS to develop our "Developer Eco-Assistant" for CPS, now considering different hardware, operating systems and programming languages. Additionally in this new version, ECC libraries can be chosen depending on the constraints of the selected hardware. As a proof of concept, we present our approach for an open source environment (the JetBrain IDEs), and a proprietary environment (the Waspmote environment), using two different hardware platforms (Android SmartPhones and Waspmote) and two different programming languages (Java and C). We focus on the encryption and communication ECCs. The primary aims of this paper are:

- To illustrate how, using our "Developer Eco-Assistant", CPS software developers can be informed about what concerns can be considered for large consumers and what are the most energy efficient designs and/or implementations, helping them to take greener decisions. This requires extending the HADAS repository to allow storing and reasoning about different hardware, operating systems and programming languages.
- To show that both open source and proprietary environments can easily make use of the sustainability analysis provided by our "Developer Eco-Assistant". This also requires extending the HADAS repository so that the information can be accessed using the different extension mechanisms allowed by these environments.
- To demonstrate that the use of our proposal is worthwhile, by showing that thanks to our "Developer Eco-Assistant" developers can make greener

decisions and considerably reduce the energy consumption of a CPS.

For example, in our motivating case study, our green assistant will inform the developer about the greenest configuration of an encryption algorithm in Java for Android using IntelliJ IDEA (e.g., the AES algorithm) or about the energy consumption of an encryption library for different hardware (e.g., mobile phone or a Raspberry Pi). In the latter case, the developer can decide to use either a smartphone or Raspberry Pi to collect the presence information sent by sensors for each floor of the building, based on the energy consumption information provided by our "Developer Eco-Assistant". The rest of the paper is organised as follows. Firstly, Section 2 describes the background required to understand our proposal. Then, Section 3 presents a general overview of the proposal. The details about the developer eco-assistant and its integration into both open-source and proprietary IDEs are described in Sections 4 and 5 respectively. The experimentation phase is detailed in Section 6. Section 7 evaluates our approach and in Section 8 we discuss the related work. Finally, in Section 9 the conclusions and ongoing work are presented.

2. Background

2.1. Variability Models

While traditional software development deals with only one variant at a time, Software Product Lines (SPLs) consider, from the beginning, many possible variants simultaneously. An SPL engineering method focuses on identifying the common and variable features of a family of products, and is often used both in industry and research. Variability modelling is the discipline of modelling both commonality and variability as part of a formal specific model [15]. A characteristic of a system is called *a feature* in the context of variability models, where the most well known are *feature models* that represent feature dependencies. Variability models, the core to many SPLs engineering methods, specify the constraints that exist among the features of a system. A *feature diagram*

(see Figure 1) is a graphical notation in a tree-form to specify a feature model. A feature model is formally represented by means of parent-child constraints and also cross-tree constraints. Cross-tree constraints occur when the selection of a feature requires or excludes the presence of another feature, or group of features (e.g., a LaTeX document implies .tex files), modelled under a different subtree.

Feature models allow the SPL engineer to generate a product configuration through a feature selection process. Then, when only a subset of variable features is selected (i.e., a *partial configuration*), more than one product is generated. Figure 1 shows an example of a variability model. It is part of the HADAS variability model that depicts the file system logging functionality, but in this section we concentrate on explaining the semantics of the feature model elements, which are also explained in the legend at the top of the figure. In this model we specify that there are many optional concerns such as `Cache`, `Communications`, `Memory`, ..., connected to the `Concern` feature through *optional selections* (represented by discontinuous lines). More than one can be selected in the same configuration (*triangle symbol* labelled with 1..*), one of them being `Memory` (Store in memory) decomposed into two optional designs (only `FileSystem` decomposition is shown in the example). There are different variants of the technology to be used (feature `FileSystemTechnologies` and its children) but only one of them (e.g. `LogBack`) can be used in a concrete configuration (*triangle symbol* labelled with 1..1)). Additionally, as the `Simple` alternative cannot work with XML files, we need to specify a cross-tree constraint between the `Simple` and the `XML` features: `Simple implies not XML` (*square symbol* with discontinuous line). Finally, there are features that represent *variables* of a particular type. When these features are selected, a value of that type must be specified. An example is the `MessageSize` feature of type `Int` that represents the size of the message to be logged, which will vary in different configurations.

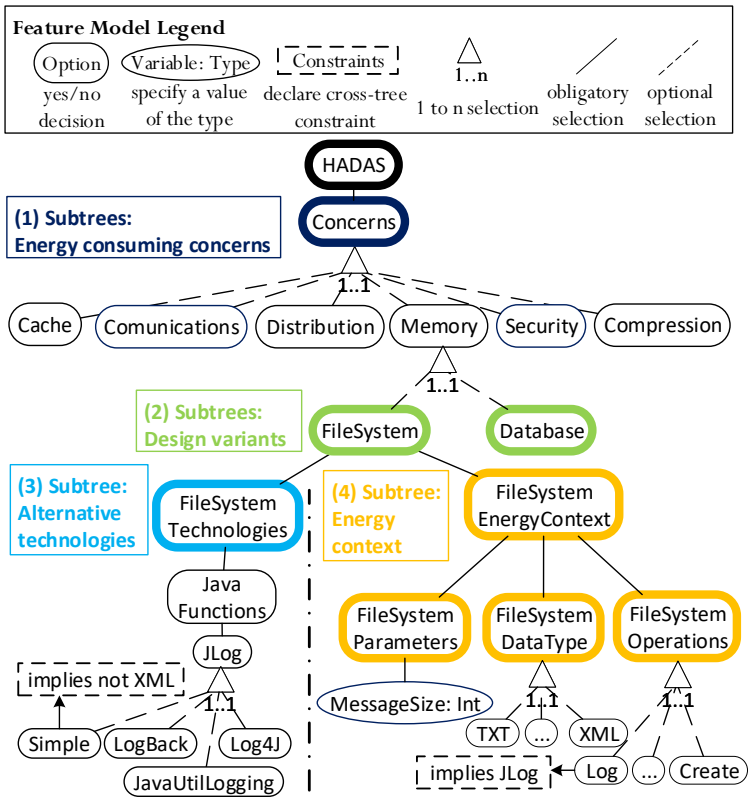

Figure 1: HADAS Variability Model: File System Logging in Java

2.2. Model Reasoners

A variability model and its possible valid configurations, called the search space, are usually verified by means of a Constraint Satisfaction Problem (CSP). CSPs are mathematical statements defined as a set of features whose state must satisfy a number of constraints (limitations) [16]. A CSP *solver* allows: (i) calculating the total number of valid configurations; (ii) automatically generating and reasoning about valid configurations; (iii) guaranteeing the fulfilment of model constraints; and (iv) providing every complete configuration that fits with the declared constraints in a variability model. Clafer [17] is an example of a lightweight modelling language, solver and model reasoner, which includes

the constraint programming library Chocosolver.

In the case of the model in Figure 1, a total of eight configurations can be generated (where the JLog ([`Log4J, LogBack, JavaUtilLogging, Simple`]) and the DataType ([`TXT, XML`]) can vary). As a constraint states that the `Simple` alternative cannot work with XML files, the total number of *valid* configurations is seven, exactly the number of valid configurations that Clafer with Chocosolver will generate. Three examples of complete valid configurations generated are:

1. Memory, Filesystem, FileSystem Technologies[...], Log4j, MessageSize = 100 Bytes, TXT, Log.
2. Memory, Filesystem, FileSystem Technologies[...], Log4j, MessageSize = 100 Bytes, XML, Log.
3. Memory, Filesystem, FileSystem Technologies[...], Simple, MessageSize = 100 Bytes, TXT, Log.

2.3. The HADAS repository

As it is hard for developers to search, read, comprehend and make use of energy consumption raw results, we have developed the HADAS repository. The HADAS repository is a service which stores the results of experimental studies in a common energy-related repository with sustainability analysis support. To specify the software variants that can affect power consumption either positively or negatively, we have developed the HADAS variability model [14]. HADAS makes use of the Clafer modelling language and solver to: specify the HADAS variability model, validate the model, and obtain every possible valid complete configuration. The HADAS repository technically contains a database that connects the 'leaf' nodes of every possible valid complete configuration of the HADAS variability model with their respective energy and time measurements, and some meta-data (a collection of still important information but not subject to 'reason' like the measurement tool, the research group, etc.). To link the database to the *solver* we have developed a PHP automatic query system that translates complete valid configurations of Clafer and Chocosolver into relational database SQL queries.

The HADAS variability model has a well-defined structure, where each level of the model is a specific purpose subtree (see Figure 1). The first subtree (Navy Blue colour) models the *ECCs* or *Concerns*, being six for the moment (`Cache`, `Communication`, ...). The second subtree (or more if they are required) model the design decision variants of a concern (in green). In Figure 1 the `Memory` ECC is decomposed in two design variants, a `FileSystem` and a `Database`. For each design alternative we define two nodes: the *alternative technologies* subtree that models the APIs or framework variants and the *energy context* subtree that models the elements that influence in the energy consumption of a given concern (e.g., the `Log` method that sends a log message).

The energy context feature has three child features: the Parameters (as `MessageSize`), the Data Type (`TXT`, `XML`...) and the Operations to perform (`Log`, `Create`, ...). Since the initial purpose of HADAS was to discover the green best practices for programming crosscutting concerns in Java for desktop computers, it does not consider different programming languages or hardware. So, HADAS has to be extended to be useful for a CPS context, i.e. heterogeneous systems in hardware and software.

3. Approach Overview

Figure 2 depicts the four main parts of our approach, the "Developer Eco-Assistant": (part 1) we extend HADAS to model not only the ECCs, but also the heterogeneity of devices, operating systems and programming languages (HADAS4CPS); (part 2) we collect energy consumption information for heterogeneous devices; (part 3) we import the energy data into an energy-efficient repository and provide this information to other environments as a web service; (part 4) we develop solutions to export this energy information for general purpose and proprietary IDEs.

Firstly (see part 1), we have to identify the larger consuming concerns of CPSs, whose energy consumption we would like to reduce. We focus on those concerns that are recurrent, so that developers have plenty of opportunities

to benefit from the sustainability information provided by our "Developer Eco-Assistant". Once the concerns to be included in our approach have been decided (e.g., Security), we then have to update the variability model, which is the formal base of our approach (Part 1). Due to the inherent heterogeneity of CPSs, HADAS4CPS was defined to model the variability of large consumers (ECCs), but considering differences in hardware, operating systems and programming languages. ECCs are usually implemented by complex and configurable APIs in different programming languages, so all these API variants are included as part of the variability model. For instance, for the Android security API there are different security providers, i.e. there are different implementations of the same security primitives; key generation algorithms can be used with different key sizes, and cipher algorithms can be used with different modes, padding and key sizes [18]. Then, using a CSP solver it is possible to generate configurations from the variability model, adapted to the devices where we will deploy our system. So, depending on the case study we will need to measure energy consumption for certain system configurations, which are automatically generated by the solver. For example, one configuration will tell us to measure the energy expenditure of the RSA algorithm for a key length of 1024 bytes using the SC provider implemented in Java for Android. So, the variability model also helps design the energy expenditure experiments.

Secondly (in part 2), the experiments to measure the energy consumption of one ECC implementation (an API or a framework) will be setup and performed by a researcher on green computing. In the context of CPSs, the same API may have different energy footprint for different hardware platforms. The energy footprint will also be different for equivalent APIs for different operating systems and programming languages. Then it is possible that for a given CPS we would need to measure the energy consumption for different implementations of the same ECC (e.g., AES encryption algorithm) according to the hardware (e.g., mobile phone), operating system (e.g., Android) or programming language (e.g., Java) of each of the devices (e.g., sensors, mobile phones, etc.). The experimental data is obtained differently for each node of the CPS, using the measurement

mechanisms available for each of them. For instance, the PowerTutor application [19] can be used to measure energy consumption of Android applications, the Watts Up? Pro [2] device is used for personal computers and Raspberry Pi, and specific shields are used for Waspmote [20] and the STM32F4 platform [3]. Running these experiments usually takes a lot of time and effort, but with our approach the output of these experiments will be available to developers as part of an IDE.

Thirdly (in part 3) the large amount of information gathered during the experimentation phase has to be formatted and stored in a repository, so that it can be easily accessed. Experimental data obtained by other researches should also be accessible for reusing. In this paper, we *integrate the experimental data* into the HADAS repository [14], which allows us to store and reason about the energy consumption of different ECCs (e.g. security and communication). The HADAS repository (energy-efficient repository) was also extended to take into account that the development of a CPS implies the use of different hardware platforms, operating systems and programming languages. Moreover, the information stored in this repository for green computing can be accessed by a general-purpose web service that any developer can use to perform a sustainability analysis of ECCs. However, to import this data into an IDE we need to provide a new mechanism, a micro-service, which is a specific web service where the variability model is partially configured depending on the characteristics of the CPS under treatment.

Finally in part 4, the energy consumption information should be presented to software developers straightforwardly, while they are coding their applications. Due to the heterogeneity of CPSs' different IDEs are used, so we need to develop several green plugins. These plugins are the entry point to access the "Developer Eco-Assistant" for the IDEs and receive information on the greenest options for one or several ECCs (bottom of Figure 2)). Two main approaches

[2] https://www.vernier.com/files/manuals/wu-pro.pdf
[3] http://mageec.org/wiki/Power_Measurement_Board

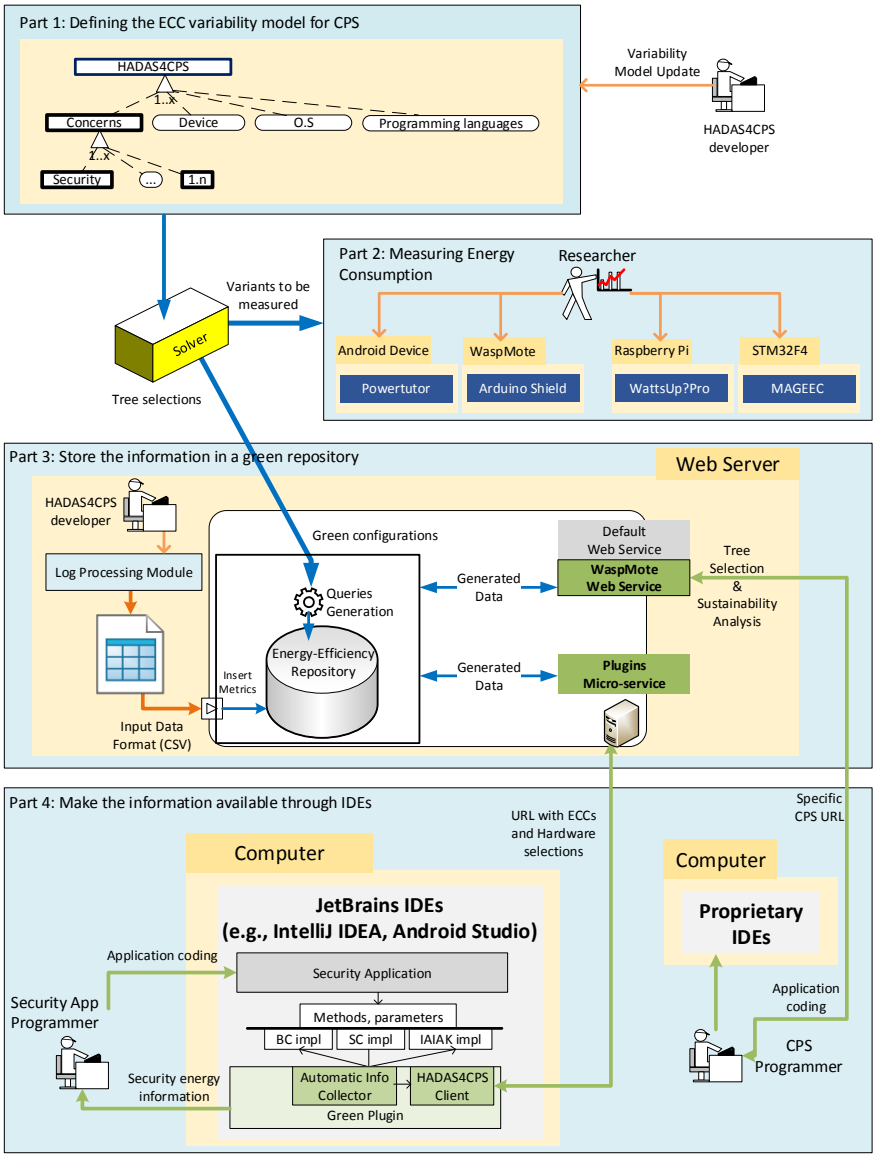

Figure 2: Architecture of the Developer Eco-Assistant

have been identified in the context of CPSs. On the one hand, the development of green plugins for Eclipse and JetBrains IDEs, for those nodes that support

them (e.g. Android devices, Raspberry Pi). However, there are platforms that provide proprietary environments (e.g. Waspmote), which complicates the development of a plugin. In this case, our eco-assistant can still support software developers by providing specific-purpose web services (e.g. the WaspMote Web Service in the middle part of the Figure 2) previously configured to facilitate the use of the energy-efficiency repository in that specific context. Thus, as previously described, the HADAS approach must be extended with a micro-service mechanism allowing open source plugins to request the information stored in the energy-efficiency repository. Moreover, in the case of proprietary environments the "Developer Eco-Assistant" must provide the same information as plugins, but with specific-purpose web services. In the rest of the paper we illustrate our approach in both situations. The development of a green pluging for an open environment such as IntelliJ IDEA for an Android device. We then show how the energy-related information can be consulted when a proprietary environment must be used. In this case we have selected a Waspmote device that is programmed in C.

The extensions to HADAS (parts 1 and 3) are detailed in Section 4, the integration of the "Developer Eco-Assistant" into the software developers' preferred IDEs (part 4) in Section 5 and the experimentation phase (part 2) is described in Section 6.

4. The Developer Eco-Assistant for CPSs

In this section we will provide more details about parts 1 and 3 of our approach, depicted in Figure 2. We focus on the new contributions incorporated to the HADAS repository to obtain HADAS4CPS.

4.1. Part 1: The HADAS4CPS variability model

Some functionalities are usually present in the programming code being executed very often and/or demanding a larger number of CPU cycles than others. So they are considered large energy consumers in our approach. The variability space of the energy consumers is represented under the ECCs nodes (e.g., Security, Communications, Compression, etc.). The different implementations of the

ECCs have to be formally specified to allow reasoning among the different valid configurations of the software system. To represent the search space of the ECCs in the context of CPSs, we have extended the HADAS variability model [14]. The new variability model will allow CPS variants and alternatives (i.e., CPSs heterogeneity) to be formally modelled, matching the different options that a CPS developer has for designing and implementing different functionalities in different embedded and smart devices.

The heterogeneity and variability of modern embedded, smart and IoT devices need to be studied to obtain insights into the energy consumption of CPSs [13, 21]. As most CPS devices are battery powered and have a limited energy budget, it is necessary to consider power consumption of software during CPS development. This is exacerbated by adding power-hungry sensors like WiFi and Bluetooth. The heterogeneity comprises: different mobile 'System on Chip' or SoCs (DSP, GPU, ...) and co-processors (like ARM's big LITTLE architecture) of similar devices (e.g., Smartphones), different commercial versions of a device (e.g., Waspmote v1.2), and different internal versions under the same brand (changing CPU or sensors due to the lack of stock or economic reasons as STM32-Discovery boards). On top of the *Device* heterogeneity, there exists the variability space in the *Operating System* that can run in the device (e.g., specific firmware, FreeRTOS, Android v8), and the *Programming Language* used to program it (e.g., Java, C). Above all, there is still the traditional variability of ECCs.

The new variability model of HADAS extends the ECCs modelling system (just considering Java and desktop applications) to additionally cope with the heterogeneity of *Devices* (hardware), *Operating Systems* and *Programming Languages*. To simplify the modelling of *Devices*, as this can be quite complicated even for mid-experts to exactly describe the device (mostly when dealing with internal versions), just commercial naming will be modelled (e.g., Waspmote 1.2, Nexus One). If necessary CPS energy researchers can add additional information (such as the name of a specific sensor regarding a company internal version of the device), which will be included in the corresponding meta-data

of that specific valid configuration. In other words, internal versions' heterogeneity will not be subject to variations. This is not critical to the service that HADAS4CPS is intended to offer, as our main objective is not to state the exact energy consumption (as it can vary slightly, for example, depending on the room temperature), but rather to compare the energy and time consumption of different complete solutions, mostly when notable differences exist (e.g., using an ultra-low power version of an embedded device, as the STM32F0+ is an ultra-low power version of the STM32F0). As new hardware versions are constantly coming out, the variability model can be easily maintained by adding new features to the tree, i.e., adding a new node (feature) per commercial evolution (Wasptome 1.5, Android v5), or, in case of internal company versions, just indicate it in the meta-data of each measurement.

In Figure 3 we can visualise the new variability model representing all the variants explored in our case study. In the first level (in navy blue), we have now four main subtrees: the `ECCs` or `Concerns`, alongside the three new branches of HADAS4CPS, the `Devices`, the `Operating Systems` (OSs), and the `Programming Languages` (PLs). The `Device` subtree contains all the information with regard to the concrete CPS (e.g., `Waspmote 1.2`). The `Operating System` subtree contains all the information of the concrete OSs, including low-level OSs as proprietary `Firmwares`. The `Programming Language` subtree registers different programming languages, including their variants (e.g., `Java for Android`). As we can see, they have inherent constrains due to the nature of CPS, where, for example, a `Device` like a Nexus One just supports an `Operating System` (Android) and a couple of `Programming Languages` (Java for Android and C). In Figure 3 there are other constrains related to our case study, because the `Waspmote` only supports `C` code running in its proprietary `Firmware`.

As for the ECCs subtree (in red), we have focused on the two that are critical, in energy and performance terms, for our case study: `Communications` and `Security`. For a clearer representation, Figure 3 represents `Communications` with just two nodes, `WiFi` and `Bluetooth`, but in the real model it is modelled following the same subtree structure under `Security`. Regarding the

ECC `Security`, in the variants design subtree (in green) we have focused on `Cryptography`. In the alternatives technologies subtree (in cyan blue) we have considered three different providers for Android (i.e., `IAIK`, `BouncyCastle` and `SpongyCastle`), and the `Libelium` proprietary libraries for the Waspmotes. Of the three different security primitives, we have focused our case study on `Cypher`, which is supported by different algorithms (e.g., `AES`, `RSA`), `Modes` (e.g., `ECB`, `CBC`) and `Paddings` (e.g., `PKCS5`, `ZEROS`). In the energy context subtree (in yellow) we have three subtrees, with two variables (`DataLength` and `KeyLength`), the `Byte` data type, and two operations (`Encrypt` and `Decrypt`).

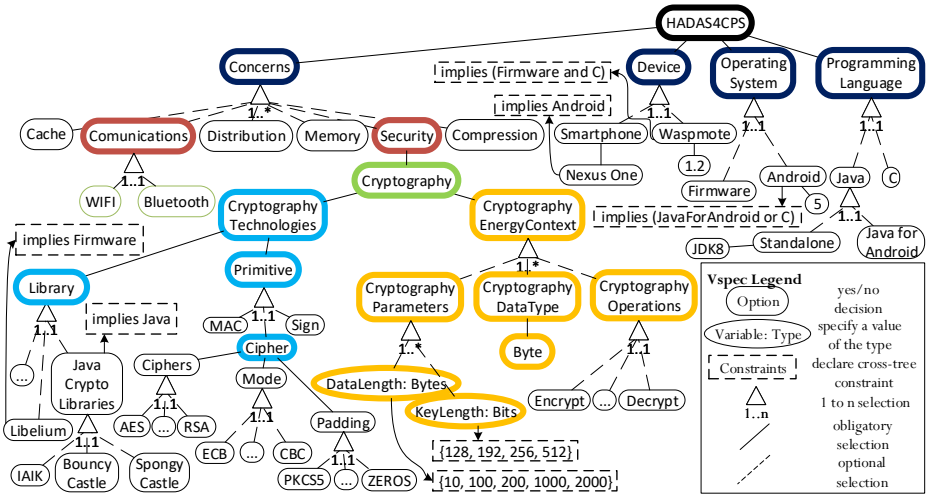

Figure 3: Security Configurations

As previously mentioned in Section 2, the HADAS variability model and now HADAS4CPS are coded in the Clafer modelling language [17], which allows each possible complete valid configuration to be obtained using the Clafer reasoner with Chocosolver. To connect the HADAS4CPS variability model with Clafer and Chocosolver automatically in the eco-assistant, we have developed several PHP scripts. Specifically, for the variability model shown in Figure 3, HADAS4CPS automatically generates 600 complete valid configurations (which

are all the different cases to be measured.)

4.2. Part 3: HADAS Reasoning and Repository

Defining a new HADAS4CPS variability model implies a similar extension of the repository. Note that the structure of the variability model has to be replicated in the database to reduce as far as possible the time spent consulting energy information at runtime. The SQL queries have to be automatically generated when the developer selects specific variants of the variability model. Since the variability model of HADAS4CPS has substantially changed, we have had to modify the internal structure of the database to connect it to this new model. We have also optimized the SQL generator of the former HADAS to cope with the much higher variability introduced by CPS systems. Concretely, we have added three new database tables corresponding to the new subtrees `Device`, `OS`, `PL` to optimize SQL queries. To summarise, the new HADAS4CPS has been built to include variability of CPSs and reasoning capacity about the energy efficiency of the code embedded in lightweight devices. As an example of usage, to reason about our case study (Figure 3), we need to incorporate the CPSs variability of the ECCs *Security* and *Communication*. We add the features that are not already present, like `C` and `Libelium`, to the HADAS4CPS variability model.

At this point, HADAS4CPS can efficiently reason over the 600 configurations of cryptographic primitives implemented for CPSs. During the reasoning, we can include user constraints, such as limiting the `Device`, the `Cipher` and/or the operation that are best for the case study of the CPS developer. After running the solver and the repository automatic process, we obtain the energy and time measurements and meta-data of each complete valid configuration considering system and user constraints. Then, analyses can be performed, as energy and/or time optimisation processes (to obtain the best suitable configuration), graphs generation, statistical procedures to generate advices about specific CPS configurations, etc.

4.3. Part 3: HADAS Micro-Service

Formerly HADAS only offered a web interface (Default Web Service at Part 3 in Figure 2), where the developer could select an ECC to study from a tree view folded form (that represents the variability model as a selectable form).Then the web form containing the user selections was automatically translated (with JavaScript and PHP scripts which we had developed) to additional constraints in Clafer modelling language understandable by the solver (Chocosolver). Next, Chocosolver automatically generated the valid configurations satisfying those constraints. This process ended by executing a PHP script that created automatically optimised SQL queries to extract the proper data from the Energy Efficiency Repository (center of Part 3 in Figure 2). HADAS outputs were generated automatically including bar and line graphs, using Google Charts API, of time and energy consumption. However, for the new HADAS4CPS, we have extended HADAS functionality with a generic micro-service that automatically provides energy efficiency and real-time performance metrics for every desired ECC to be studied. The completely new HADAS4CPS micro-service can be seen as a low-level interface to directly retrieve eco-efficiency data from the latest version of the energy efficiency repository. This is the new mechanism that we have developed to extend the former HADAS service to allow external programs and services to connect with the energy efficiency repository (like a green plugin). The micro-service is accessible through an URL [4], containing nine arrays (one per type of subtree) ready to register the options that the CPS programmer wants to evaluate (e.g. Device = Waspmote v1.2, Algorithm = AES, Key Size = 124, Data Type = Byte, Operation = Decrypt). Once the URL has been requested, the micro-service automatically performs the same described reasoning process, generating each complete valid configuration considering the constraints encoded in the URL (the ECC

[4] `https://hadas.caosd.lcc.uma.es?device[]=value&operatingsystem[]=value&programminglanguage[]=value&concern[]=value&design[]=value&technology[]=value&parameter[]=name=value&datatype[]=value&operation[]=value`

to study), and obtaining the metrics as a response to the automatic SQL query generated by our PHP process. The micro-service automatically returns the complete valid configurations, alongside its metrics (Joules and Seconds) and meta-data in a JSON register with the following structure. The structure of the JSON register is similar to the nine subtree division of HADAS4CPS model (accounting for CPS heterogeneity). Under the energy context of the variability model it is possible to specify parameters of a variable range (e.g., integer), but this exponentially increases the number of possible configurations, drastically slowing down the queries execution. So, for now we allow only one totally variable parameter in a single selection (others need to have an explicit value):

Listing 1: HADAS micro-service JSON output structure

```
HADAS_JSON={
  "#":{
      "Device_A":{
          "Operating System_B":{
              "Programming Language_C":{
                  "Concern_D":{
                          "Design_E":{
                              "Technologies":{
                                  "T1": "Technology_F"
                                  "T2": "Technology_G"
                              }
                              "Fixed Parameters":{
                                  "P1": "Parameter_H = ..."
                                  "P2": "Parameter_I = ..."
                              }
                              "Datatype": "Datatype_J"
                              "Operation": "Operation_K"
                              "Meta": "Data"
                              "Variable Parameter= ...":{
                                "Joules": "Value"
                                "Seconds": "Value"
                              }
                              "Variable Parameter= ...":{
                                "Joules": "Value"
                                "Seconds": "Value"
                              }
                              ...
}...}...}...}...}...}...}
```

As this micro-service interface is quite simple and compatible (due to the standard JSON format), it can easily bring energy and performance information support through plugins, extensions and modules. In this paper, we validate the micro-service with a Green Plugin for JetBrains IDEs and Java code 5.

4.4. Part 3: HADAS Specific-Purpose Web-Services

A particularity of CPSs is that they are usually commercially bundled into a package of devices, proprietary libraries, and proprietary IDEs, which facilitates the programming and the deployment of the code. Consequently, developing an extension to their proprietary IDEs and parsing the unique functions of their proprietary libraries (uncommon naming) are, practically speaking, impossible and mostly useless tasks. To address this limitation, we have recently developed in HADAS4CPS' specific web-interface services accessed through a CPS-specific URL, where the variability/heterogeneity is already restricted to a concrete family of devices (e.g., Waspmotes). To validate this solution, and, as Waspmote is one of the CPS devices that is proprietary, and one which we are familiar with, we have developed a Waspmote web-interface. Waspmote developers can access their intrinsic variability, configurations, metrics from our experimentation, and HADAS4CPS sustainability analyses, without dropping the advantages of Libelium IDE and Libelium C libraries. Waspmote variability model is a partially instantiated variability model, accessed through its URL [5], allowing the developer's desired use-case to be restricted throughout a reduced HADAS4CPS tree-folded form as shown in Figure 4. The sustainability analysis is the same automatic procedure as the default interface [14], including energy and time line and bar graphs, which are plotted with Google Charts technologies, with real outputs as is shown in Figures 11 and 12.

5. Integrating the Developer Eco-Assistant into IDEs

Both open-source and proprietary IDEs may be used in a CPS depending on the hardware. As a proof of concept, we have followed two different approaches, to make our Developer Eco-Assistant accessible to developers using any of the JetBrains IDEs, and also to developers using proprietary environments.

The integration of our Developer Eco-Assistant into an open-source IDE

[5] `https://hadas.caosd.lcc.uma.es/waspmote/`

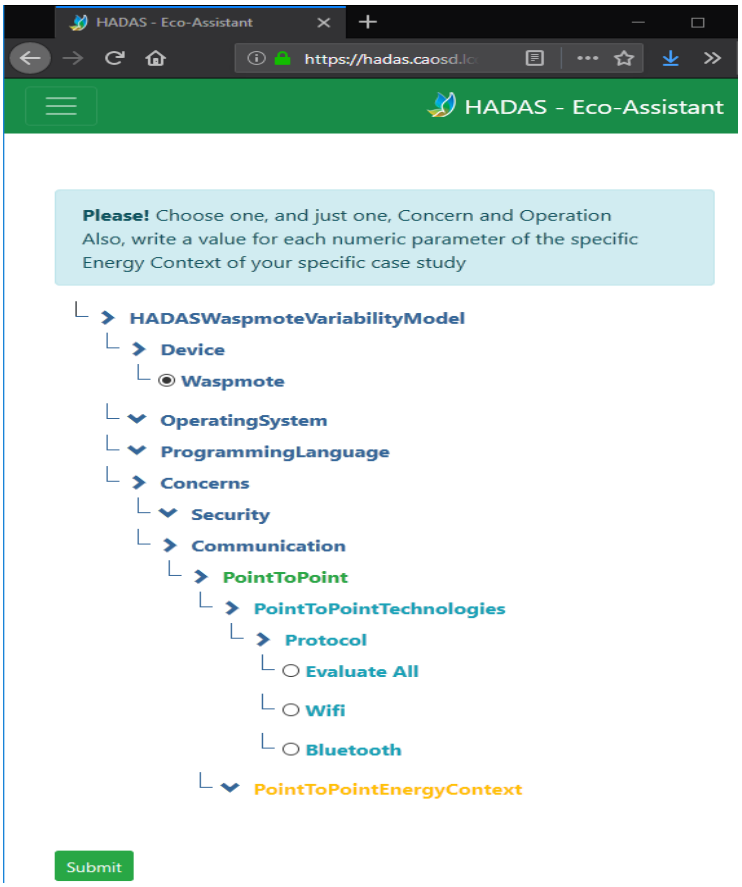

Figure 4: Tree-form view of HADAS Waspmote web-service

implies the development of a plugin. A plugin can be defined as a software mechanism to extend the functionality of a system, in our case, an IDE. In a previous paper [22], we developed a prototype version of our plugin for Android Studio, an Android bundled version of IntelliJ IDEA Community. In this paper we extend the scope of the plugin to support not only Android applications, but also CPS applications using the ECCs identified in the eco-assistant. We have addressed a single plugin compatible with all JetBrains IDEs, due to the high market share of these IDEs (including CPS programming) [6]. As our case study

[6] https://www.g2crowd.com/categories/integrated-development-environment-ide

deals with smartphones and Java compatible devices, we focus on the Java programming language and the Java specific Jetbrains IDE IntelliJ IDEA. So, the Developer Eco-Assistant has been integrated as a plugin for the JetBrains IDEs that acts as a client, where the plugin connects to the HADAS4CPS new microservice (Subsection 4.3). The plugin can be downloaded and installed in any of the JetBrains IDEs [7]. In our case study, as we are decrypting with Android smartphones the data sent by CPS devices, the ECCs to analyse are Security and Communication (see menu bar in Figure 5). The plugin analyses source code searching for these ECC features (like Android OS, encryption functions, key lengths, libraries) and reports energy consumption hints to CPS developers. Specifically, the plugin creates direct-accesses to the concerns selection at the IDE menu bar (e.g., Security, Communication). After opening a code to edit, and clicking on ECC in the IDE menu, the plugin will automatically perform a code parse based on previously extracted information of the methods/functions of that ECC, based on the ECCs API definition and common implementations.

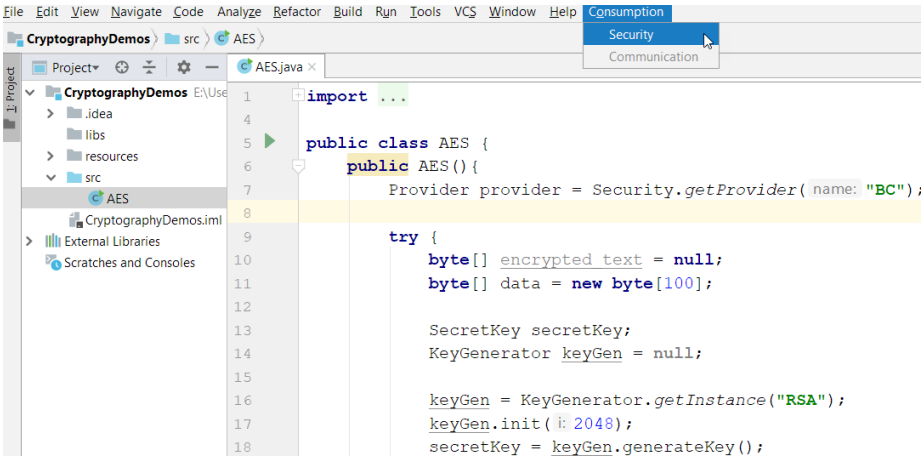

Figure 5: Plugin capture: selecting the ECC

The parsing functionalities of the plugin have been developed using the func-

[7] https://hadas.caosd.lcc.uma.es/jb_green_plugin/

tionality of JetBrains IDEs themselves, the JetBrains Program Structure Interface (PSI) [8]. PSI is a layer of the IntelliJ Platform responsible for parsing files and creating the syntactic and semantic code model of applications. For example, using PSI it is possible to capture all the program variable elements and to store them in an array of *psiElements* for further processing:

Listing 2: Jetbrains parse PSI Java Class example

```
psiVariables = PsiTreeUtil.collectElements(psiClass, new
    PsiElementFilter() {...}
```

Considering the HADAS4CPS variability model explained in Section 4.1, this model is defined in terms of ECC libraries that implement a known API with standard functions naming. For instance, in the case of security, the three providers studied implement different versions of the same Java Security API. These APIs need to be previously examined to automatically identify the PSI elements related to a concrete ECC, and to know when and how to invoke the HADAS4CPS micro-service. As shown in Figure 6, after the ECC *Security* has been selected, and the lexicographic components automatically parsed, the plugin for our Developer Eco-Assistant opens a window, offering the CPS developers a choice of which parameters and recognised values to select/unselect, and from them, which ones will be sent to the HADAS4CPS micro-service, reasoning about HADAS4CPS variability model with the developer constrains (case study). For instance, from the piece of code shown in Figure 5, the plugin is able to automatically identify that the provider is BouncyCastle (i.e., it is the default one and no other is specified in the `getInstance()` static method call in the `KeyGenerator` class), the cipher is AES as specified in the code, the mode and the padding are the default ones because they are not explicitly specified, the key length is 128 as specified in the code, and the data length is 100 bytes, which is the size of the created buffer. To clarify, as shown in Figure 6, if the developer does not want the *Provider*, *Cipher*, *Key length: 128*

[8] `https://www.jetbrains.org/intellij/sdk/docs/basics/architectural_overview/psi.html`

and *Data length: 100* to vary during his/her case study, he/she will select them, leaving *Mode* and *Padding* un-selected and subject to vary (this implies selecting/unselecting those features in the variability model as shown on the right of Figure 6). After connecting with the HADAS4CPS micro-service through its URL (sub-section 4.3) with the selected constraints, the plugin automatically receives the JSON register with the complete valid configurations calculated by HADAS4CPS with the aforementioned user selected constraints. Then, the plugin automatically parses the JSON register into an array with default Java Classes, calculating four key components for the sustainability suggestions:

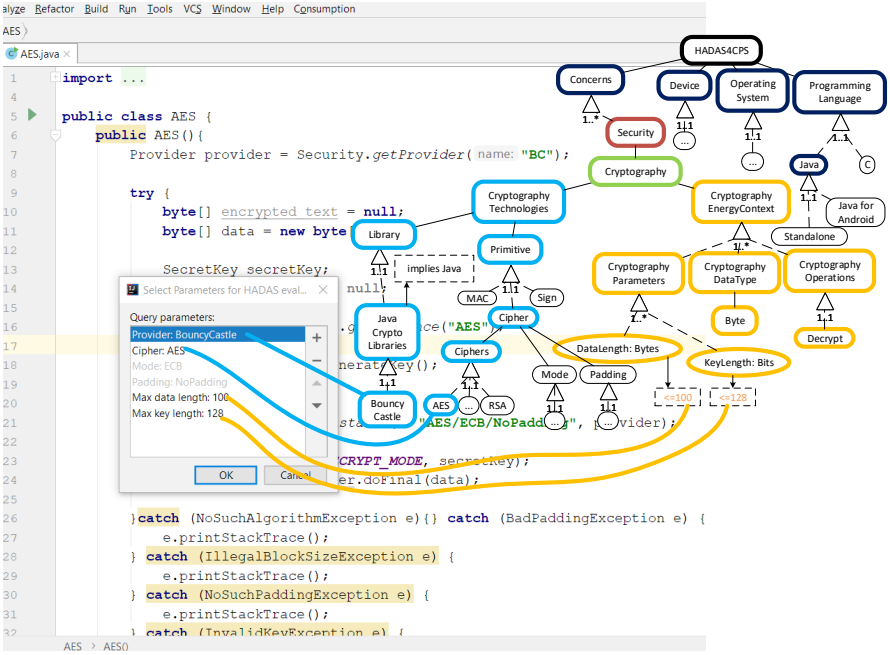

Figure 6: Plugin capture: selecting the case study constraints

1. Consumption in *Watts* for each configuration, dividing Joules by Seconds.
2. The *Greenest solution*, searching for the configurations with the minimum energy consumption (minimising Joules).

3. The *Fastest solution*, searching for the configurations that finish the earliest, with the minimum runtime (minimising Seconds).
4. The *Balanced solution*, searching for the configurations with the minimum power consumption (minimising Watts).

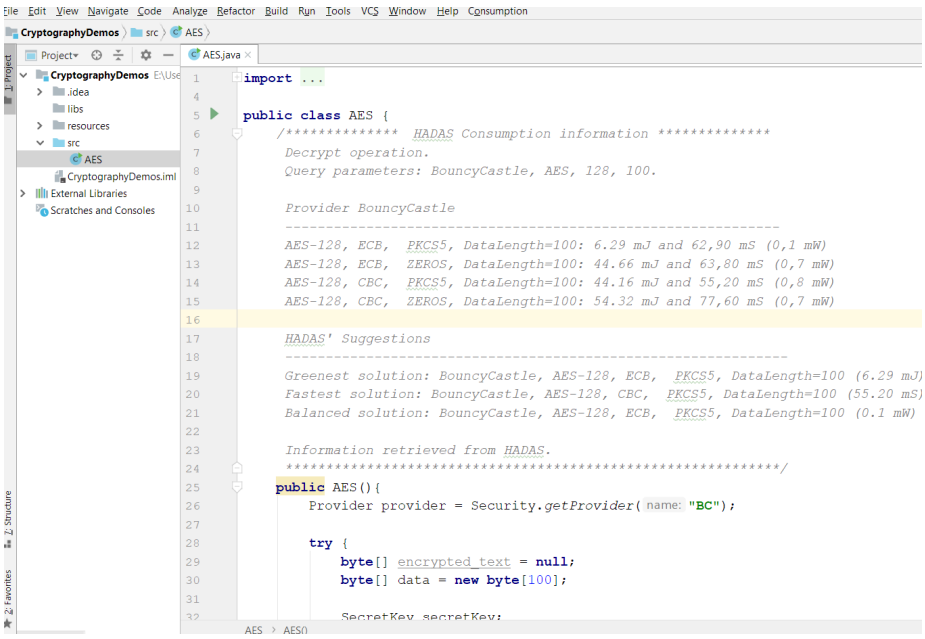

Figure 7: Plugin capture: the sustainability analysis

As shown in Figure 7, each configuration with its consumption, together with the suggestions, are shown as a Java code comment, as it is less intrusive and user-friendly option than pop-ups or new tabs.

The HADAS4CPS micro-service returns a JSON with valid configurations varying the Mode and the Padding, in this example a combinatorial with Modes *ECB* or *CBC*, and Padding *PKCS5* or *ZEROS*. In addition, as the plugin detects the *Java* language, it is constrained in the URL request as a variability model constraint as well. Then, the HADAS4CPS micro-service generates a total of four complete valid configurations and serial of metrics, which are processed by the plugin. The plugin processes the JSON data, and generates a comment

with the data (Figure 7) suggesting that *ECB* Mode and *PKCS5* Padding is the greenest, and also, the most balanced solution, based on the available experiments in the HADAS4CPS repository. In the case of the fastest configuration, the solution is *CBC* Mode and *PKCS5* Padding. Finally, the developer can consider the information suggested, and modify the code based on them, or in the other configurations provided (e.g., he/she wants another type of balanced solution, prioritising run-time, but consuming a bit less).

6. Experimentation Phase

Our CPS motivating scenario includes smartphones and Waspmotes, therefore during the experimentation phase we need to collect and store the energy consumption information of the cited platforms inside the HADAS repository. The ECCs required in our scenario are Communication and Security ECCs, especially cryptography. As a proof of concept, two methods have been deployed to collect consumption information, a software and a hardware profiling. In this section we describe these profiling platforms, while the results are discussed in the evaluation section.

6.1. Software Profiling Platform

For the smartphone device, the energy profiling tool for Android is based on PowerTutor[9] [19], an Android application to profile energy consumption based on the CPU, network interface, display, and GPS elements. The application uses a theoretical energy model built for three phone models: HTC G1, HTC G2 and Nexus One.

As shown in Figure 8, we modified the PowerTutor application to include an Android mechanism for automatically collecting the energy consumption information (BroadCastReceiver). Moreover, a Crypto application has been deployed with all cryptographic primitives detailed in the Android Security API. The goal of this application is to execute the cryptographic primitives between

[9] `http://ziyang.eecs.umich.edu/projects/powertutor/documentation.html`

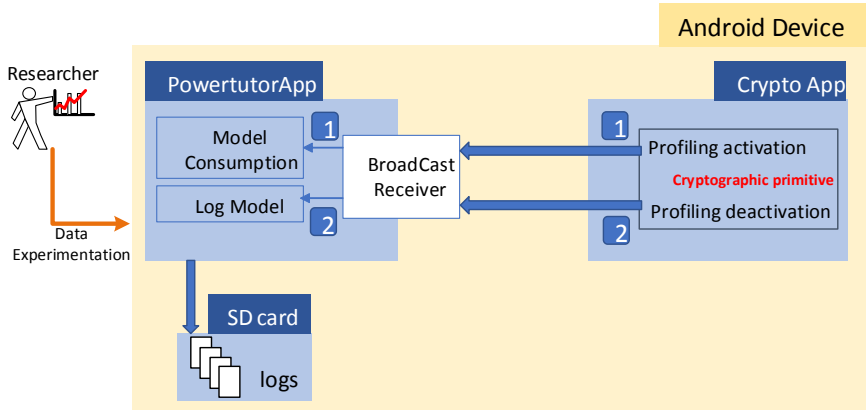

Figure 8: Android Profiling Platform

two commands that activate and deactivate the profiling service of the PowerTutor application. A log file will be stored automatically when the profiling service is deactivated. Therefore, Powertutor does not need to interact with the user, so the data collection is automatic. The generated logs are then processed to provide the information by means of CSV files. These CSV files are the format required to integrate the energy consumption information into the HADAS repository.

6.2. Hardware Profiling Platform

In the case of Waspmotes, the measurement process has been addressed using an Arduino Shield connected to an Arduino [20]. Originally, the Arduino Shield was deployed for real-time systems, robot programming and automatic control subjects, although its design makes it suitable for measuring low energy consumption executions, such as cryptographic primitives and communications with a limited size.

The shield has a 64kB SRAM memory to store acquired-data. Therefore, each execution of an experiment is limited to approximately 1.8 seconds of data records. This limitation has to be taken into consideration when designing

the sketch of the experiment, i.e. connection and disconnection of Bluetooth communication are too time consuming to include in our experiments.

Any hardware device powered by USB can be connected to the Arduino Shield, therefore the shield collects real energy consumption information of the attached device. Since the shield constantly collects the energy information, it is necessary to add a mechanism to determine the consumption of a given procedure. In the rest of this section we describe how to measure the consumption of communication and security procedures in a Waspmote.

Waspmote is a proprietary mote for Wireless Sensor Networks. Waspmote is based on a modular architecture, therefore the developer can only integrate the modules needed in each device. In our experiment, we have used Waspmote PRO v1.2 with ATmega1281 Microcontroller and 8KB SRAM. Additionally, to achieve consumption communications evaluations, WiFi and Bluetooth modules have been plugged-in to socket0 on the board.

Waspmote has eight digital outputs in the sensor connector pins. Figure 9 shows the Waspmote connected to the Arduino Shield. We have established two connections between Waspmote and the Arduino Shield, the USB connector to feed the WaspMote, and the green wired to exchange the signal control. The signal control is used to advise when the Arduino Shield must start and stop the measurement process. If the developer wants to measure the consumption of a code executed in the mote, the following C code must be executed in the Waspmote.

Listing 3: Example of primitive execution

```
digitalWrite(DIGITAL1,HIGH);
  <Code to measure>
digitalWrite(DIGITAL1,LOW);
```

Previously, in the setup section of the Waspmote code, digital output 1 has been chosen as the output signal control, and the green wire is placed accordingly as shown in Figure 9.

Listing 4: Configuration of digital one pin as output

```
pinMode(DIGITAL1,OUTPUT);
```

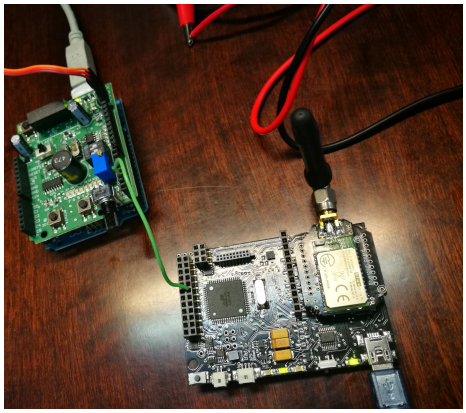

Figure 9: Bluetooth measurement

Figure 10 shows an example of the consumption of an AES primitive. The control signal, the blue line, has two states, zero and one. The control signal is set to one, *HIGH*, when the cryptographic primitive is executed. The time consumed is calculated using the start and end point of control signal. Moreover, we have measured the time consumption of the procedure in the software and the results indicate that the Arduino Shield behaves as expected.Furthermore, the energy consumed for the code is the derivatives of consumption function between the initial and final point of the control signal.

7. Evaluation

This section discusses the experimental data obtained for Android devices and Waspmote, as well as the data validity and the scalability of HADAS4CPS. Moreover, a survey has been conducted among software developers to evaluate our approach in terms of the plugin's usefulness and usability. Finally, other relevant issues are also discussed.

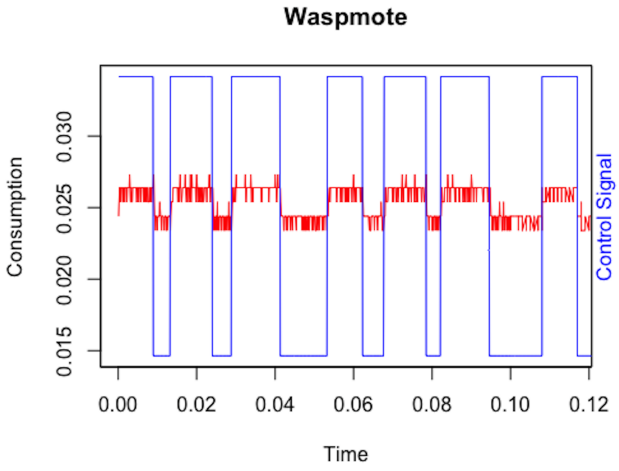

Figure 10: Waspmote measurement data example

7.1. Measure Scenarios: Cryptography and Communication

This subsection focuses on the consumption data obtained for Android devices and Waspmote, applying the experimentation procedures described in Section 6. We focus on the security and communication procedures since they are essential in CPS scenarios. The goal is to show developers what cryptographic procedures and communication methods are the greenest without worsen the system's performance or decreasing its security.

Returning to our motivating scenario, sensor information is collected in the Waspmotes and sent to the mobile phone of the Smart Building operator via Bluetooth or Wifi. This information is stored in the mobile phone and then, at a given moment, it is sent to a central computer for further processing. Examples of the questions that software developers can answer using our approach are: (1) *Which encryption algorithm or encryption configuration is the most energy-efficient;* (2) *Should I sent the data using Bluetooth or WiFi*; (3) *Should I encrypt and send the data as it is received, or is it more energy efficient to store it and encrypt and send bigger data buffers*, or (4) *Are there important differences in the energy consumption of similar solutions depending on the hardware*

platform where the sensors are deployed?

7.1.1. Waspmote Scenario: Cryptography and Communication

Waspmote offers two cryptographic algorithms, symmetric and asymmetric, AES and RSA respectively. We have developed a Waspmote sketch to measure the same configuration (mode, padding) with the three different key lengths.

Evaluations of the symmetric primitive cryptography AES are configured with three different keys (128, 192 and 256), two modes (ECB and CBC) and two padding configuration (PKCS5 and ZERO). Each configuration (key length, mode and padding) is evaluated, encrypting three different length strings: 10, 100 and 200 bytes. Table 1 details the execution time of each configuration of the AES encryption. In this case, the measurement is taken using both software and hardware procedures. The software procedure is performed calling the function *millis*. The *millis* function measures the execution time from when Waspmote is turned on, therefore if we set an instruction between two *millis* calls, we can calculate how many milliseconds the execution of the code under evaluation takes. For the hardware procedure the time is calculated using the Arduino Shield. Specifically the time is the interval between the activation and deactivation of the control signal. At first glance, there is not significant difference between hardware and software measurement, even so we have performed a statistical test that confirms our initial hypothesis.

Table 2 shows the energy consumption of each configuration. This experimental data shows relevant information that software developers can obtain by using our plugins and the HADAS4CPS eco-assistant:

- **For a particular key length, the use of different modes or paddings with AES does not affect energy consumption significantly.** With this information software developers can decide, for instance, to use the most secure configuration without penalizing energy consumption.

- **Energy consumption for the AES algorithm increases if the key length increases.** Although this conclusion may seem obvious, it is

Table 1: Waspmote AES time software and hardware

			Data length (bytes)			Data length (bytes)		
Key Length	**Mode**	**Padding**	**10**	**100**	**200**	**10**	**100**	**200**
			Time Software (ms)			**Time Hardware (ms)**		
128	ECB	PKCS5	1,00	8,60	15,80	1,33	8,98	16,01
		ZEROS	1,00	8,40	15,60	1,30	8,64	16,74
	CBC	PKCS5	1,00	9,00	16,70	1,42	9,00	16,59
		ZEROS	1,00	8,60	16,70	1,44	9,08	16,96
192	ECB	PKCS5	1,00	10,00	18,70	1,65	10,68	19,94
		ZEROS	1,00	10,00	18,60	1,74	10,56	19,32
	CBC	PKCS5	1,00	10,70	20,00	1,82	10,86	19,97
		ZEROS	1,00	10,30	20,00	1,70	10,82	20,00
256	ECB	PKCS5	2,00	12,00	22,00	2,00	12,31	22,05
		ZEROS	2,00	12,00	22,00	2,03	12,31	22,73
	CBC	PKCS5	2,20	12,00	23,20	2,01	12,37	22,82
		ZEROS	2,00	12,20	23,10	2,19	12,47	23,00

important to know how large the increase is before taking the decision of which one to use. The data indicates that the increase of key length is approximately 1.2 times in both cases; from 128 to 192 and from 192 to 256 key lengths. Thus, in the case the developer wants to move from less to the most secure key the consumption difference is around 1.5 times.

- **Energy consumption for the AES algorithm increases if the length of the encrypted data increases, but not proportionally to the data length.** An increase of 10 to 100 bytes implies a 6 times increase in energy consumption, whereas the difference from 100 to 200 bytes translates to around 1.8 times.

Thus, using our approach the sustainability analyses performed by developers allows them to identify that they can assume the energy consumption of the most secure key configuration. Furthermore, it is recommended to encrypt using longer data packages. For instance, using 256 key length, ECB mode and PKCS5 padding, if the developer needs to encrypt 200 bytes, the consumption

will be 0.58 mJ. Otherwise, in the case he/she encrypts two packages of 100 bytes, 0.64 mJ will be consumed. Finally, 1 mJ will be consumed if the developer decides to use 10 bytes packages. This means that **42% of energy is saved when the longest length is used.** In other words, it is preferable to store the data to build higher buffers and then encrypt them, instead of encrypting short buffers. After this analysis, and in case the developer was encrypting short buffers, he/she will change the code to encrypt higher buffers and therefore reduce the energy consumption of the implementation.

Table 2: Waspmote AES consumption hardware

Consumption (milliJoules)			**Data length (bytes)**		
Key Length	**Mode**	**Padding**	**10**	**100**	**200**
128	ECB	PKCS5	0,033	0,23	0,42
		ZEROS	0,034	0,22	0,42
	CBC	PKCS5	0,036	0,24	0,42
		ZEROS	0,037	0,24	0,44
192	ECB	PKCS5	0,041	0,27	0,52
		ZEROS	0,045	0,27	0,49
	CBC	PKCS5	0,046	0,29	0,51
		ZEROS	0,043	0,29	0,52
256	ECB	PKCS5	0,050	0,32	0,58
		ZEROS	0,053	0,31	0,58
	CBC	PKCS5	0,050	0,33	0,58
		ZEROS	0,056	0,33	0,60

Waspmote also supports the RSA algorithm. It provides support for encrypt-decrypt and sign-verify, although no key generation is supported due to high computation requirements. The operation works correctly using 512 length key but its performance is erratic with longer length keys. Table 3 shows the time execution in software and hardware, encrypting two different data lengths. The information provided by the plugin in this case indicates:

- **Energy consumption for the RSA algorithm does not depend on the data length encrypted.** In this case, the conclusion of the

experiment is different than for the AES algorithm since neither the time nor the energy consumption depend on the data length encrypted.

- **There is important difference in energy consumption between AES and RSA.** In this case the comparison would be between the most secure configuration for AES (256 key length, ECB mode and PKCS5 padding) and for RSA (512 key length) since 512 key length is not available for AES. In this case software developer need to know that for cyphering 100 bytes, **using RSA with 512 key length consumes 97% more energy than using AES with 256 key length.**

Indeed, this result illustrates another benefit of using our approach, since the energy footprint tendency is different depending on the algorithm and thus no assumptions can be made before performing the sustainability analyses.

Table 3: Waspmote RSA encryption 512 key length

	Software	**Hardware**	
Data length (bytes)	**Time (mS)**	**Time (mS)**	**Consumption (mJ)**
10	454	455,17	11,48
100	455	454,03	11,26

In our CPS scenario, the data encrypted in the Waspmote is then sent to a mobile phone. Regarding communication, two modules are evaluated, Bluetooth and WiFi. A string with different lengths has been sent between the Waspmote and an Android smartphone in the Bluetooth case, and between the Waspmote and a laptop in the WiFi communication. The Waspmote is configured as a client in both communications and a proof of concept Bluetooth terminal Android application and a Java echo Server have been deployed for Bluetooth and WiFi communications, respectively.

Table 4 details the software and hardware time execution (milliSeconds) and energy consumption (milliJoules) in both communication scenarios. As in previous cases, there are no considerable differences between software and hardware time measurement.

Table 4: Waspmote Communication consumption

	Bluetooth			WiFi		
	Software	**Hardware**		**Software**	**Hardware**	
Data length	**Time**	**Time**	**Cons.**	**Time**	**Time**	**Cons.**
10	100.06	99,42	3,38	3,56	3,10	0,19
100	107,98	107,26	3,62	11,56	10,71	0,65
1000	187,00	186,41	9,65	89,00	89,74	5,75
2000	273,29	273,22	13,28	176,40	176,27	11,43

Figure 11 shows a comparative between the energy consumption of Bluetooth and WiFi modules. In both cases, the establishment phase of the communication is not included in the consumption process, focusing as it does only on the communication phase. In this case, the sustainability analysis that software developers can perform is:

- **The Bluetooth module consumes more energy and time than the WiFi module.** The chart (Figure 11) shows that communication using the Bluetooth module consumes more than using the WiFi module, specially between 10 and 1000 bytes, from 3 to 2 times approximately. In the case of 2000 bytes the difference is not so significant. The same is shown in Figure 12, although the gap between the time consumed is reduced when the transmission data length is increased.

- **If Bluetooth is required, it is almost mandatory to send long buffers.** When sending many short buffers the difference in the energy consumption is extremely high and this can seriously affect the system's energy consumption.

For instance, if a developer needs to send 2000 bytes in a single WiFi connection the energy consumed is 11.43 mJ, as shown in Table 4. In the case other buffer lengths are selected, the consumption will be 38 mJ for 200 packages of 10 bytes, 13 mJ for 20 packages of 100 bytes and 11.5 mJ for 2 packages of 1000 bytes. This means that **70% of energy is saved when the longest**

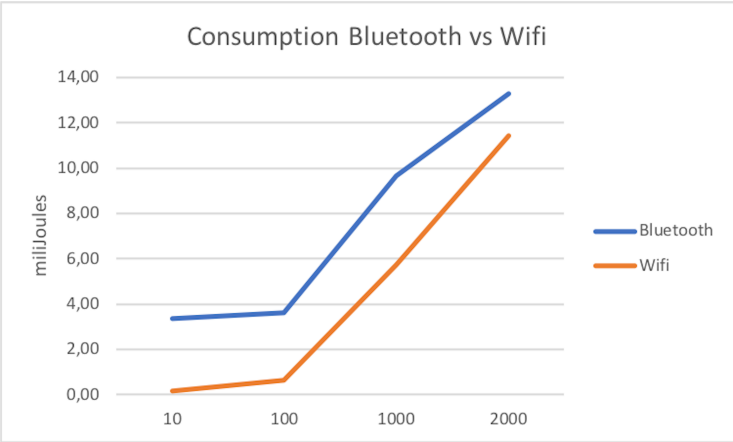

Figure 11: Waspmote Consumption Bluetooth vs Wifi modules

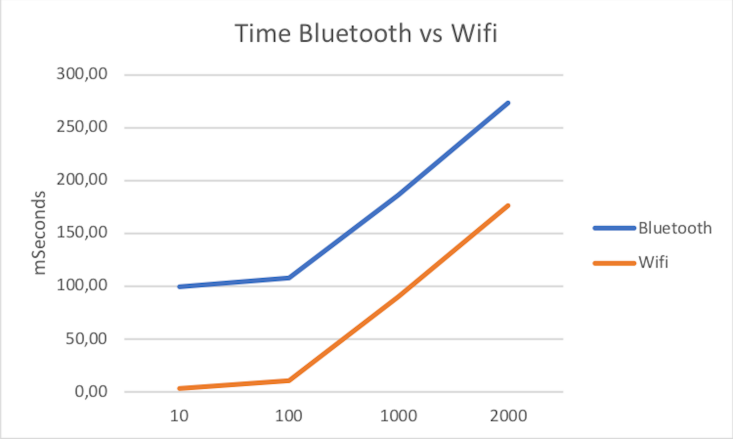

Figure 12: Waspmote Time Bluetooth vs Wifi modules

length is used. Thus, if the developer wishes to send large amounts of data it is preferable to use a long buffer in a single communication than short packages in several transmissions, although the consumption gap between 2 packages of 1000 bytes and one package of 2000 bytes is almost negligible.

The same experiment applied to Bluetooth shows that a communication of 2000 bytes consumed 13.28 mJ. If other buffer lengths are used the increase of energy consumption is extremely high: 676 mJ for 10 bytes buffers, 67.6 mJ for

100 bytes buffers and 19.3 mJ for 1000 bytes length. In this case, the energy consumed decreases considerably when the buffer length is increased (**98% of energy is saved when the longest length is used.**), therefore it is advisable to use the greatest buffer communication in a Bluetooth connection when large amounts of data must be sent.

7.1.2. Android Scenario: Cryptography and Communication

For Android Smartphones we discuss the results of the same experiments performed with Waspmote. Note that the conclusions are not always the same and this demonstrates the necessity of the sustainability analyses offered by our approach:

- **For the AES algorithm, the results are similar** and indicate that using the most secure configuration can be afforded without a considerable increase in energy or time consumption. Moreover, it is always better to encrypt/decrypt large buffers than smaller ones (Table 5 and Table 6).

- **However, for the RSA algorithm the results are different** for the encryption operation. Thus, the time required to encrypt/decrypt and the energy consumed during the decryption operation do not depend on the data length, as happens with Waspmote. However, for the encryption operation, the energy consumption of encrypting 100 bytes almost double the energy required of encrypting 10 bytes.

- **The time and energy consumption of the WiFi module is independent from the data length**, so sending longer buffers is the recommended option.

- **There are considerable differences between Waspmote and Android** on the time and energy that is required to encrypt/decrypt the data, and also, to send the encrypted data through the network (e.g. sending 2000 bytes using the WiFi in Waspmote consumes 11.43 mJ, while sending the same data length using the WiFi in Android consumes 4409,10

mJ). This clearly indicates the importance of using an eco-assistant in the development of CPSs to help decide on the hardware where the different elements of the CPSs is best deployed to save time and energy. In this particular case, we use the encryption and communication libraries provided by Waspmote, when the programming language is C, while in Android the programming language is Java for Android, and we have the performance overhead introduced by the Android Layers, including the Dalvik Virtual Machine.

In addition to AES and RSA, for Android applications there are several security providers with the possibility of using many other encryption algorithms with additional key lengths, modes and paddings. In [18] the reader can find a systematic experimental study that reports about the time and energy consumption of all the possibilities offered by the Android security API. This information is already incorporated inside HADAS4CPS and is accessible to software developers through our plugins.

Table 5: Android AES encryption consumption

ENCRYPT (milliJoules)			**Data length (bytes)**		
			10	**100**	**200**
128	**ECB**	**PKCS5**	18,51	6,29	40,40
		ZERO	14,34	44,66	64,35
	CBC	**PKCS5**	38,08	44,16	49
		ZERO	19,59	54,32	31,48
192	**ECB**	**PKCS5**	6,41	31,25	24,32
		ZERO	0	42,48	37,08
	CBC	**PKCS5**	11,80	20,46	45,90
		ZERO	14,84	44,38	40,08
256	**ECB**	**PKCS5**	7,21	49,44	25,80
		ZERO	7,10	45,52	51,66
	CBC	**PKCS5**	5,72	12,38	42,24
		ZERO	19,35	42,21	24,76

Table 6: Android AES decryption consumption

DECRYPT (milliJoules)			**Data length (bytes)**		
			10	**100**	**200**
128	**ECB**	**PKCS5**	19,44	39,60	36,72
		ZERO	0	31,00	58,68
	CBC	**PKCS5**	25,80	10,94	11,48
		ZERO	22,14	28,40	29,10
192	**ECB**	**PKCS5**	6,15	49,20	45,57
		ZERO	6,94	15,84	32,75
	CBC	**PKCS5**	12,82	37,05	24,56
		ZERO	6,17	39,72	71,80
256	**ECB**	**PKCS5**	12,30	66,40	62,60
		ZERO	5,42	40,38	66,87
	CBC	**PKCS5**	26,52	37,05	44,10
		ZERO	33,30	71,64	32,80

Table 7: Android RSA 512 key length

	Encryption		**Decryption**	
Data length	**Time (mS)**	**Cons. (mJ)**	**Time (mS)**	**Cons. (mJ)**
10	59,00	5,90	76,10	144,59
100	53,60	10,72	77,20	138,96

7.2. Validity of the experimental data

To check the internal validity of our data, we analyse the precision of the results obtained. For the Android smartphones, we have opted for PowerTutor, a software solution, instead of using more precise tools implemented in hardware. We have chosen PowerTutor since the results provided by this tool have been validated in different studies. Although hardware solutions provide higher precision measurements, it is more difficult to be sure of which part of the software executed is responsible for the consumption. To mitigate this, we have replicated the experiments on two different mobile phones obtaining similar results. Moreover, in [18] we performed a statistical analysis of the results, using the Wilcoxon rank sum test[23] and the statistical package R, to discover

Table 8: Android AES encryption time

ENCRYPT (milliSeconds)			**Data length (bytes)**		
			10	**100**	**200**
128	**ECB**	**PKCS5**	61,70	62,90	80,80
		ZERO	71,70	63,80	71,50
	CBC	**PKCS5**	54,40	55,20	49,00
		ZERO	65,30	77,60	78,70
192	**ECB**	**PKCS5**	64,10	62,50	60,80
		ZERO	65,20	70,80	61,80
	CBC	**PKCS5**	59,00	68,20	76,50
		ZERO	74,20	63,40	66,80
256	**ECB**	**PKCS5**	72,10	61,80	64,50
		ZERO	71,00	56,90	73,80
	CBC	**PKCS5**	57,20	61,90	70,40
		ZERO	64,50	60,30	61,90

Table 9: Android AES decryption time

DECRYPT (milliSeconds)			**Data length (bytes)**		
			10	**100**	**200**
128	**ECB**	**PKCS5**	64,80	66,00	61,20
		ZERO	59,20	62,00	65,20
	CBC	**PKCS5**	51,60	54,70	57,40
		ZERO	73,80	71,00	58,20
192	**ECB**	**PKCS5**	61,50	61,50	65,10
		ZERO	69,40	52,80	65,50
	CBC	**PKCS5**	64,10	74,10	61,40
		ZERO	61,70	66,20	71,80
256	**ECB**	**PKCS5**	61,50	66,40	62,60
		ZERO	54,20	67,30	74,30
	CBC	**PKCS5**	66,30	74,10	73,50
		ZERO	66,60	59,70	65,60

Table 10: Android WiFi time and consumption

	Time (mS)	**Consumption(mJ)**
10	133,44	4630,52
100	140,50	4805,10
1000	129,60	4445,28
2000	127,80	4409,10

if the differences in the mean values are significant. To measure the energy consumption in the Waspmote platform, we have used a hardware solution and the comparison using different measurement mechanisms was not possible. To mitigate this, we have replicated all the experiments to measure the time consumption using both, a software and our hardware solution, and no important differences were found. Note that our approach is not intended to report absolute energy values, but rather we wish to give recommendations to developers based on comparative results.

Concerning the external validity, we have identified two threats to validity. First, we have not tested our findings in third party applications. So, we do not consider the cost of invoking the cryptographic algorithms and the communication primitives. We think that by not considering this, we can provide results irrespective of the application that is using these algorithms, which can be stored in a repository such as HADAS4CPS. However, for now, we cannot report on the energy savings of real applications that use our recommendations, which is planned for future work. The second threat concerns the generalisation of the results to other hardware elements of CPSs, such as a Raspberry Pi and an STM32F4 platform, as discussed during the motivation of our approach. This is done incrementally, and the results will be incorporated in the HADAS4CPS repository as soon as the data collection processes end.

7.3. HADAS4CPS Scalability

The scalability tests have been carried out on an Intel (R) Core i7-4790 CPU @ 3.60 GHz processor with 16 GB of memory RAM and an SSD disk under Microsoft Windows 10 x64 in high performance mode. The parts of

Table 11: HADAS4CPS scalability

Nodes Dependency	Configurations	Milliseconds
5	**10**	**4**
5	50	6
10	**10**	**10**
10	200	16
20	**10**	**32**
20	800	54

HADAS4CPS with a high computation load are: Clafer and Chocosolver for the generation of configurations, and the access to the database for generating SQL queries and retrieving measurements. So, we have focused our tests on testing complex configurations (more nodes per configuration that creates longer SQL queries) and large configurations (more SQL queries). We have measured the response time of the micro-service, and improved the database model of previous versions of HADAS4CPS for the high variability and extra features (e.g. Hardware) that CPSs devices have. We have included database tables for Device, Operating System and Programming Language names. In addition, we have solved the regular problem of *Entity-Attribute-Value* [10] with a single column solution for Attribute-Value, which increased the scalability of the HADAS4CPS micro-service, lowering the complexity of the queries and runtime dependency from exponential to linear, with improvements of at least a 100% [14]. Table 11 is intended to benchmark the spectrum of the analysis that the plugin could realise in sufficient time. Regarding the number of *Configurations* to compare, we have set 800 as the maximum even if it is very unlikely to happen (probably the developer will constraint more the variability). For any of the cases, the response times are extremely low (less than 0.054 seconds), even for high *Nodes Dependency*. In general, as noted in Table 11, the *Nodes*

[10] `http://mysql.rjweb.org/doc.php/eav`

Dependency is the critical variable in performance, rather than the number of queries, so because the data depends on more technologies and more parameters, higher times can be observed. However, with the new database model, the HADAS4CPS repository can cope with highly complex scenarios with lots of data, with response times of lower than a second. Regarding data bandwidth, the URL inputs and JSON outputs are small for a web micro-service, so it does not affect the performance of the whole plugin system, even for slow connections. Summarising, HADAS4CPS scales pretty well, and suffices for our case study with low response times.

Regarding the HADAS4CPS variability approach, considering how large the number of total possible valid configurations (search space) can get in CPS case studies, we need a procedure to reduce the number of experiments (measurements). Our next step is to apply the SPL technique called *Random Sampling* [24], for feature models with large search spaces greatly reducing the number of experiments. In general terms, we identify the features that truly induce great variations of consumption, and use them to classify the configurations into groups with low variation of consumption, needing just one measurement per group.

Regarding the plugin, even though there is an overhead in JetBrains IDEs, it is almost negligible. It just runs on the IDE, which implies that every software developed is unaffected by our plugin theoretical consumption.

7.4. CPS developers' experience

We have evaluated our approach in terms of usefulness and usability of the plugin. A total of 17 participants were asked to participate in a real evaluation; to code their efficient Java software in IntelliJ IDEA while making use of the plugin. After this evaluation, they completed a survey based on general questions about their gender and age, programming skills, sustainability interest and technical opinion about the plugin results. The survey has been devel-

oped with an instance of LimeSurvey hosted in the Universidad de Málaga [11]. The specific results and statistics can be consulted at [12]. Regarding population profile, the average age was 32 for both genders. The education level varied between bachelor's degree and full professor, all of them employed and equally distributed between academia and the private sector. The respondents were concerned about the Earth's sustainability (Likert-scale 3.94/5), they mostly had not heard about energy-aware software engineering, and mostly agreed on the usefulness of getting energy consumption information in their preferred IDEs (Likert-scale 3.88/5). Most of the respondents had worked in more than three large projects and were familiar with several programming languages as Java, C, Python and web technologies, as well as with their famous IDEs (as the JetBrains IDEs), and had been professionally programming over five years (one of them more than twenty). They had all programmed lightweight devices with smartphones, Raspberries and alike, and sensor boards (as Waspmotes) being the most common answers. Regarding the IntelliJ IDEA plugin, most of the questions were Likert-scales to evaluate different aspects of the approach and the plugin. Average results of the Likert-scale questions are depicted in Table 12 where the least positive answer is 1, with 5 being the most positive. CPS developers had not considered a solution like our plugin before (0.4/5), they found the plugin functionality somewhat user-friendly (3.6/5), considering the information provided as relatively easy to understand (3.9/5). The interface was considered consistent (3.9/5) and the prompt to constraint the variability of the alternatives clear (3.9/5). Even if they considered the usability of the plugin as complex (4.1/5), they would be likely to recommend it (3.8/5) and make use of it (3.8/5). Regarding the output, they preferred a list of all the configurations and their consumption highlighting the most eco-efficient/fast/balanced solutions. Suggestions to improve the plugin were:

- To present the date time of the execution in the comment.

[11] https://encuestas.uma.es/index.php?sid=92571&newtest=Y&lang=en
[12] https://hadas.caosd.lcc.uma.es/survey2018.pdf

Table 12: Results of the IntelliJ IDEA plugin for the HADAS4CPS approach survey

Likert-scale question	Average
1. The developers have considered a similar solution before	0.4
2. The plugin is user-friendly	3.6
3. The information provided by the plugin is understandable	3.9
4. The plugin interface is consistent with the IDE	3.9
5. The prompt to constraint the variability is clear	3.9
6. The plugin is complex	4.1
7. The developers will recommend the plugin	3.8
8. The developers will make use of the plugin	3.8

- To have a pop-up instead of a comment.
- To have an option to generate reports in CSV files.
- To have an option 'Copy to clipboard'.
- To have an option 'More details' with statistical information.
- To design a more user-friendly table of results.
- To show graphs in the IDE.
- To release it with C/C++ measurements/suggestions.
- To include all the metrics in the optimised solutions (e.g., not just Seconds for the fastest solution) .
- To include checkboxes in the prompt, instead of '+' and '-' buttons.

7.5. Discussion

In this section, we discuss other issues that are relevant to our approach. Firstly, the integration of research measured values into the HADAS4CPS repository is not a trivial task, so a measurements-log processing module has to be customised, to translate the metrics, taken by different researchers, to the format

requested by HADAS4CPS. Besides, for Waspmote and the Android measures, we have automatised most of the process for generating the HADAS4CPS input CSV files into the format required to be able to map the information to the new CPS variability model structure. Secondly, the HADAS4CPS micro-service is accessed by an URL that contains arrays that have to be translated to constraints in the variability model. In this case, HADAS4CPS needs to be robust enough to detect any invalid value. Otherwise, when an invalid constraint is provided to Clafer, the variability reasoner will not work properly and the micro-service will not return the energy consumption information. This problem did not happen in the HADAS general-purpose web service or in the specific-purpose web service developed for Waspmote. The web form provided to the users was automatically generated from the information contained in the tree so that users could not introduce invalid values.

Finally, for this paper we have focused on energy consumption, but as shown in the experimental data we could also cover performance and trade-off (energy and time), as the micro-service already provides the necessary data. There could also be a discussion of whether these plugins add a security concern in themselves, but, in any case, there are just partial configurations and raw energy data transactions at programming time.

8. Related Work

We have divided the related work section into the topics: greener software for CPSs (focusing in Security), CPSs heterogeneity and energy consumption, ideas to solve it as our Green Plugins, and eco-assistant and repository efforts valid for external IDE solutions to obtain energy insights.

Eco-efficiency is a key factor in CPSs since it determines the autonomy of the device. However, CPSs developers tend to underestimate the energy consumption of their CPS devices at a software level [25]. For instance, in [26] the authors propose a green energy-powered CPS architecture based on a service composition in the context of fog computing. However the usage context and the insights

for the deployed functionality are omitted. In [27], the context is considered to increase the lifetime by reducing the amount of data transmission in CPS in the cloud. However, they solely focus on communication, while our approach, also tested for security, with a broader scope, suitable for any ECC and CPS existing configuration. The goal of setting-up a usable high-performance environment is treated in [28] to investigate complex energy CPSs with application scenarios that served as test cases, however, software simulations where used, instead of real devices and measurements. The multi-core processors CPS capabilities and energy consumption of multi-core algorithms are studies in [29]. However, the insights gained are of a very low-level code abstraction, which makes it hard to implement in real-world CPS scenarios. In [30] a framework for CPS development is formalised. They evaluate the effectiveness of their proposal in terms of availability, downtime, downtime cost and reliability of the CPS, however energy consumption is not treated. In Embedded Systems, the total energy consumption is minimised while the timing constraint is satisfied in [31]. In addition, they propose a heuristic approach to efficiently obtain a near-optimal solution, however it is only useful for hybrid memory systems. With respect to security, only the work in [32] takes into consideration that the algorithms can work in different modes (e.g. ECB, CBC, CFB, OFB, CTR and CCM). The main differences between this approach and our work, it is that our experiments take into consideration all the parameters that may influence the energy consumption of cryptographic primitives (i.e., different crypto providers, different algorithms and operations, different key lengths, different modes and different paddings). An approach with similar goals to ours, but focusing on performance evaluation instead of energy-consumption evaluation, is found in [33].

In the paper [13] the authors study how to gain advantage of the heterogeneity of CPSs to minimise energy consumption. The authors demonstrate that the energy consumption is minimised by relying on smartwatch calculation and sensors during a Bluetooth connection between our smartphone and our smartwatch. The basis is that a CPU or a GPS of an smartwatch, while obtaining the same results in a similar time-frame, consume less. The consumption

is minimised even further if the smartwatch can collect data by itself, dumping the data to the smartphone later on (the smartphone may have been switched off). In the study [21], the authors combine load balancing, energy efficiency, hardware heterogeneity and application heterogeneity in heterogeneous and distributed embedded systems. They propose an efficient algorithm to solve the joint optimisation problem using the Lagrange method. Their algorithm determines the appropriate speed of each core of an heterogeneous multicore CPU (cores are different, some more powerful than others) by using a fitting data method to fit the relationship between task arrival rate and core speed. In summary, we can make use of the heterogeneity of CPSs to decrease energy consumption.

Studying APIs is a time-consuming process, especially when the task concerns the meaning of each parameter and its possible values. A current trend is to improve APIs usability [34], where authors identify the common mistakes in API design and make suggestions as to how to solve some undesired situations. From our point of view, the same effort is mandatory to evaluate the implementation in APIs. For open source codes, the implementation of API is available to developers, therefore they can check the correctness of the functionality. This verification can be a time-consuming task since an ordinary API has an elevated number of code lines, which must to be verified. However, it is common that APIs do not offer public implementation, i.e. IAIK cryptographic APIs. In this situation the usability of the interface may indeed be appropriate or even the implementation may correspond to expected functionality, but its execution produces an elevated energy consumption. Nowadays, developers have tools to check API usability and implementation when available. Additionally, a kind of plugin for High Performance Computing (HPC) is developed in [35], however, it has been developed for a specific type of HPC, the data format is proprietary, and uses local measurements. Also, it does not support any kind of integration with IDEs. For energy estimations based on design and executions model of Palladio software, exists the add-on Power Consumption Analyzer (PCA) [36]. It does not use any real metrics, and to obtain estimations, even the complete

hardware must be modelled, as well as details of how many CPU cycles needs every single instruction. However, there are no tools like the plugin for open IDEs, and like the specific web-services for proprietary IDEs, presented in this paper with respect to CPS software development. Another important difference of our approach is the integration of the experimental data collected in the HADAS repository, which promotes its reuse and supports the analysis of the energy consumption of alternative designs and implementations.

In some approaches, such as Selflab [37], the use of a repository to offer several eco-efficient configurations to the developer is part of their future work. One of the most used is GreenMiner [38], which is a complete solution as it provides power metering tools, as well as a repository that stores the collected data, where data mining techniques can be applied. This paper discusses the difficulty of energy data processing since there is a lack of energy data normalisation, and a base format, on which researchers could match the results of their experiments is not defined. Contrarily, this repository is local (deployed individually) to the measurement system. Instead, in HADAS we proposed a collaborative approach, where the results of the work done by different researchers are sent to a shared repository. A similar approach is followed in [39]. The authors discuss the difficulty of performing energy data analysis of a specific use case. They define an energy modeller that allows them to restrict the search results of the repository. They present an architecture compliant with a standard cloud architecture where energy efficiency is addressed specifically at all layers of the cloud software stack and over the entire lifecycle of the cloud application. In the publication [14] a prototype providing technical insights into HADAS, just for web services has been discussed. We presented the advantages of exploiting feature models to reason about software, and choose between assorted designs and implementation alternatives, considering energy consumption. The technical characteristics included: trying different modelling languages and solvers, finally acquiring Clafer and Chocosolver, building the first energy repository in a MariaDB relational database, which suffered several modifications and adjustments later on, and connecting the solver and the repository creating a single

graph at the end. As the first prototype of HADAS, it was deployed in our faculty intra-net, and evaluated with Windows web servers and back-end of web pages consumption. In [11] authors propose an Internet of Things (IoT) components repository instead of a generic repository like HADAS4CPS. This article uses a repository for components that consume less, supplying a 'required and provided' interface as specified in the architecture of an IoT service. This approach does not include measurements of specific implementations as HADAS4CPS does, and it reasons at the architectural level, so its actual usefulness is limited. Besides, the hardware variability is not considered, which is necessary in the CPS domain.

9. Conclusions

In this paper, we have presented the concept of a *Developer Eco-Assistant*, which supports CPS software developers in making sounder decisions with respect to the energy consumption of the ECCs used in their applications. Specifically, in this paper, as a proof of concept, we have focused on the security and communication ECCs for both, Android Smartphones and the Waspmote boards. A plugin for JetBrains IDEs and a specific-purpose web service for Waspmotes have been developed.

We have enhanced the Android and Waspmote APIs, so that software developers can receive contextual help about energy consumption; information previously calculated and stored in the HADAS repository. This process is transparent to the software developers, who can continue to program their applications in the usual way. The only action needed is to activate our *Developer Eco-Assistant*, so that the information is provided as they are typing the code in their preferred IDEs. We have demonstrated that using our approach it is possible to save between 40% and 90% of energy depending on the scenario, while considering the same hardware device. Moreover, we have also demonstrated that it is possible to reduce or increase the energy consumption considerably, depending on the hardware devices where a concrete software functionality is

deployed (e.g. the Waspmote boards and the Android Smartphone in our scenarios).

Our next steps take two different directions. Firstly, more experimental data has to be gathered and integrated into the HADAS eco-assistant, providing energy information of a larger variety of CPS devices. An end goal is to help software developers to decide not only about alternative software solutions, but also, about the most energy-efficient hardware where deploying the software solution. As CPS developers have suggested in the survey, we will measure several configurations of CPSs compatible with generic C/C++ IDEs. Another goal is to provide highly configurable plugins that can be adapted for different users' levels of expertise, with regard to energy consumption, and to different sustainability analysis scenarios. A more complete trade-off analysis between energy and performance will also be incorporated into the eco-assistant. In line with the suggestions made in the surveys, we will improve the interface design of the plugin considering a prompt for the results, the table alignments, info and user-friendliness, data-time plot, and exporting options.

Acknowledgement

This work is supported by the projects Magic P12-TIC1814 and HADAS TIN2015-64841-R (co-financed by FEDER funds). We are particularly grateful for the assistance given by Prof. Juan-Antonio Fernández-Madrigal from the 'Departamento de Ingeniería de Sistemas y Automática' of the University of Málaga in the use of the Waspmote energy profiling shield.

References

[1] Q. Li, M. Zhou, The survey and future evolution of green computing, in: Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications, IEEE Computer Society, 2011, pp. 230–233.

[2] A. Hooper, Green computing, Communication of the ACM 51 (10) (2008) 11–13.

[3] C. Pang, A. Hindle, B. Adams, A. E. Hassan, What do programmers know about software energy consumption?, IEEE Software 33 (3) (2016) 83–89.

[4] G. Pinto, F. Castor, Y. D. Liu, Mining questions about software energy consumption, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 22–31.

[5] A. Banerjee, A. Roychoudhury, Automated re-factoring of android apps to enhance energy-efficiency, in: Mobile Software Engineering and Systems (MOBILESoft), 2016 IEEE/ACM International Conference on, IEEE, 2016, pp. 139–150.

[6] M. Bokhari, M. Wagner, Optimising energy consumption heuristically on android mobile phones, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, ACM, 2016, pp. 1139–1140.

[7] A. Merlo, M. Migliardi, L. Caviglione, A survey on energy-aware security mechanisms, Pervasive and Mobile Computing 24 (2015) 77–90.

[8] D. S. A. Minaam, H. M. Abdual-Kader, M. M. Hadhoud, Evaluating the effects of symmetric cryptography algorithms on power consumption for different data types., IJ Network Security 11 (2) (2010) 78–87.

[9] A. S. Wander, N. Gura, H. Eberle, V. Gupta, S. C. Shantz, Energy analysis of public-key cryptography for wireless sensor networks, in: Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on, IEEE, 2005, pp. 324–328.

[10] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, A. Hindle, Energy profiles of java collections classes, in: Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on, IEEE, 2016, pp. 225–236.

[11] D. Kim, J.-Y. Choi, J.-E. Hong, Evaluating energy efficiency of internet of things software architecture based on reusable software components, International Journal of Distributed Sensor Networks 13 (1) (2017) 1550147716682738.

[12] I. Manotas, L. Pollock, J. Clause, Seeds: a software engineer's energy-optimization decision support framework, in: Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 503–514.

[13] C. Shen, M. Srivastava, Exploring hardware heterogeneity to improve pervasive context inferences, Computer 50 (6) (2017) 19–26.

[14] D.-J. Munoz, M. Pinto, L. Fuentes, Hadas and web services: Eco-efficiency assistant and repository use case evaluation, in: Energy and Sustainability in Small Developing Economies (ES2DE), 2017 International Conference in, IEEE, 2017, pp. 1–6.

[15] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, ACM, New York, NY, USA, 2013, pp. 7:1–7:8.

[16] A. Eiben, Z. Ruttkay, Constraint satisfaction problems.

[17] M. Antkiewicz, K. Bak, A. Murashkin, R. Olaechea, J. H. J. Liang, K. Czarnecki, Clafer tools for product line engineering, in: Proceedings of the 17th International Software Product Line Conference co-located workshops, ACM, 2013, pp. 130–135.

[18] J. A. Montenegro, M. Pinto, L. Fuentes, What do software developers need to know to build secure energy-efficient android applications?, IEEE Access 6 (2018) 1428–1450.

[19] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, L. Yang, Accurate online power estimation and automatic battery behavior based

power model generation for smartphones, in: Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, 2010, pp. 105–114.

[20] A. Gongora, J.-A. Fernández-Madrigal, A. Cruz-Martín, V. Arevalo, C. Galindo, C. Sanchez-Garrido, J. Monroy, J. Canete, Shield arduino de bajo coste para la enseñanza de asignaturas de ingeniería, in: JCER'17, II Jornadas de Computación Empotrada y Reconfigurable, 2017.

[21] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang, K. Li, Energy-efficient resource utilization for heterogeneous embedded computing systems, IEEE Transactions on Computers 66 (9) (2017) 1518–1531.

[22] D.-J. Munoz, J. A. Montenegro, M. Pinto, L. Fuentes, Green security plugin for pervasive computing using the hadas toolkit, in: 15th Intl Conf on Pervasive Intelligence and Computing, 2nd Dependable, Autonomic and Secure Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/-CyberSciTech), 2017 IEEE 15th Intl C, IEEE, 2017.

[23] C. Wild, G. Seber, Chance Encounters: A First Course in Data Analysis and Inference, John Wiley & Sons, 1999.

[24] J. Oh, D. Batory, M. Myers, N. Siegmund, Finding near-optimal configurations in product lines by random sampling, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, 2017, pp. 61–71.

[25] J. Shi, J. Wan, H. Yan, H. Suo, A survey of cyber-physical systems, in: Wireless Communications and Signal Processing (WCSP), 2011 International Conference on, IEEE, 2011, pp. 1–6.

[26] D. Zeng, L. Gu, H. Yao, Towards energy efficient service composition in green energy powered cyber–physical fog systems, Future Generation Computer Systems.

[27] Y. Liu, A. Liu, S. Guo, Z. Li, Y.-J. Choi, H. Sekiya, Context-aware collect data with energy efficient in cyber–physical cloud systems, Future Generation Computer Systems.

[28] P. Palensky, E. Widl, A. Elsheikh, Simulating cyber-physical energy systems: Challenges, tools and methods, IEEE Transactions on Systems, Man, and Cybernetics: Systems 44 (3) (2014) 318–326.

[29] S. A. Murtza, A. Ahmad, M. Y. Qadri, N. N. Qadri, J. Ahmed, Optimizing energy and throughput for mpsocs: an integer particle swarm optimization approach, Computing (2017) 1–18.

[30] S. Parvin, F. K. Hussain, O. K. Hussain, T. Thein, J. S. Park, Multi-cyber framework for availability enhancement of cyber physical systems, Computing 95 (10-11) (2013) 927–948.

[31] G. Wang, Y. Guan, Y. Wang, Z. Shao, Energy-aware assignment and scheduling for hybrid main memory in embedded systems, Computing 98 (3) (2016) 279–301.

[32] A. Castiglione, F. Palmieri, U. Fiore, A. Castiglione, A. De Santis, Modeling energy-efficient secure communications in multi-mode wireless mobile devices, Journal of Computer and System Sciences 81 (8) (2015) 1464–1478.

[33] D. González, O. Esparza, J. L. Muñoz, J. Alins, J. Mata, Evaluation of cryptographic capabilities for the android platform, in: International Conference on Future Network Systems and Security, Springer, 2015, pp. 16–30.

[34] B. A. Myers, J. Stylos, Improving api usability, Communications of the ACM 59 (6) (2016) 62–69.

[35] C. Guillen, C. Navarrete, D. Brayford, W. Hesse, M. Brehm, Energy model derivation for the dvfs automatic tuning plugin: tuning energy and power related tuning objectives, Computing 99 (8) (2017) 747–764.

[36] C. Stier, H. Groenda, A. Koziolek, Towards modeling and analysis of power consumption of self-adaptive software systems in palladio, in: SOSP'14 Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days 2014, 2014, p. 28.

[37] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, J. Visser, Seflab: A lab for measuring software energy footprints, in: Proceedings of the 2nd International Workshop on Green and Sustainable Software, IEEE Press, 2013, pp. 30–37.

[38] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, S. Romansky, Greenminer: A hardware based mining software repositories software energy consumption framework, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 12–21.

[39] K. Djemame, D. Armstrong, R. Kavanagh, A. Juan Ferrer, D. Garcia Perez, D. Antona, J.-C. Deprez, C. Ponsard, D. Ortiz, M. Macías Lloret, et al., Energy efficiency embedded service lifecycle: Towards an energy efficient cloud computing architecture, in: Joint Workshop Proceedings of the 2nd International Conference on ICT for Sustainability 2014, CEUR-WS. org, 2014, pp. 1–6.