Context-aware energy-efficient applications for cyber-physical systems

Jose-Miguel Horcas *, Mónica Pinto, Lidia Fuentes

Universidad de Málaga, AndalucÃa Tech, Spain

Keywords:
Energy efficient cyber-physical systems
software sustainability
self-adaptive greenability
Dynamic Software Product Lines

A B S T R A C T

Software systems have a strong impact on the energy consumption of the hardware they use. This is especially important for cyber-physical systems where power consumption strongly influences the battery life. For this reason, software developers should be more aware of the energy consumed by their systems. Moreover, software systems should be developed to adapt their behavior to minimize the energy consumed during their execution. This can be done by monitoring the usage context of the system and having runtime support to react to those changes that impact the energy footprint negatively. Although both the hardware and the software parts of cyber-physical systems can be adapted to reduce its energy consumption, this paper focuses on software adaptation. Concretely, the paper illustrates how to address the problem of developing context-aware energy-efficient applications using a Green Eco-Assistant that makes use of advanced software engineering methods, such as Dynamic Software Product Lines and Separation of Concerns. The main steps of our approach are illustrated by applying them to a cyber-physical system case study.

1. Introduction

The percentage of global emissions attributable to Information Systems is expected to further increase in the coming years, due to the proliferation of Internet-connected devices omnipresent in our daily lives [1]. Software never consumes energy in itself, rather its design, implementation and usage context strongly affect the energy consumed by the hardware [2,3]. So developers should be more aware of the energy consumed by these systems, and try to build energy-efficient applications that self-adapt their behavior to minimize the power consumed during their execution, i.e., develop self-greening applications.

Regrettably, there is a narrow view of developers and users about their responsibility for the energy consumed during application execution. They rarely address energy efficiency as some recent studies show [4,5]. Moreover, existing experimental results about how to optimize energy consumption at design time [3,6,7] were not conceived as reusable solutions of runtime energy optimizations. Therefore, although software energy efficiency is becoming increasingly important, development processes of self-greening systems supported by tools are still in their infancy.

The energy consumption of an application depends on several factors, such as the hardware resources, operating system, input parameters, and workload [6]. Several works identify which parts of an application's code influence the energy consumption, for example encryption and compression algorithms, communication, or storage [8,9]. Indeed, once the application is deployed, these approaches evidence the strong influence that the *usage context* has on the energy consumed by certain functionalities [10]. It depends, for example, on the amount of data the system needs to store, transfer or query, or on how the user interacts with the system. This means that self-greening applications should not only be prepared at design time to be energy-efficient; they also need to be context sensitive in order to adapt their behavior to minimize the energy consumed during their execution [4,5].

In our approach, the usage context is defined in the energy scope as *all aspects that vary under different usage conditions of a functionality and affect product performance for the energy-efficiency attribute*. For example, the amount of data to be exchanged (e.g., the size of the objects to be sent from one device to another), the communication frequency between the devices, or the communication protocol (e.g., WiFi, Bluetooth), are all aspects that affect the energy consumption of a specific functionality. Identifying the usage context's variables for each functionality and analyzing how they affect the energy consumption is a complex task, since the usage context has a high degree of variability. For instance, each contextual variable (e.g., size of the object to be sent) can take

* Corresponding author. Tel.: +34625257121.
E-mail addresses: horcas@lcc.uma.es (J.-M. Horcas), pinto@lcc.uma.es (M. Pinto), lff@lcc.uma.es (L. Fuentes).

different values (e.g., between 1 and 1000 MBytes) based on the usage of the application.

This paper shows how advanced software engineering methods, such as Dynamic Software Product Lines (DSPLs) [11] and Separation of Concerns (SoC) [12], can help to develop self-greening energy-efficient applications for cyber-physical systems. Concretely, we present a Green Eco-Assistant for the development of context-aware energy-efficient applications. We propose to collect energy-related information at design time and use it at runtime to adapt the application behavior to the real energy consumption. Our approach is illustrated through a running example in the domain of Intelligent Transportation Cyber-Physical Systems (CPS). Although in this example we focus on the monitoring, storing, communication and compression concerns, the Green Eco-Assistant could include any other large consumer concern where several design and implementation solutions can be identified. We demonstrate that there are scenarios in which CPS applications can reduce their energy-consumption up to 70%, depending on the current usage context.

In summary, this paper renders the following contributions:

- A software engineering approach that assists developers during the design and development of context-aware energy-efficient applications.
- A process for modeling the usage context and the configurable implementations of recurrent functionalities that can affect the energy efficiency of the applications. This includes a generic schema or template to model the variability of the usage context and the configurable parameters.
- A non-intrusive design and implementation solution that endows applications with self-greening capacities at a low energy cost.

After this introduction, Section 2 discusses the background and Section 3 the related work. Section 4 presents the CPS case study used to illustrate our approach. Then, Section 5 presents the main challenges and an overview of how our approach addresses these challenges. Sections 6, 7, and 8 give the details of applying our approach to the CPS case study. Section 9 evaluates our approach and discusses the threats to validity. Finally, Section 10 concludes the paper.

2. Background

This section briefly presents background information about Dynamic Software Product Lines and Separation of Concerns.

2.1. Dynamic Software Product Lines

A *Software Product Line (SPL)* is "a set of software-intensive systems that share a common, managed set of *features* satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [13]. A feature is a characteristic or end-user-visible behavior of a software system. Features are used in SPL engineering to specify and documenting commonalities and differences of the products, and to guide structure, reuse, and variation across all phases of the software life cycle. A *Dynamic Software Product Line (DSPL)* brings this engineering process to runtime, where a single system is able to adapt its behavior at runtime [11].

Variability modeling is the main activity of both SPLs and DSPLs, where the common and variable features of the system are specified in a *variability model* or *feature model*. Then, the SPL engineering process generates products by selecting specific characteristics specified in the variability model. In a DSPL, the variability model describes the potential range of variations that can be produced at runtime for a single system — i.e., the dynamic variation points. Therefore, the software architecture supports all possible adaptations defined by the set of dynamic variation points [11].

2.2. Separation of Concerns and Aspect-Orientation

Aspect-Oriented Software Development (AOSD)[1]~[14] promotes the principle of *separation of concerns* throughout all the phases of the software life cycle, by separating *crosscutting concerns*. In S. Apel el al, *Crosscutting is a structural relationship between the representation of two concerns*. Since crosscutting concerns are normally hard to find, understand and work with, separating and specifying them as *aspects* enhances the reconfiguration management of the system.

Several techniques (e.g., design patterns, mixing classes) have been developed for dealing with the problem of modularization of crosscutting concerns. One of the most advanced and sophisticated technique is *Aspect-Oriented Programming* (AOP) [15]. In AOP, implementation of crosscutting concerns are encapsulated in a new entity named *aspect*, and the code of the base application contains only the main functionality of the system excluding any reference to the crosscutting concerns. Modeling of crosscutting concerns as a separate entity, such as an aspect, in which its implementation appears encapsulated only in a part of the program, smooths coupling between modules and increases cohesion of each of them. Moreover, as a consequence of low coupling and a high cohesion, the maintainability of the global system improves due to the fact that changes in a module affect only that module; and thus, this facilitates the reconfiguration of the system. Also the reusability improves due to both base code and aspects that can be reused easier in different systems. There are a lot of crosscutting concerns that are usually useful to treat separately and so can be modeled as examples of aspects: logging, authentication, trace, coordination, synchronization, security, persistence, fail-over, error detection and correction, memory management, internationalization, localization, monitoring, data validation, transaction processing, caching, etc.

3. Related Work

The software developer community is starting to pay more and more attention to the energy efficiency attribute. Here we summarize some representative works (Table 1). For each considered approach, we indicate the type of study, the main output, and the knowledge that is derived from each work.

Empirical studies. Recent empirical studies [4,5] made at different stages of the software life cycle show that software developers do not have enough knowledge about how to reduce the energy consumption of their software solutions. Thus, the majority of developers are not aware about how much energy their applications will consume and so, they rarely address energy efficiency. Even practitioners that have experience with green software engineering have significant misconceptions about how to reduce energy consumption [5]. These studies also evidence the lack of tool support of green computing, not only at the code level, but also at higher abstraction levels — i.e., requirements and software architecture levels [16].

Experimental works at code level. There are plenty of experimental approaches that try to identify what parts of an application influence more in the total energy footprint of an application — i.e., to identify the energy hotspots [7,17,18]. These works propose to minimize energy consumption by focusing on code level optimizations. A common goal to all of them is the definition of energy profiles for different energy consuming concerns. They usually focus on one particular energy consuming concern and report the energy consumption of different implementations [10].

[1] http://www.aosd.net/

Table 1
Comparison of energy-aware approaches.

Type	Appr.	Output	Knowledge
Empirical Studies	[4]	Qualitative study exploring the knowledge of practitioners interested on energy consumption from different perspectives (requirements, design and construction).	Green software practitioners care and think about energy; however, they are not as successful as expected because they lack necessary information and tool support.
	[5]	Qualitative study exploring the knowledge of practitioners about energy consumption.	Programmers rarely address energy. There are important misconceptions about software energy consumption.
	[16]	Qualitative study exploring requirements engineering practitioners' behavior towards sustainability.	Lack of methodological support; lack of management support; requirements trade-off and risks.
Experimental works	[7]	Empirical evaluation of 21 design patterns. Compiler transformations to detect and transform patterns during compilation for better energy efficiency with no impact on coding practices.	The energy consumption of design patterns highly depend on the running environment; several studies identified both patterns and anti-patterns regarding energy consumption.
	[10]	Energy profiles of operations on Java Collections.	It claims that a per-method analysis of energy consumption must be made.
	[17]	Quantitative information about the energy consumption of 405 Android apps.	More than 60% of energy consumed in idle states; the network is the most energy consuming component; developers should focus on code optimization.
	[18]	Define the SEED framework for the automatic optimization of energy usage of applications by making code level changes.	Support is needed to integrate the insights gained by existing experimental studies to help identifying the more energy-efficient alternatives.
Reasoning about energy efficiency	[2]	Architecture Description Language and tool set with support for the specification, calculation and analysis of energy consumption.	At the architectural level the energy consumption can be estimated based on resource consumption (CPU, HDD, etc.) and usage models.
	[19]	Plug-in integrated with the AADL tool to support the specification and analysis of energy consumption.	It focuses in the energy overhead of inter-process communication, an important service of embedded systems.
	[20]	Identify energy efficiency as a quality attribute and define green architectural tactics for cloud applications. Identify relationships between different architectural tactics.	Energy efficiency addressed from a software architecture perspective. Software architects need reusable tactics for considering energy efficiency in their application designs.
Energy-based reconfiguration	[9]	Maintain the energy consumption of the system within reasonable levels.	Monitor the consumption of previously identified energy hotspots at runtime.
	[21]	Build real-time profiles of energy consumption.	Monitor and react to changes to update the behavior of applications to their "energy usage profile".
	[22]	A Dynamically Reconfigurable Energy Aware Modular Software (DREAMS) architecture.	Dynamic reconfiguration of energy aware software in sensor networks.
Energy-aware CPSs	[23]	A green energy powered CPS architecture for fog computing.	Formulate a energy-efficient CPS service composition problem and propose a heuristic algorithm.
	[24]	Context-aware sensing and collection data scheme with energy efficient in CPS.	Increase lifetime and data collection accuracy, and decreasing data transmission in CPS.

Reasoning about energy efficiency at design level. There are other works that demonstrate that changes at the design level tend to have a larger impact in energy consumption [3,6]. These works consider energy efficiency as a new quality attribute [6]. What is important at this level is to be able to compare the energy consumed by different design alternatives, and also to be able to perform a trade-off between energy efficiency and other quality attributes such as performance. There are some relevant approaches that focus on the design of catalogs of energy-aware design patterns [7], as well as new architecture description languages that incorporate an energy profile and analysis support [2,6]. The experimental part of these works consists of checking at the code level the effects of applying specific design or architectural patterns [6]. Also, some works [2,19] provide support to specify the relationships between components modeling different energy concerns. These relationships can then be used during the analysis phase to see how a energy concern (e.g., compression) can influence in other energy concerns (e.g., communication or data storage). But, the identification and specification of dependencies between energy concerns has to be done manually by the software engineer.

Energy-based reconfiguration at runtime level. Here we focus on proposals that are able to monitor changes on the user behavioral patterns and react to the effects of those changes on the consumption of energy. They should also be able to update the behavior of applications to their "energy usage profile". The final goal is to maintain the energy consumption of the software system within reasonable levels. Some proposals monitor the energy consumption of previously identified energy hotspots at runtime [9], and others build real-time profiles of energy consumption [21]. Moreover, there are examples of the dynamic reconfiguration of energy aware software in different domains. For instance, DREAMS [22] is a Dynamically Reconfigurable Energy Aware Modular Software architecture for sensor networks. However, none of these work defines a generic and reusable approach as we make.

Energy-aware cyber-physical systems. Energy efficiency is a key factor in CPS since it determines the autonomy of the device. However, few attention is paid to energy consumption of CPS at the software level [23,24]. For instance, in [23] the authors propose a green energy powered CPS architecture based on a service composition in the context of fog computing. But the usage context and the corresponding reconfiguration of the deployed functionality are omitted. In [24], 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 is more generic suitable for any energy consuming concern.

Summarizing, there is a lack of methodological and tool support that makes the reuse of the energy information collected by existing experimental works highly difficult. Moreover, the large variability of design and implementation solutions requires the use of advanced software engineering methods in order to reason about energy efficiency. Finally, despite the importance of controlling the energy consumption of battery-powered devices in CPSs, there are

not too many works covering this topic from a software perspective in the context of CPSs.

4. Intelligent Transportation CPS case study

Our CPS is a road unit consisting of a Raspberry Pi (RPi) and multiple sensors deployed in a highway that monitor contextual information about the traffic, road status, weather, etc. The RPi collects all that information, generates a file and sends it to a cloud server to be processed. The functionality of the road unit can be highly configured and the question is whether these changes affect the energy consumption of the CPS.

For instance, we have identified several functionalities as energy consuming concerns that will influence more or less to the energy consumption, according to how some of their parameters vary. Additionally, developers can use different software solutions to implement these functionalities (e.g. different compression algorithms) and the energy consumption will vary depending on the selected one. As an example, the road unit can be configured to *monitor* a variable set of parameters every second (from 8 up to 128 parameters), so 86,400 samples per parameter are collected by the end of the day. The size of the generated file depends on the number of parameters to be monitored, e.g, monitoring 8 parameters generates a 10 MB file, while monitoring 128 parameters generates a 160 MB file. The *communication* with the server can also be configured with different frequencies. For instance, the file can be sent every day, or can be archived to be sent once a week. Due to the space limitation (the maximum storage capacity of the RPi is 512 MB), the size of the managed files can be reduced using several *compression* algorithms that are available in the RPi (e.g., LZ77, Burrow-Wheeler, LZMA2,...).

Let us suppose that, initially, the road unit has been deployed with a basic configuration that monitors 16 parameters and generates a file of 20 MB by the end of the day. This file is archived daily and it is sent to the server once a week that will generate some statistics and a report about traffic density at different times of the day for a week. Before sending the file, it is compressed by using the LZ77 compression algorithm. Several questions arise here, as for instance, *what is the energy consumption of this system configuration?, is there any other system configuration with similar functionality but lower energy consumption?*

Moreover, when a context change is detected (e.g., a traffic accident, rainy days, etc.) the functionality of the road unit should be dynamically reconfigured to the new context. For example, during high affluence of traffic (e.g., on summer), the road unit increases the number of parameters to be monitored up to its maximum (128). However, *what is the impact of this runtime change on the energy consumption?* It is supposed that an increment on the file size will also increase the energy consumption of the device, but software developers need evidences of the energy consumption of their decisions, as well as the possibility of performing a sustainability analysis that helps them to decide among different system configurations. Using this case study we will illustrate how our approach helps software developers to answer these questions.

5. The Green Eco-Assistant: Challenges and Overview

This section identifies the main challenges that arise in the development of self-adaptive energy-efficient applications and provides an overview of how our Green Eco-Assistant copes with these challenges.

5.1. Challenges

Five main challenges have been identified:

Challenge 1 (C1): Empirical studies [4,5] show that software developers need help to identify runtime energy hotspots. A *runtime energy hotspot* is a point in the application that when there is a considerable increase in power consumption due to a change in the usage context, it is possible to reduce it by modifying the application software components that provoke this power rise. Recent studies propose some green computing practices [10], however developers do not know how to apply them in their developments. The main conclusion is that software developers need more precise evidence about how to tackle the energy efficiency problem, a methodology, and tool support to effectively address software sustainability [4,5]. Thus, **the first challenge is to provide the means to identify the runtime energy hotspots.** In the context of our CPS case study, our approach will help software developers to identify monitoring, communication and compression as runtime energy hotspots.

Challenge 2 (C2): Finding the most energy-efficient solution for each runtime energy hotspot is not trivial since there is high variability of components that implement the functionality required by the hotspot with different energy costs. For example, for the communication energy hotspot, each protocol (e.g., WiFi, Bluetooth) will consume a different amount of energy depending on the communication frequency and the size of the data to be sent. Thus, after identifying the energy hotspot, software developers need to model the energy consuming concerns, being aware of the variability of the existing solutions, including the parameters that could affect the energy expenditure. The *energy consuming concerns* are the concerns that model the runtime energy hotspots at design time (e.g., privacy, caching, etc.). **The challenge is to explicitly define the variability of design solutions that can mitigate the energy consumption according to current user interaction.** Following with the case study, the variability of the monitoring, communication and compression concerns are modeled in the Green Eco-Assistant, ready to be reused by software developers.

Challenge 3 (C3): The energy consumption highly depends on several factors, and some of them, such as the usage context, will vary at runtime. So, the variability of the usage context should be explicitly modeled and its impact on the power consumption linked to the different application functionalities. Moreover, the energy consumption of each variant of the energy hotspots and their usage context should be provided for application developers in a format so that they can easily access, compare and analyze its impact at runtime. Thus, **the third challenge is to provide developers with tools that help them making a sensible eco-efficiency analysis at design time, about the possibilities of optimizing energy consumption at runtime for a given application.** In our example, software developers can use our approach to know the energy consumption of their system initial configuration. More interesting is the sustainability analysis that can be done to check if there are other configurations with similar functionality and lower energy footprint.

Challenge 4 (C4): The eco-efficiency analysis may result in more than one design solution for a given energy hotspot, each one fitting a different usage pattern. This means that the application needs to be able to react to changes in the usage patterns at runtime in order to self-adapt to the variant with least energy expenditure. There are some related papers that perform dynamic reconfiguration of energy aware software [21,22], but they are domain specific and do not provide a generic and reusable approach, which we consider developers need. So, **an important challenge is to define energy reconfiguration rules to adapt the application to the varying usage patterns by exploiting the energy saving scenarios identified in the eco-efficiency analysis.** By comparing different usage contexts and their energy consumption the CPS can be adapted at runtime not only for the context change identified

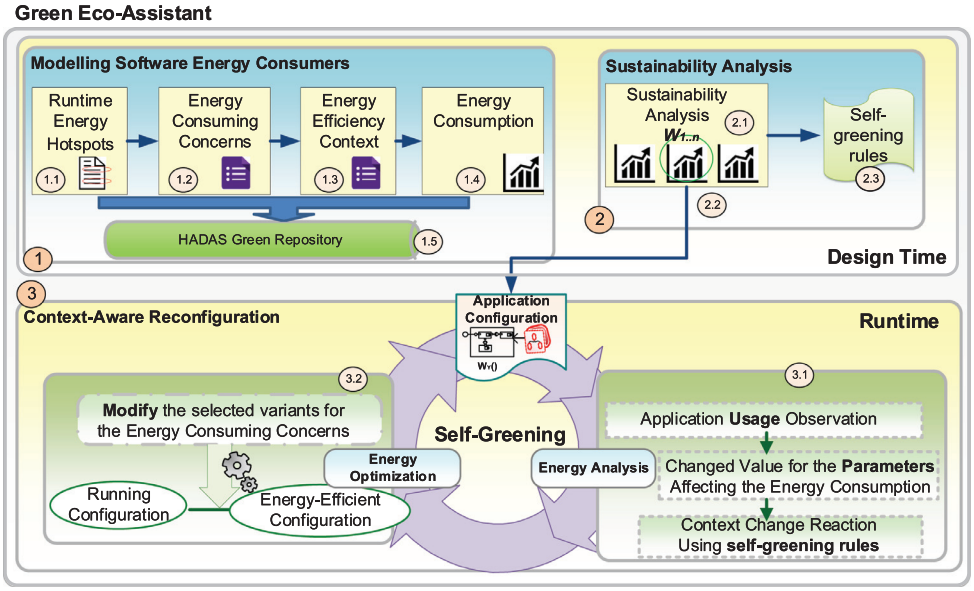

Fig. 1. The Green Eco-Assistant approach.

in the system specification, but also to other ones that save energy without penalizing the system quality of service.

Challenge 5 (C5): The energy reconfiguration rules will drive the application adaptation at runtime by replacing the modules that implement the energy consuming concerns with others more energy efficient for a new execution context. The **last challenge is to provide a non-intrusive design and implementation solution that endows applications with self-greening capacities at a low energy cost.**

5.2. Overview

This section overviews how the Green Eco-Assistant copes with the challenges presented. The details are provided in the rest of the paper. Figure 1 presents the approach, where three main steps are differentiated. Steps 1 and 2 occur at design time and step 3 at runtime: (1) modeling the software energy consumers; (2) performing the sustainability analysis, and (3) the context-aware reconfiguration at runtime.

During the modeling of the software energy consumers the eco-assistant copes with Challenges 1 and 2. Learning to recognize energy hotspots is absolutely essential and helpful in any energy-aware development process. However, software developers still do not have the skills to identify these energy hotspots. Additionally, there are not catalogs of runtime energy hotspots, similar to the existing catalogs of design patterns. Trying to cope with this shortcoming, and after analyzing several approaches, we can conclude that many energy hotspots are recurrent, and appear in the majority of applications [22]. So, our approach helps developers in this task by providing a list of the most recurrent energy hotspots. Then, application developers can select those energy hotspots identified as part of the application's functionality (e.g., compression), and the variants they want to explore (e.g., to compress the data file in the CPS device or send it without compression to the server). This selection is done through a set of forms provided by the assistant (label 1.1).

The concerns that model the runtime energy hotspots at design time can be considered as energy consuming concerns (label 1.2), which could be designed in different ways. For example, there are different options to store data (in a data structure, cache memory, etc.), each with a different energy consumption that depends on some input parameters that can vary at runtime such as the size or type of data. In addition, they are usually scattered or crosscut several components (i.e., they are crosscutting concerns) [25], so it is beneficial to model and implement them independently of the system's functionality, to facilitate their replacement at runtime by more eco-efficient designs or implementations. Moreover, the energy consumption of these consuming concerns will depend on the usage context and, for this reason, the part of the usage context with an impact on the energy consumption of a concern — i.e., the energy-efficiency context[2] needs to be identified and modeled (label 1.3). Taking into account all these factors, our approach calculates the energy consumption of all the concern variants (label 1.4). Since these concerns are common to many applications, we propose storing them in the HADAS Green Repository [26] ready to be reused (label 1.5).

Challenges 3 and 4 are satisfied by the step 2 (label 2). The key to the success of self-greening applications is to fully exploit the energy saving possibilities arising at runtime. So, the main role of the Green Eco-Assistant is to provide the necessary means to make a sustainability analysis (label 2.1), at design time, about the possibilities of optimizing energy consumption at runtime for a given application. This means that our approach can be used to generate an initial application configuration that satisfy our energy requirements (label 2.2), but also to see whether it is worthwhile specifying a reconfiguration rule to replace, at runtime, a specific concern implementation with another due to, for instance, a drastic change in the usage context (label 2.3).

Finally, we cope with challenge 5 in the third step (label 3). Once the initial system configuration has been deployed, the system has to monitor (label 3.1) and reconfigure the current system (label 3.2). The greatest challenge here is to define a self-greening mechanism that wastes the least amount of energy. The context-aware reconfiguration loop is an adaptation of the classic MAPE (Monitoring, Analysis, Plan and Execution) reference model [27]. We mainly (M)onitor the application usage (*Application usage observation*), (A)nalyse the context in the*context change reaction* task using the self-greening rules and generate the (P)lan, and finally

[2] In the paper we use the terms energy-efficiency context and usage context indistinctly, understanding that the energy-efficiency context is the usage context that affects the energy consumption of applications.

the (E)xecution of the plan is reflected in both *changed value for the parameters affecting the energy consumption* and *modify the selected variants for the energy consuming concerns*.

6. Modeling the Energy Consuming Concerns

We based on the concepts of "runtime energy hotspot", "energy-consuming concerns" and "usage context" previously introduced. We argue that energy-aware software engineering solutions should focus more on reusable concerns, and less in specific systems. So, in our approach we will consider the energy influence of recurrent concerns such as compression, encryption or communication. Additionally, for each of these recurrent energy consuming concerns, there are several alternative designs, and each of them could have different energy consumption, which mainly depends on some input parameters, such as the size or data type. All the alternative design solutions for every energy consuming concern are modeled and their energy consumption is stored into a Green repository, that contains a database with power consumption measures [28]. In our approach, application developers can use the Green Eco-Assistant at design time to perform a sustainability analysis of the different variants. The Green Eco-Assistant then generates the initial application configuration fulfilling the developer needs. But, this sustainability analysis will also help to identify those situations where the energy expenditure strongly depends on some parameters of the usage context that can vary at runtime. So, our solution aims to help designers to identify the opportunities to save energy not only at design, but also at runtime. This information will be used by the developer to specify the self-greening rules that will trigger a reconfiguration at runtime of the cyber-physical system.

Figure 2presents an schema of the process for modeling the energy consuming concerns, including the energy-efficiency context, and for collecting the energy consumption measures. The result of applying this process to our CPS is presented in Figure 3 and Figure 4.

The first step to model the energy consuming concerns is the identification of those functionalities that most influence the energy consumption (Step 1 in Figure 2). Our approach focuses on those functionalities that are recurrent, which researchers identify as large energy consumers [8,9]. The functionalities implemented by application independent frameworks, which are usually required by many applications, are good candidates to be incorporated into our process. This is because the analysis of the energy consumed by this kind of functionality can be conducted independently of an application's internal operation, for different implementation frameworks. Some examples of these functionalities are storage, encryption, compression, and communication.

There are plenty of studies showing that there is a high variability of alternative implementations and design solutions to many energy consuming concerns [6,7,10], and some of them permit their replacement at runtime to achieve energy savings. For this reason, we follow a DSPL approach [11] to explicitly model the variability of the energy consuming concerns, using a variability model (e.g., feature models [29], CVL [30]), which defines the configuration space — i.e., the possible allowed self-adaptations triggered by different energy contexts. For instance, top of Figure 3 shows an excerpt of the Green Eco-Assistant variability model (i.e., the feature model tree) in the Common Variability Language (CVL) [30], with some energy consuming concerns like monitoring, security, caching, compression, code migration, archiving, synchronization and communication. In this paper we focus on monitoring, data compression and communication, three concerns present in our CPS case study. It can be seen that for the compression concern we include several algorithms that consume more or less energy depending on the file size (bottom of Figure 3), which usually varies at runtime. For the communication concern, transmission of the data can be done with multiple protocols (WiFi, Bluetooth, NFC), the energy consumption of which also depends on the file size.

6.1. Modeling the usage context and the configurable parameters

Considering that the consumption of the energy consuming concerns depends on the usage context, *what are the variables that affect the energy consumption, and so should be defined as part of the usage context?* [2] The complexity of identifying and modeling the usage context is increased if we consider that the functionality of each energy consuming concern provides many operations and each of them may impact differently on the energy consumption (e.g., compress and decompress a file). Additionally, each operation can support different data types (e.g., text file, parameters file, audio and video files) that may vary in size and need to be managed differently. As can be seen in Figure 2 in our process the *Usage Context* of an application is defined as the set of contextual variables, operations, and data types that may affect the energy consumption of an energy consuming concern. Then, Step 2 in Figure 2 identifies all the variables and the values that they can take, the operations, and the data types for each energy consuming concern.

The energy consumption of an energy consuming concern directly depends on how efficient the functionality implementation makes use of hardware resources. For example, different compression algorithms have different levels of energy consumption that depend on the compression format and on the compression rate, among other configurable parameters. We propose to extract the information about the configurable parameters that the different frameworks offer (Step 3). In this paper, we specify all the possible variants of the usage context and the implementations of the functionality (Step 4), as part of the same variability model [31]. The relevance of modeling the variability of the energy consuming concerns and their usage context is to be able to compare the energy consumption of different configurations of the same functionality (design variations, different frameworks and parameters), independently of the execution environment (hardware, operating system).

For each functionality (e.g., compression, encryption, communication,...) proposed in the variability model schema in Figure 2, we differentiate the mandatory features of the usage context (`Contextual Variables`, `Contextual Operations`, and `Contextual Data Types`) from the mandatory features of the implementations (`Configurable Parameters` and `Frameworks`). This schema is fixed and is the same for all functionalities, while the specific features of the usage context and the implementations depend on each functionality. Restrictions between features (cross-tree constraints) are specified as OCL constraints, for example when a particular framework does not support a specific characteristic (e.g., a specific data type, operation or configurable parameter).

Identifying the energy consuming concerns with their configurable parameters and with their usage context (i.e., contextual variables, operations and data types) is not a trivial task for non domain experts. For example, the encryption functionality has two basic operations (encrypt and decrypt), and can operate over different data types (string or objects in general). The logging functionality has only one operation (log a status message), and the type is usually a string. However, identifying the contextual variables of the usage context is even harder. In our experience contextual variables should fulfill the following properties:

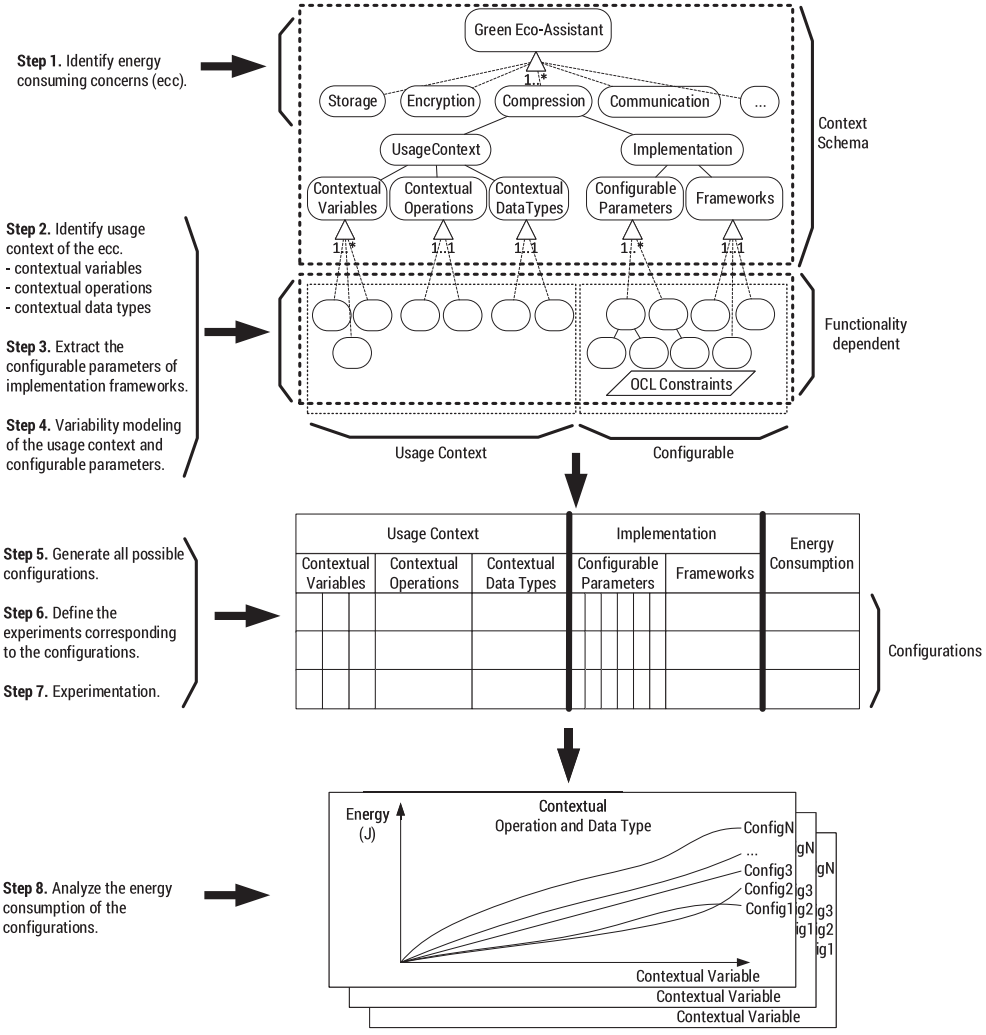

Fig. 2. Schema of the process for modeling the energy consumer concerns.

- They can be attributes of the input parameters of the operations (e.g., size of the object to be compressed, length of the string to be logged).
- They can represent states of the functionality (e.g., current capacity of a memory cache).
- Their values can change at runtime as a response to a user interaction over different executions.
- Their values are unknown a priori, but could be monitored to find the values during execution.
- A variation of their value may affect the quality attributes of the application (e.g., energy consumption, performance).

These properties allow identifying and differentiating the variables or parameters that should be modeled under the usage context from the configurable parameters of a particular implementation [31]. For example, the maximum capacity of a cache memory is something that we can configure in the framework, and will not change unless the developer or the application decides to change it. The same is true for the encryption algorithm, or with the compression format.

Coming back to Figure 3, for compression, the contextual variables of the usage context is the current size of the file to be compressed (the `FileSize` feature). Compression offers the classical operations for `Compress` and `Decompress` a file. The data types supported by the compression algorithms, in this example, are binary files (`BIN`), text files (`TXT`), and files of parameters like float numbers (`CSV`).

In addition, some implementation characteristics that usually have every implementations of the compression concern are the encoding algorithm and the level or ratio of compression (the `Level` feature). We have included several algorithms (`LZ77`, `Burrows-Wheeler` and `LZMA2`); and implementations (`gzip`, `bzip2`, and `xz`) that are usually available in the CPS devices considered (e.g., in the RPi of our case study). Note that not all compression implementations support every algorithm along with its implementation characteristics. For instance, in this case, each compression tool supports one of the algorithm specified. This is expressed in our variability model in CVL as constrains between features, which are usually specified as OCL restrictions and attached to the affected features (e.g., `gzip` $\Leftrightarrow$ `LZ77`).

6.2. Generating different configurations

Using an SPL approach and its tool support [32,33], once we have modeled all the variability of the energy-efficiency context and the implementations, we can automatically generate different configurations and define the experiments to estimate the energy consumption of each concern for each of the CPS devices (Steps 5 and 6 in Figure 2).

In particular, in our Green Eco-Assistant, we mapped the variability models to logical constraints in a Constraint Satisfaction

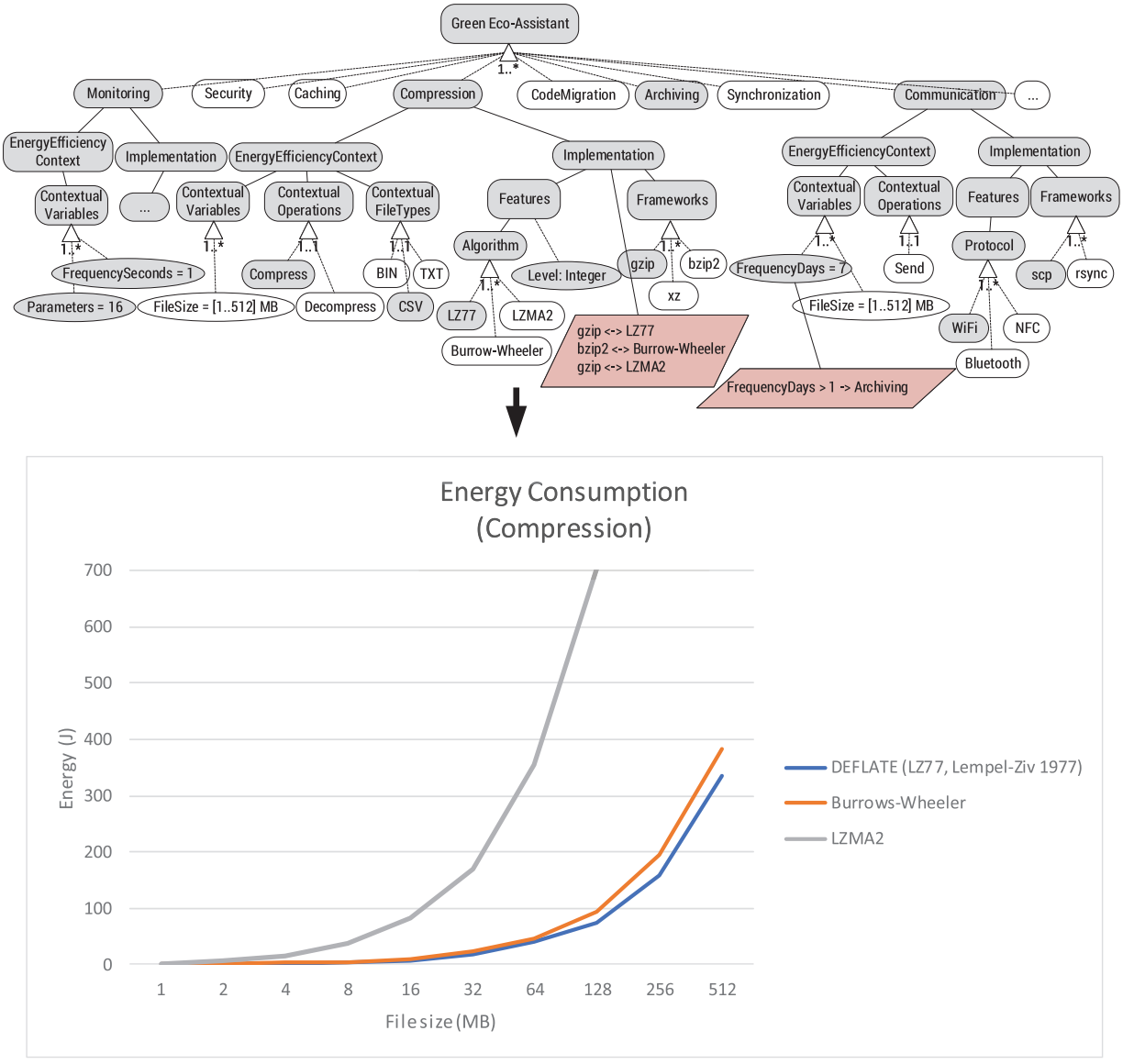

Fig. 3. Our approach applied to our cyber-physical system.

Problem (CSP). This formalization allows us to resolve the CSP problem using a CSP solver to guarantee that we generate valid configurations. Here, we have used Clafer[3], a general-purpose lightweight modeling language, and CHOCO[4], a Java library for constraints programming, to implement the variability models and configurations as CSP problems, and resolve them.

6.3. Estimating the energy consumption

What the developer needs to know at design time are the options that exist to address a specific runtime energy consuming concern, and the expected energy consumption of each of them at runtime. Energy consumption mainly depends on the resources that each application component is expected to consume (e.g., cpu cycles, and disk access) and on the hardware characteristics (e.g., cpu cycles/s, and MB/s.). With this information, it is possible to estimate the expected energy consumption by conducting experimental studies, or by simulating energy models. Note that the exact number of Joules consumed by different energy consuming concerns considering specific hardware is not so important to identify energy consumption trends, although the relative energy is. So, the intention is to store the energy consumption obtained following different approaches, and provide this information to the developer. Certainly, we could gather results from many already published experimental studies [10,34], store them in the HADAS Green Repository and provide advice based on these results.

The energy consumption shown in this paper was experimentally calculated from real products, but can also be predicted, through simulation, by an architecture design [35], or benchmarks [9]. Experiments to estimate the energy consumption of a given functionality must cover the whole range of variations for the possible values of the usage context variables and configurable parameters (Step 7 in Figure 2). Otherwise the experimental profile will be incomplete. A row in the table shown in Figure 2 represents one configuration of the variability model. The last column `Energy Consumption` contains the experimentation results for each configuration. Each contextual variable of the usage context can be seen as a energy function F that depends on that variable, like, $F(FileSize)$ for the file size of the compression functionality.

[3] http://www.clafer.org/
[4] http://www.choco-solver.org/

		USAGE CONTEXT					IMPLEMENTATION					ENERGY CONSUMPTION (J)
		Contextual Variables			Contextual Operations	Contextual Data Types	Configurable Parameters				Frameworks	
		#Params	File Size (MB)	Frequency (days)			Algorithm	Level	Protocol	...		
		...	...	...	...	...	...	...	...		...	...
Optimum configs. →	config1	16	20	0	monitoring, communication	CSV	-	6	WiFi		scp	13,27
Optimum configs. →	config2	128	160	0	monitoring, communication	CSV	-	6	WiFi		scp	117,29
	config3	16	9	0	monitoring, compression, communication	CSV	LZ77	6	WiFi		gzip, scp	20,47
	config4	128	72	0	monitoring, compression, communication	CSV	LZ77	6	WiFi		gzip, scp	171,78
	config5	16	60	3	monitoring, archiving, communication	CSV	-	6	WiFi		tar, scp	28,79
	config6	128	480	3	monitoring, archiving, communication	CSV	-	6	WiFi		tar, scp	277,16
	config7	16	27	3	monitoring, archiving, compression, communication	CSV	LZ77	6	WiFi		tar, gzip, scp	51,68
	config8	128	215	3	monitoring, archiving, compression, communication	CSV	LZ77	6	WiFi		tar, gzip, scp	467,74
	config9	16	27	3	monitoring, compression, archiving, communication	CSV	LZ77	6	WiFi		gzip, tar, scp	50,45
	config10	128	216	3	monitoring, compression, archiving, communication	CSV	LZ77	6	WiFi		gzip, tar, scp	429,29
	config11	16	140	7	monitoring, archiving, communication	CSV	-	6	WiFi		tar, scp	63,15
	config12	128	1120	7	monitoring, archiving, communication	CSV	-	6	WiFi		tar, scp	4790,41
Initial config. →	config13	16	63	7	monitoring, archiving, compression, communication	CSV	LZ77	6	WiFi		tar, gzip, scp	111,88
	config14	128	384	7	monitoring, archiving, compression, communication	CSV	LZ77	6	WiFi		tar, gzip, scp	3155,78
	config15	16	63	7	monitoring, compression, archiving, communication	CSV	LZ77	6	WiFi		gzip, tar, scp	110,40
	config16	128	504	7	monitoring, compression, archiving, communication	CSV	LZ77	6	WiFi		gzip, tar, scp	933,12
	...	...	...	...	...	...	...	...	...		...	...

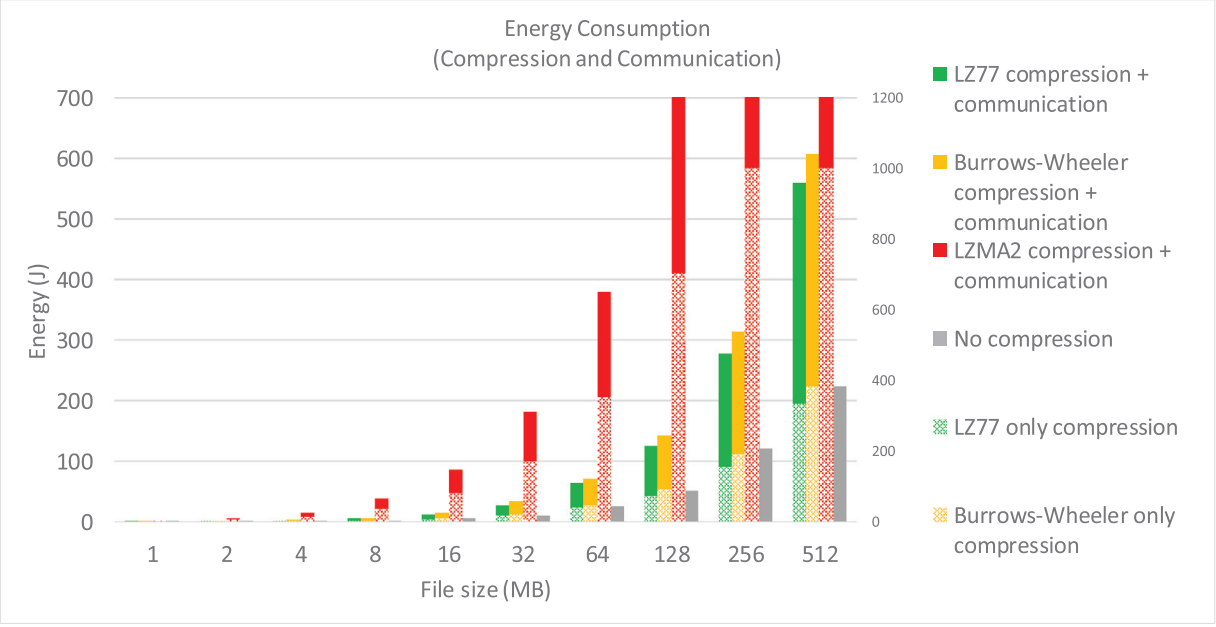

Fig. 4. Energy-efficiency configurations for compression and communication.

This function will be different for each contextual operation and data type of the functionality. By varying the dependent variable (e.g., file size) for each configuration of the implementation characteristics (e.g., different compression algorithms), we can analyze how each configuration influences energy consumption (`Config1`, `Config2`,..., `ConfigN` in the graph in Figure 2). When the energy efficiency context is composed by more than one variable, as is the case of the monitoring or communication functionality, the energy experiments are carried out by varying one variable while maintaining the others fixed. Whatever the approach used to calculate the expected energy consumption, the effort of measuring, estimating and/or simulating the energy expenditure of each of the possible energy consuming concerns would be an intractable task for developers. So, the goal is to save time for application developers by automating as much as possible this manual and tedious job and storing the results in the HADAS Green Repository. The Green Eco-Assistant then helps developers make informed decisions about the energy consumption of the selected concerns, through a sustainability analysis, as next section explains.

7. Analyzing and selecting energy-efficient configurations

The Green Eco-Assistant helps developers carry out a comparative analysis of the power consumption of different solutions for a given runtime energy hotspot. The developer is aware that the decision of choosing a energy consuming concern (e.g., compression) can only be made considering the expected use of the application (i.e., the usage context). It becomes necessary to codify reconfiguration rules (Figure 1, label 2.3) to replace a solution when the current one is no longer the most energy-efficient, under the current usage context. Reconfiguration rules may be described using Event Condition Action (ECA) rules [36], a simple but efficient reconfiguration mechanism that consumes less than other computationally more complex approaches like, for example, optimization algorithms. The event will be a variation in the parameter value that affects the energy expenditure of a given concern (e.g., file size for the compression concern); the condition will be the specific value that makes the current energy consuming concern implementation no longer optimal (e.g., file size greater than a particular value); and, the action will be to replace the current component config-

uration with a more eco-efficient solution (e.g., changing an algorithm by a greener one).

However, the previous reasoning cannot be performed in isolation for each energy-consuming concern, because reducing the energy of one concern can have a collateral effect of incrementing the energy expenditure of others. This means that energy consuming concerns have interactions between them. Normally, interactions occurs when an energy consuming concern (e.g., compression) affects the value of a usage context variable (e.g., file size) of another energy consuming concern (e.g., communication). Our approach will help developers jointly reason over different concerns, by showing the graphics with the energy consumption for the entire configuration, as shown in the graph of Figure 4.

To illustrate all these analysis and the reconfiguration rules that can be extracted of the analysis, let us consider the following energy-saving scenarios in the context of the CPS case study:

- **Scenario 1. Different energy-efficient implementations of an energy consuming concern.** In Figure 3 we can see that for a file size less than 2 MB all compression algorithms consume similar energy, so the developer can initially deploy, for example, the Burrows-Wheeler algorithm. But, when this size increases more than 16 MB, then the LZ77 algorithm is greener. Since both the file size and compression ratio depend on what the user needs at each moment, it is not enough to just generate an initial configuration of an energy-efficient application. It becomes necessary to reconfigure the current deployed configuration by another one more energy-efficient (e.g., replace the Burrows-Wheeler algorithm by LZ77). This reasoning can be described with the following ECA rule:
  ECA1: $compression \wedge filesize > 16\,MB \Rightarrow LZ77\ algorithm$
- **Scenario 2. Dependencies between energy-consuming concerns due to the concerns' usage context.** When the communication frequency of the road unit is greater than a day, the generated files need to be archived (i.e., the files are saved together into a single archive) to facilitate their management and perform a unique sending when communication occurs. In this case, the archiving concern needs to be deployed. On the other hand, when the file is sent every day, the archiving concern is not necessary and can be removed/deactivated from the application. The following two ECA rules describe this situation:
  ECA2: $frequency > 1 \Rightarrow archiving\ concern$
  ECA3: $frequency = 1 \Rightarrow \neg archiving\ concern$
  Although the energy consumption of the archiving concern is insignificant, its inclusion affect the energy consumption of others concerns because of the modification of the file size archived with every new file that is archived.
- **Scenario 3. Dependencies between energy-consuming concerns due to their functionality.** Let us suppose that the developer has initially deployed the road unit to compress the files, thinking that compressing the files reduces the energy consumption of sending them to the server. In this case, we need to know the total energy consumption of compressing the file and sending it to the server. Note that different compression algorithms produce compressed files of different sizes, and therefore the energy consumed by the communication concern will be different, depending on the compression algorithm previously used. In this particular scenario (as shown in the bottom graphic of Figure 4), the compression process always consumes more energy than sending the file without compression. This fact depends on the compression ratio of the file that in turns depends on the file type. This means that for parameters files (.csv files) as the considered in our case study, the compressed file size is still too big and the energy wasted in the compression concern cancels out the benefits of sending the file compressed. Note that this situation depends on the type of the file to be compressed, and that for text files instead of parameters files, compression will often be worth it. Thus, the compression concern seems to be dispensable in all configurations when the content file is not text from the point of view of the energy consumption:
  ECA4: $filetype = CSV \ \vee \ filetype = BIN \Rightarrow \neg compression\ concern$
  ECA5: $filtetype = TXT \Rightarrow compression\ concern$
  The difference between Scenario 2 and 3 is that in the latter the interacting concerns are high energy consumers and, consequently, the global energy consumption of the configuration has to take into account the energy consumption of both concerns together.
- **Scenario 4. Mandatory energy consuming concerns.** Under some usage contexts, the presence of an energy consuming concerns can be required despite the fact that its inclusion increases the energy expenditure of the global system, otherwise the behavior of the application will be inconsistent or not working as expected. For instance, the road unit cannot store more than 512 MB — i.e., this is the capacity of the RPi, so compression is still needed when the files are archived during several days before sending them to the server:
  ECA6: $frequency > 1 \wedge frequency \cdot filesize > MAXSTORAGE \Rightarrow compression\ concern$
- **Scenario 5. Order of energy consuming concerns.** The order in which the concerns are applied can affect the energy consumption of the global system. For example, archiving the files during several days and then compressing a bigger archive is not always possible because of the storage limitation of the device, apart from the energy consumption of compressing a huge file. In this kind of situations, compressing the files before archiving them is the only and greener solution:
  ECA7: $frequency \cdot filesize \geq MAXSTORAGE \Rightarrow precedence: compression, archiving$

In the following section, we show a possible implementation of a self-greening application for the CPS and how the specified ECA rules are integrated and use in the reconfiguration mechanism.

8. Energy-aware reconfiguration

How can we implement a self-greening application without overloading the system with heavy-energy monitoring and reconfiguration mechanisms? What elements should be monitored at runtime? How can we analyze the context to enforce a self-greening behavior without complicating the resulting code?

As described in Section 5.1, the greatest challenge is to define a self-greening mechanism that wastes the least amount of energy, so applying burdensome, self-adaptation approaches (e.g., manipulating models@runtime [35]) are not recommended. In addition, since eco-efficient concerns crosscut several application components it makes sense to follow an approach based on Separation of Concerns [15] to implement energy-related concerns separately from the application's functional components, facilitating their replacement at runtime.

Since we need to observe the runtime variation of some parameters, the subject-observer design pattern could be a good option, and the use of events. We have found one solution, which is not intrusive and also eco-efficient, which is Aspect-Oriented Programming (AOP) [15]. With AOP, it is possible to define interception points in the application base code where we want to inject an extra-functional property, like the energy consuming concerns in our case. Before, around or after executing these interception points we can inject code related to self-greening functionality separately from the core application code. Moreover, the injected code can be easily changed at runtime using the weaving and unweaving mechanisms provided by AOP.

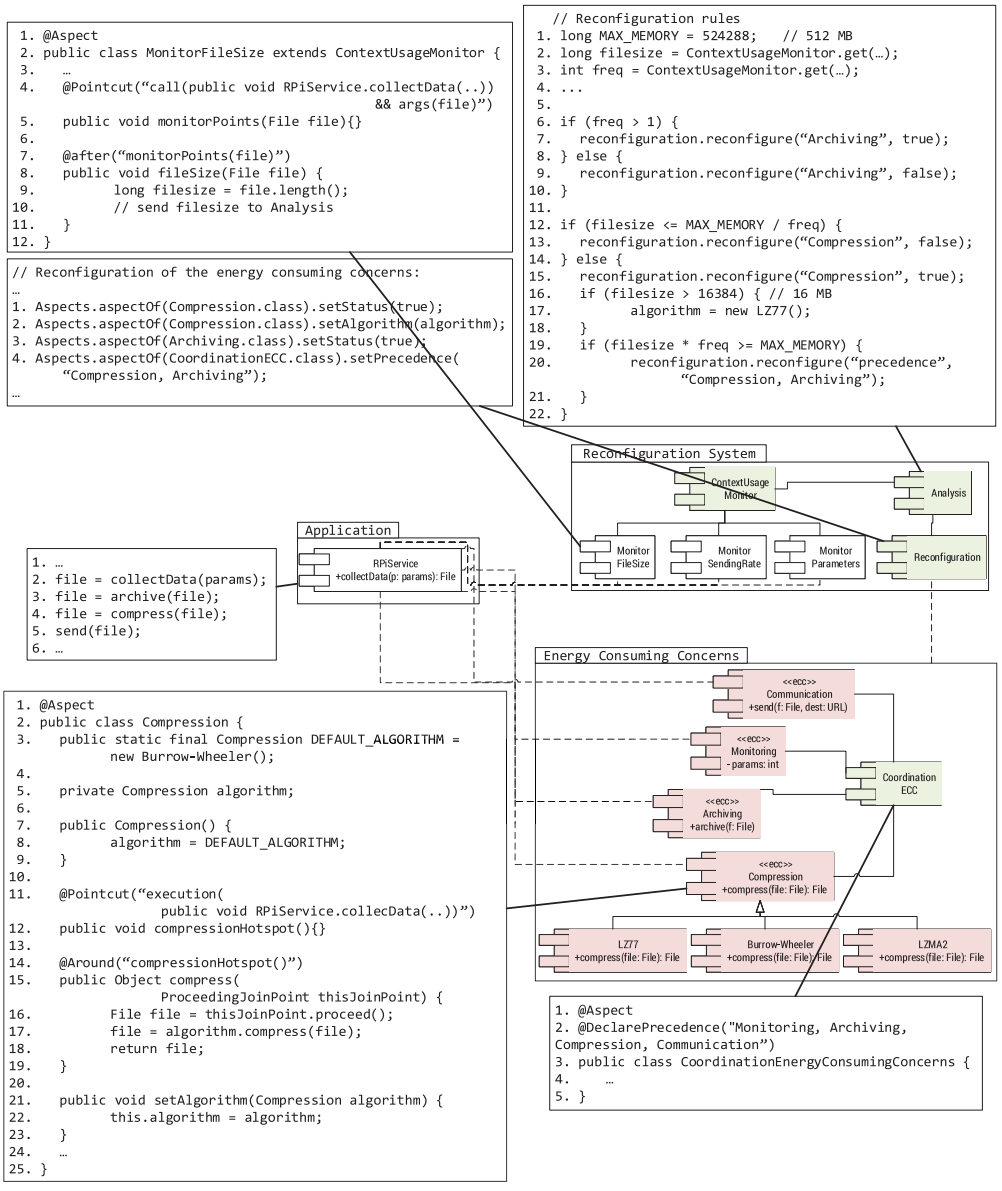

Fig. 5. Reconfiguration using the Green Eco-Assistant.

Figure 5shows an example of an aspect-oriented design solution for implementing self-greening applications in Java and AspectJ [15] (an Aspect-Oriented extension of Java). Three packages can be observed, one representing the application (`Application`) that contains the base code, one representing the reconfiguration mechanism (`Reconfiguration System`) and the last one representing the energy consuming concerns (`Energy Consuming Concerns`). In the context of our CPS, the application code collects the sensor data (`RPiService.collectData()` method), archives the file (`RPiService.archive()`), compresses it (`RPiService.compress()`) and finally sends the file through the network (`RPiService.send()`).

The event monitoring is implemented as part of the `Reconfiguration System` and as separated code, which is then injected into the base code of the application. At runtime, we only need to observe those parameters whose variation implies that the current configuration is no longer the most energy efficient — i.e., these parameters are the events that appeared in the ECA rules defined in the previous step (the file size in our case). So, we propose implementing a `ContextUsageMonitor` aspect for each of the parameters to be observed. The value captured by each monitoring class is sent to the `Analysis` component that contains the ECA rules to decide whether or not a reconfiguration is needed. If the rules determine that a new configuration is greener, the `Analysis` component will send the new configuration to the `Reconfiguration` component.

The `Reconfiguration` component directly interacts with the energy consuming concerns by enabling/disabling them and reconfiguring their internal behavior. The runtime energy consuming concerns (i.e., components with stereotype <<ecc>> in package `Energy Consuming Concerns`) are also implemented as aspects and are non-intrusively injected into the application code. This provides a light solution in terms of energy consumption (see Section 9) and allows an easier reconfiguration of the energy consuming concerns. As shown in the code of Figure 5, the AspectJ annotations (`@Aspect`) are interpreted at compile time by the aspect compiler that weaves the energy consuming concerns (implemented as "aspects") with the application classes at the bytecode level, so there is no overhead at runtime.

Returning to our CPS, in Figure 5 we have defined the `MonitorFileSize` aspect that monitors the size of the data files collected by the `RPiService.collectData()` method (line 4 in the `MonitorFileSize` aspect). Each new parameter value is sent (line 10) to the `Analysis` component, so the `Analysis` component has information about the most recent activity of the device and thereby makes more accurate decisions. In our example, the `Analysis` considers the file size and the frequency of sending the file to the server (line 2 and 3 in the `Analysis` component). The rest of the code of this component shows the implementation of the ECA rules defined in the previous section. Lines 7-11 correspond to the ECA rule for archiving the files locally when the sending frequency is higher than one day (`ECA2` and `ECA3`). Lines 12-15 implement `ECA4` that activates the compression concern when files to be archived exceed the maximum storage of the device. Finally, lines 16-21 implements `ECA1` that sets the encryption algorithm to the most eco-efficient alternative (line 17) and `ECA7` that changes the order of application (precedence) of the energy consuming concerns — i.e., first compressing and then archiving (line 20). `ECA4` and `ECA5` are omitted in the example for space limitation but they are implemented similarly.

The `Reconfiguration` component will activate and/or deactivate the appropriate concerns, and will change the precedence of the different concerns (lines 1 to 4 in the `Reconfiguration` component). In addition, the `Reconfiguration` component is responsible for changing the current configuration of the activated concern, for example, changing the compression algorithm (line 2). The energy consuming concerns crosscut the base application to inject the appropriate functionality in the correct place. For instance, the `Compression` aspect crosscuts the base application to compress the data file after collecting it (lines 16-18 in the `Compression` component).

9. Evaluation and Threats to Validity

In this section we evaluate our proposal, and discuss the threats to validity and lessons learnt.

9.1. Experimentation

In this section we first discuss the reliability of the experiments conducted to estimate the energy consumption of the different configurations of the energy consuming concerns (Figure 4). All resources are available, so experiments can be replicated with the same results[5]. Then we discuss the internal and external validity of the experimentation. The internal validity examines whether the experiment results are influenced or not by other factors apart from those considered in the experiments. The external validity analyzes whether the results obtained in the experimentation can be generalized or not.

9.1.1. Experimentation Set-Up

The energy consumption results presented in this paper were calculated through experimentation, by using a wattmeter, in particular the *WattsUp? Pro* meter. WattsUp? Pro monitors the electric power in Watts of any given circuit. The experiments were performed on a Raspberry Pi 3 Model B[6], with the Raspbian 8 (Jessie) system (Figure 6). For the compression functionality, we have used the tools *gzip, bzip2*, and *xz*, while for the communication concern we have used the *scp* tool. Although the profiling is done at the device level, we isolate the process functionality we are interested in and build a controlled experiment. The RPi device continuously consumes 0.9 W in the idle state, and 1.9 W with the sensors connected. As result, WattsUp? Pro generates a CSV file with the estimated power (in Watts) for each timestamps (1 s) of the execution of the process, which is repeated 15 times. We have implemented a Python script to automate the use of WattsUp? Pro and of our experiments, which extracts the information from the CSV file and calculates the energy consumption in Joules.[7]

9.1.2. Accuracy of the results

On the one hand, we choose the WattsUp? Pro because it allows measuring the power consuming of different kinds of devices at the hardware level, so we think it is a good choice to measure the energy consumption of CPS. Taking the measurements with hardware-based tools is more precise than estimating the energy consumption with software tools at the code level. Although WattsUp? Pro provides measurement with one second of precision, it is enough for our approach. Note that energy consuming concerns (e.g., compression, encryption) are usually expensive operations, while simple operations that last less than one second have an insignificant energy consumption. On the other hand, we have performed 15 runs for each experiment taking the median of the measurements as representative energy value [37]. Although increasing the number of runs will improve the accuracy of the results, we consider that 15 runs is enough in our approach because our goal is not to calculate the exact values obtained for each different configuration, but we are interested in identifying energy consumption variations and tendencies when the usage context varies. In any case, we consider to increase the number of experiments and runs, including the study of more energy consuming concerns as our ongoing work.

9.1.3. Generalization of results

Results from experimentation may vary due to many factors, principally due to the hardware of the device, but also due to the current usage context, the particular implementation of the energy consuming concerns (e.g., programming language, frameworks), or the connected sensors and peripherals. To mitigate the external threats to validity we have performed a subset of the experiments on a different device (Raspberry Pi with Raspian 9 - Stretch), and have obtained similar results for the energy consumption of the energy consuming concerns. In any case, our approach is independent from the experimental values and can be applied using different sources of data with energy information.

9.1.4. Applicability

For each consuming concerns considered in this paper, Table 2 shows the number of contextual features, implementation (configurable) features, configurations and thus, number of experiments performed, and the average of the execution time required to perform each experiment. Generating all the possible configurations of the energy consuming concerns and performing all the required experiments may sound an intractable task. However, it is demonstrated [34] that the number of contextual features and configurable parameters of the energy consuming concerns that affect the energy consumption is usually low and the total number of configurations and experiments is manageable. For instance, as shown in Table 2, for compression that is the more complicated concern, there are only 6 contextual features and 7 implementation features that can affect the energy consumption. So, the maximum number of configurations, and therefore experiments to be performed is 1620. Moreover, it is important to clarify that the experimentation is done only once, and the results can be reused by

[5] http://www.caosd.lcc.uma.es/research/rsc/Horcas2018-AHN-exp.zip
[6] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[7] The script and the experimental results are available in http://www.caosd.lcc.uma.es/research/rsc/Horcas2018-AHN-exp.zip.

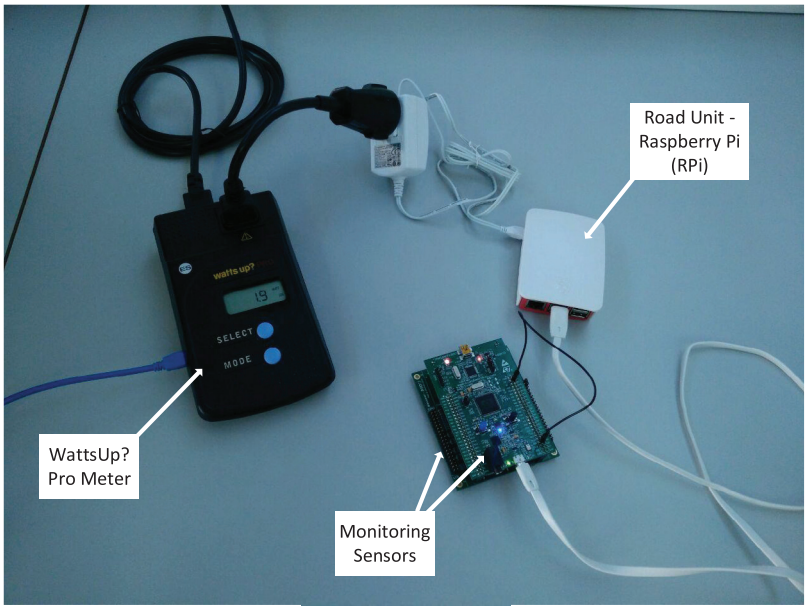

Fig. 6. Experimentation set-up simulating the road unit of the CPS.

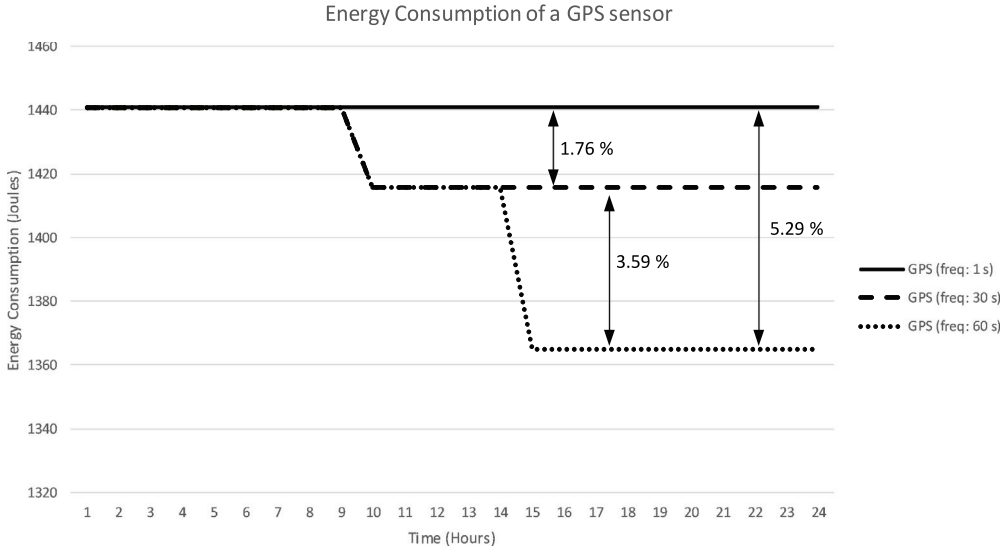

Fig. 7. Energy consumption of the GPS sensor with different configurations.

Table 2
Experimentations over the energy consuming concerns.

ECC	Contextual features	Impl. features	Configurations/Experiments	Avg. Time (s)
Monitoring	2	3	24	11.50
Archiving	2	1	70	5.16
Compression	6	7	1620	158.00
Communication	3	5	420	98.55

the Green Eco-Assistant in multiple applications that require those energy consuming concerns.

9.2. Energy efficiency of the reconfiguration mechanism

As explained in Section 8, we tested our implementation of the reconfiguration mechanism with AspectJ, and the results showed that the energy consumption of the proposed implementation is insignificant compared to the total amount of energy wasted by the energy consuming concerns. In fact, since aspects are woven with the application code at compile time there is not overhead at runtime. In addition, changing the status of an aspect from activated to deactivated or the other way around is an negligible operation the execution time of which is less than one second, and therefore, its energy consumption is practically insignificant. Even though, we plan to perform a practical study, comparing different reconfiguration mechanisms in terms of energy efficiency, as part of our future work.

9.3. Self-adaptation in the context of CPS

Any software system can benefit from applying our approach, including the software part of CPSs as demonstrated in the next

subsection. There are however two issues that need to be considered in the context of CPSs:

- *The processing capability of the CPS nodes.* The physical part of a CPS is normally heterogeneous, with different types of physical components that communicate among them. Not all of them will have enough processing capability for a complex software reconfiguration to make sense. For instance, in a sensor mote that is only gathering sensory information the software part is so simple that the reconfiguration suggested by the Green Eco-Assistant could be only to change parameters' values, e.g., the sampling frequency. However, in nodes with higher processing capability, such as a Raspberry Pi or a mobile phone, our approach can help the CPS to considerably reduce its energy consumption by performing more complex adaptations.
- *The reconfiguration capability of the CPS nodes.* Another thing to be considered is where the reconfiguration mechanism can be allocated at runtime. Firstly, the impossibility of deploying any reconfiguration mechanism in a concrete node needs to be taken into account. For instance, let us suppose a sensor mote with some processing capability, although very limited. This means that it will be possible, for instance, to choose between different data compression algorithms, but without adapting them at runtime. In these cases, our approach is still useful because the design-time sustainability analysis can be done to decide the algorithm with the lowest energy consumption. Additionally, there will be nodes with enough processing capability as to receive a reconfiguration order (e.g. to change its sampling frequency) but without enough resources to allocate the reconfiguration mechanism. In this case the reconfiguration mechanism needs to be running in another node of the CPS (e.g. a Raspberry Pi) in charge of sending the reconfiguration orders to nodes with lower resources. Finally, for those nodes where the deployment of a runtime adaptation mechanism can be afforded, for instance, a Raspberry Pi or a smartphone, there is still the issue of the reconfiguration mechanism to be used depending on the possibilities offered by the operating system and the programming language used in each node. In this sense, the use of AspectJ [15] in Section 8 is for illustrative process and the same approach can be implemented using other separation of concerns approaches such as AspectC [38] (for the C programming language), or an implementation of the injection design pattern [39,40] (for other programming languages).

9.4. Benefits of using the Green Eco-Assistant

To evaluate the benefits to energy efficiency that a software developer can obtain we have applied our approach in two scenarios. In the first one the device is a Raspberry Pi with enough processing and reconfiguration capabilities and we evaluate the benefits that can be obtained when different configurations are considered. In the second scenario we demonstrate how our approach can be sucessfully applied also to improve the battery lifetime in sensor motes without enough capability to deploy our reconfiguration mechanism.

9.4.1. Benefits in CPS nodes with large processing capability

We have chosen a subset of all configurations for the energy consuming concerns in our running example and have compared them in order to check if it makes sense to use our approach to advice developers in finding greener configurations.

Top of Figure 4 shows a set of configurations of the monitoring, archiving, compression, and communication concerns of our CPS case study. For the initial configuration described in Section 4, the road unit is initially deployed with a configuration to monitor 16 parameters, generating a file of 20 MB each day. The file is archived every day, and it is sent to the server every seven days after compressing it, by using the LZ77 compression algorithm — i.e., the greenest according to Figure 3 (config. 13).

At some point the context will change (e.g., when an accident occurs or on summer months) and the road unit will be reconfigured to monitor 128 parameters. This generates a file of 160 MB each day, so archiving the files during seven days after compressing them is not possible because of the storage limitation of the device (see config. 14). This requires to reconfigure the road unit to compress the files before archiving them (see config. 16).

However, according to the results showed in Figure 4, the energy consumption of the new configuration (config. 16) is too high (933.12 J), and thus, a greener solution should be deployed. The sustainability analysis of the eco-assistant helps to realize that a possibility is to change the frequency to send the file every three days instead of seven days (config. 10 with a consumption of 429.29 J). Moreover, there are different greener solutions because the file size affects compression to a greater extent than it does in communication. So, sending the uncompressed file to the server drastically decreases the energy consumption of the global solution (see config. 6 with a consumption of 277.16 J). This means that we need an additional reconfiguration rule that specifies that the compression concern will apply only when the files are archived for several days, to avoid surpassing the storage capacity of the device. Otherwise, sending a parameter file every day without compression is the greenest solution (configs. 1 and 2).

As conclusions, in our CPS example, if the initial configuration of the road unit is maintained when the context changes we miss the opportunity to save around 54% of the energy consumption, since more parameters are monitored and bigger files are produced. Just by reducing the sending frequency of the files from 7 to 3 days to considerably improve the energy-efficiency of the CPS. In addition, deactivating the compression concerns when the capacity storage of the device is enough for the generated files saves around 70% of the energy consumed by the CPS device.

9.4.2. Benefits in CPS nodes with limited procesing capability

To demonstrate the benefits of our approach in those cases when the Green Eco-Assistant makes no sense to be deployed in a CPS node with limited capabilities, we show how our approach can reduce the energy consumption of such kind of devices by reconfiguring them, despite that they do not have enough power to execute the whole reconfiguration system by themselves. In particular, we reconfigure and evaluate the energy consumption of a geolocation sensor (i.e., a GPS sensor) that continuously samples for the location of a vehicle with a configurable sampling frequency.

Figure 7shows the energy consumption of the GPS sensor before and after the Green Eco-Assistant reconfigures it. The Green Eco-Assistant (running in other CPS node with enough capabilities, such as the road unit of our case study) determines the most appropriate configuration for the GPS sensor. In this case, the GPS sensor is checking for the location every second, consuming 1441 J/h. At some point, the context will change (e.g., the traffic flow decreases) and the road unit decides to reconfigure the GPS sensor by changing its sampling frequency every 30 seconds. The new configuration consumes 1415 J/h (1.76% less). When the traffic flow continues decreasing, the GPS sensor will be reconfigured to sample every 60 seconds, reducing its energy consumption to 1364 J/h (3.59% less).

10. Conclusions and Future Work

We have presented a self-greening approach that aims to optimize the energy consumption of applications at runtime. We have focused on those concerns whose consumption depends on parameters that can vary at runtime, according to the usage context

and other contextual information (e.g., available memory, battery level, file size). In order to specify the self-greening rules, we have developed a runtime energy consuming concerns repository with information about relative energy consumption of some recurrent functionalities. The graphics generated by the HADAS Green Repository are used to analyze the possibilities of optimizing energy consumption at runtime. Indeed, we have shown that there are valuable opportunities to optimize the energy consumption at runtime that should not be neglected by developers.

The approach presented in this paper puts the basis to build a dynamic reconfiguration approach for self-greening applications. The approach has been instantiated with 3 different functionalities (monitoring, compression, and communication) and different real implementations with multiple contextual variables. Taking into account the variability of the usage context in cyber-physical systems allows obtaining improvements greeter than 50% in energy consumption. In particular, we have demonstrated that for our CPS system the energy consumption can be reduced up to 70%, depending on the current usage context.

As future work, we plan to complete the evaluation of the Green Eco-Assistant to demonstrate its usage and benefits in real CPS systems, with hundreds of sensors and different kinds of devices. To do so, we will provide a proof of concept with different scenarios of an Intelligent Transportation System (ITS) with multiple sensors whose parameters can be reconfigured with our approach. We also plan to evaluate the energy consumption of different reconfiguration mechanisms for mobile phones, such as different proxy patterns and the *Xposed*[8] framework for modifying code of Android applications at runtime.

Acknowledgments

This work is supported by the projects Magic P12-TIC1814 and HADAS TIN2015-64841-R (co-financed by FEDER funds).

References

[1] Q. Li, M. Zhou, The survey and future evolution of green computing, in: 2011 IEEE/ACM International Conference on Green Computing and Communications, 2011, pp. 230–233, doi:10.1109/GreenCom.2011.47.

[2] C. Stier, A. Koziolek, H. Groenda, R.H. Reussner, Model-based energy efficiency analysis of software architectures, in: European Conference on Software Architecture, ECSA, 2015, pp. 221–238.

[3] K. Grosskop, J. Visser, Identification of application-level energy optimizations, in: 2013 International Conference on ICT for Sustainability, 2013, pp. 101–107, doi:10.3929/ethz-a-007337628.

[4] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, J. Clause, An empirical study of practitioners' perspectives on green software engineering, in: International Conference on Software Engineering - ICSE, 2016, pp. 237–248.

[5] C. Pang, A. Hindle, B. Adams, A.E. Hassan, What do programmers know about software energy consumption? IEEE Software 33 (3) (2016) 83–89, doi:10.1109/MS.2015.83.

[6] E. Jagroep, J.M. van der Werf, S. Brinkkemper, L. Blom, R. van Vliet, Extending software architecture views with an energy consumption perspective, Computing (2016) 1–21, doi:10.1007/s00607-016-0502-0.

[7] A. Noureddine, A. Rajan, Optimising energy consumption of design patterns, in: International Conference on Software Engineering - ICSE, 2015, pp. 623–626.

[8] 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), doi:10.1177/1550147716682738. 1550147716682738

[9] A. Noureddine, R. Rouvoy, L. Seinturier, Monitoring energy hotspots in software, Automated Software Engg. 22 (3) (2015) 291–332, doi:10.1007/s10515-014-0171-1.

[10] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, A. Hindle, Energy profiles of java collections classes, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, ACM, New York, NY, USA, 2016, pp. 225–236, doi:10.1145/2884781.2884869.

[11] S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid, Dynamic software product lines, Computer 41 (4) (2008) 93–95, doi:10.1109/MC.2008.123.

[12] W.L. Hürsch, C.V. Lopes, Separation of Concerns, Technical Report, 1995.

[13] K. Pohl, G. Böckle, F.J.v.d. Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[14] R.E. Filman, T. Elrad, S. Clarke, M. Aksit, et al., Aspect-oriented software development, Addison Wesley, 2004.

[15] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, Aspect-oriented programming, in: M. Akşit, S. Matsuoka (Eds.), ECOOP'97 — Object-Oriented Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 220–242.

[16] R. Chitchyan, C. Becker, S. Betz, L. Duboc, B. Penzenstadler, N. Seyff, C.C. Venters, Sustainability design in requirements engineering: State of practice, in: Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, ACM, New York, NY, USA, 2016, pp. 533–542, doi:10.1145/2889160.2889217.

[17] D. Li, S. Hao, J. Gui, W.G.J. Halfond, An empirical study of the energy consumption of android applications, in: 2014 IEEE International Conference on Software Maintenance and Evolution, 2014, pp. 121–130, doi:10.1109/ICSME.2014.34.

[18] 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, ICSE 2014, ACM, New York, NY, USA, 2014, pp. 503–514, doi:10.1145/2568225.2568297.

[19] B. Ouni, H. Ben Rekhissa, C. Belleudy, Inter-process communication energy estimation through AADL modeling, in: International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012, pp. 225–228.

[20] G. Procaccianti, P. Lago, G.A. Lewis, Green architectural tactics for the cloud, in: IEEE/IFIP Conference on Software Architecture - WICSA, 2014, pp. 41–44.

[21] S. Götz, C. Wilke, S. Cech, U. Aßmann, Runtime variability management for energy-efficient software by contract negotiation, in: Proceedings of the International Workshop on Models@ run. time, 2011.

[22] A.E. Kouche, L. Al-Awami, H. Hassanein, Dynamically reconfigurable energy aware modular software (dreams) architecture for wsns in industrial environments, Procedia Computer Science 5 (2011) 264–271, doi:10.1016/j.procs.2011.07.035. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011)

[23] D. Zeng, L. Gu, H. Yao, Towards energy efficient service composition in green energy powered cyber-physical fog systems, Future Generation Computer Systems (2018), doi:10.1016/j.future.2018.01.060.

[24] 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 (2017), doi:10.1016/j.future.2017.05.029.

[25] S. Chinenyeze, X. Liu, A.Y. Al-Dubai, An aspect oriented model for software energy efficiency in decentralised servers, in: ICT for Sustainability 2014 (ICT4S-14), 2014, doi:10.2991/ict4s-14.2014.14.

[26] D.-J. Munoz, M. Pinto, L. Fuentes, Green software development and research with the hadas toolkit, in: Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings, ECSA '17, ACM, New York, NY, USA, 2017, pp. 205–211, doi:10.1145/3129790.3129818.

[27] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50, doi:10.1109/MC.2003.1160055.

[28] D.J. Munoz, M. Pinto, L. Fuentes, HADAS and web services: Eco-efficiency assistant and repository use case evaluation, in: 2017 International Conference in Energy and Sustainability in Small Developing Economies (ES2DE), 2017, pp. 1–6, doi:10.1109/ES2DE.2017.8015334.

[29] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1990.

[30] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G.K. Olsen, A. Svendsen, Adding standardized variability to domain specific languages, in: International Software Product Line Conference, SPLC, 2008, pp. 139–148.

[31] A. Murguzur, R. Capilla, S. Trujillo, O. Ortiz, R.E. Lopez-Herrejon, Context variability modeling for runtime configuration of service-based dynamic software product lines, in: Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2, SPLC '14, ACM, New York, NY, USA, 2014, pp. 2–9, doi:10.1145/2647908.2655957.

[32] M. Mendonca, M. Branco, D. Cowan, S.p.l.o.t.: Software product lines online tools, in: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA '09, ACM, New York, NY, USA, 2009, pp. 761–762, doi:10.1145/1639950.1640002.

[33] D. Benavides, S. Segura, P. Trinidad, A.R. Cortés, FAMA: tooling a framework for the automated analysis of feature models, in: First International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2007, Limerick, Ireland, January 16-18, 2007. Proceedings, 2007, pp. 129–134.

[34] J.M. Horcas, M. Pinto, L. Fuentes, Variability models for generating efficient configurations of functional quality attributes, Information & Software Technology 95 (2018) 147–164, doi:10.1016/j.infsof.2017.10.018.

[35] R.H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, K. Krogmann, Modeling and simulating software architectures: the Palladio approach, MIT Press, 2016.

[36] G. Blair, N. Bencomo, R.B. France, Models@ run.time, Computer 42 (10) (2009) 22–27, doi:10.1109/MC.2009.326.

[37] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Exper-

[8] http://www.repo.xposed.info/

imentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.
[38] Y. Coady, G. Kiczales, M. Feeley, G. Smolyn, Using aspectc to improve the modularity of path-specific customization in operating system code, SIGSOFT Softw. Eng. Notes 26 (5) (2001) 88–98, doi:10.1145/503271.503223.
[39] D.R. Prasanna, Dependency Injection, 1st, Manning Publications Co., Greenwich, CT, USA, 2009.
[40] E. Gamma, Design patterns: elements of reusable object-oriented software, Pearson Education India, 1995.