Self-Adaptive Energy-Efficent Applications: The HADAS Developing Approach

Jose-Miguel Horcas, Mónica Pinto, Lidia Fuentes
Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga, CAOSD Group
Málaga, SPAIN
{horcas, pinto, lff}@lcc.uma.es

Nadia Gámez
Universidad Internacional de la Rioja
La Rioja, SPAIN
nadia.gamez@unir.net

***Abstract*—Software systems have a strong impact on the energy consumption of the hardware they use. For this reason, software developers should be more aware of the energy consumed by their systems. Moreover, software systems should be developed to be able to adapt their behavior to minimize the energy consumed during their execution. This paper illustrates how to address the problem of developing self-adaptive energy-efficient applications using the HADAS approach. HADAS makes use of advanced software engineering methods, such as Dynamic Software Product Lines and Aspect-Oriented Software Development. The main steps of the HADAS approach, both during the design of the application and also at runtime are illustrated by applying them to a running case study.**

Keywords—energy-efficient applications, self-adaptation, HADAS, Dynamic Software Product Lines, Aspect-Oriented Software Development

I. 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]. Although software systems do not directly consume energy, they strongly affect the energy consumption of the hardware [2]. So developers should be more aware of the energy consumed by these systems during their usage, and try to develop energy-efficient applications that adapt their behavior to minimize the energy consumed during their execution, i.e., develop self-greening applications [3,4].

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 [3,4], principally due to a lack of appropriate methodologies and tools which would help them to produce self-adaptable green software at runtime. Therefore, although software energy efficiency is becoming increasingly important, development processes of self-greening systems supported by tools are still in their infancy. There are plenty of approaches that present experimental results about how to optimize energy consumption at design time [5,6,7], but little effort has been made to explore reusable solutions of runtime energy optimizations.

Indeed, once deployed, the energy consumed by a system depends on several factors, determined mainly by the usage context [8]. 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. So, the user behavioral pattern impacts very strongly on the final energy expenditure of applications. Therefore, applications should not only be prepared at design time to be energy-efficient; they also need to be self-adaptable to the runtime context usage.

This paper illustrates how advanced software engineering methods, such as Dynamic Software Product Lines (DSPLs) [9] and Aspect-Oriented Software Development (AOSD) [10], can help address the problem of developing self-adaptive energy-efficient applications. Concretely, we present the HADAS approach for the analysis and development of self-adaptive energy-efficient applications. HADAS proposes to collect energy-related information at design time and use it at runtime to adapt the application behavior to the real energy consumption. HADAS bases on the concepts of *runtime energy hotspot* and *energy consuming concerns.* A *runtime energy hotspot* is a point in the application that under certain conditions can consume much energy and, if these conditions change at runtime it is possible to reduce this energy consumption by modifying the application components. The *energy consuming concerns* are the concerns that model the runtime energy hotspots at design time. They could be designed in different ways, with different energy consumption that depends on some input parameters such as size of type of data. All the alternative design solutions for every energy consuming concern are stored in HADAS so that at design time application developers can perform a sustainability analysis of the different variants. HADAS then generates the initial application configuration. This sustainability analysis will also help to identify those situations where the energy expenditure strongly depends on some parameters that can vary at runtime. This information will be used by the developer to specify the self-greening rules that will trigger a reconfiguration at runtime.

After this introduction, in Section II we discuss the main challenges that arise in the development of our approach. Then, in Sections III, IV and V we describe how HADAS addresses these challenges. Finally, our conclusions are presented in section VI.

II. RELATED WORK

The software developer community is starting to pay more and more attention to the energy-efficiency concerns. Here we summarize some representative works.

Empirical studies. Recent empirical studies [3,4] 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 application 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 [**Error! Bookmark not defined.**]. 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 [**Error! Bookmark not defined.**].

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]. 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 [8].

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 [5]. These works consider energy consumption 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 tradeoff between energy efficiency and other quality attributes. 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 [14]. The experimental part of these works consists of checking at the code level the effects of applying specific design or architectural patterns [14].

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 [**Error! Bookmark not defined.**], and others build real-time profiles of energy consumption [11]. Moreover, there are examples of the dynamic reconfiguration of energy aware software in different domains. For instance, [12] presents DREAMS, a Dynamically Reconfigurable Energy Aware Modular Software architecture for sensor networks. None of them defines a generic and reusable approach as we make.

III. CHALLENGES

This section identifies the main challenges that arise in the development of self-adaptive energy-efficient applications:

Challenge 1 (C1): Providing the means to identify runtime energy hotspots, i.e., to identify the code pieces that consume more or less energy depending on the dynamically varying contexts. However, recent empirical studies [3,4] show that software developers need help to identify such energy hotspots. There have been recent studies that propose some green computing practices [8], however developers do not know how to apply them in their developments. The main conclusion of these studies is that software developers need more precise evidence about how to tackle the energy efficiency problem and some methodological and tool support to help them effectively address it [3,4].

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 compression energy hotspot, each compression algorithm could consume a different amount of energy depending on the compression ratio and the file size. Thus, after identifying the energy hotspot, software developers need to be aware of the variability of the existing solutions, including the parameters that could affect the energy expenditure. Another important challenge is to explicitly define the variability of design solutions that can mitigate the energy consumption according to current user interaction.

Challenge 3 (C3): Predicting the energy expenditure of software energy hotspots at design time could provide hints about the final power consumption of the application. However, the energy consumption highly depends on several factors, and some of them will vary at runtime. So, energy consumption of each variant of the energy hotspots 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 make a sensible eco-efficiency analysis at design time, about the possibilities of optimizing energy consumption at runtime for a given application.

Challenge 4 (C4): The eco-efficiency analysis may result in more than one design solution for a given energy hotspot, each one fitting a distinct 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. 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. There are some related papers that perform dynamic reconfiguration of energy aware software [12], but they are domain specific and do not provide a generic and reusable approach, which we consider developers need.

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.

In the rest of the paper we will discuss how HADAS cope with these challenges using a running case study.

IV. MODELLING RUNTIME ENERGY CONSUMING CONCERNS

Figure 1 presents the HADAS approach. Firstly (label 1), at design time developers have to discover which application requirements may strongly impact the power consumption at execution time, so they can be classified as runtime energy hotspots (label 1.1). Likewise as designers are able to identify which part of the application demands a particular design pattern, they now have to develop the instinct to identify the concrete runtime energy hotspots of their applications. Learning to recognize energy hotspots is absolutely essential and helpful in any energy-aware development process. However, as indicated in the introduction, software developers do not still have the skills to identify these energy hotspots. Additionally, there are not catalogues of runtime energy hotspots, similar to the existing catalogues 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 [12]. So, HADAS 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. store), and the variants they want to explore (e.g. to store data in a local file or in a server). This selection is done through a set of forms provided by HADAS (label 1.2).

The concerns that model the runtime energy hotspots at design time can be considered as energy consuming concerns, 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) [13], so it would be 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. Since these concerns are common to many applications we propose storing them in the HADAS Green Repository ready to be reused (label 1.3).

There are plenty of studies showing that there is a high variability of alternative implementations and design solutions to many energy consuming concerns [6,7,8], and some of them permit their replacement at runtime to achieve energy savings. For this reason, HADAS follows a DSPL approach [9] to explicitly model the variability of energy consuming concerns, using a variability model, concretely CVL [14]. The motivation behind the use of CVL is that it easily maintains connections between energy consuming concern variants and the set of component models that implement this variant. The top of Figure 2 shows an excerpt of the HADAS variability model with some energy consuming concerns like Store, Communication, Compression or Security. We focus on data compression, one concern present in a Media Store (MS) application used as the case study. For the compression concern we include several algorithms that consume more or less energy depending on the file size, which usually varies at runtime.

What the developer needs to know at design time are the options that exist to address a concrete 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. For HADAS, the concrete number of joules consumed by different energy consuming concerns considering specific hardware is not so important, although the relative energy is, to identify energy consumption trends. So, the intention of HADAS 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, store them in the HADAS repository and provide advice based on these results.

The energy consumption shown in this paper was experimentally calculated, but we have also explored the use of the Palladio toolset [15], an IDE perfectly well suited for predicting, through simulation, the energy consumed by an architecture design. Indeed, the component model shown in Figure 2 is based on the Palladio Component Model. 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 of HADAS 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 repository.

Returning to the MS example, the energy consumption for each audio codec variant was experimentally calculated. At the bottom right of Figure 2, we show the power consumption graphic for compressing 9 WAV audio files of different sizes (from 4Mb to 512 MB) using the following audio compression algorithms implemented in Java: Java LAME 3.99.3 to create MP3 audio files using a bit rate of 128Mb, Vorbis-java (libvorbis-1.1.2) to compress in OGG files, and Java Speex Encoder v0.9.7 to compress in SPX files. The energy expenditure is measured with JouleMeter, a Microsoft tool that measures the energy of software applications running on a computer. We repeated each experiment several times and took the median in Joules that appears in the graphic. This tool has been calibrated using Watts'Up to obtain the real power consumption depending on each hardware component. All the experiments were conducted on a Desktop PC with Intel Core i7 CPU, 3.4GHz, 16 GB of RAM under Windows 10, 64 bits We have implemented a Python script to automate the use of JouleMeter in our experiments (the script is available on http://150.214.108.91/horcas/energy-meter.)

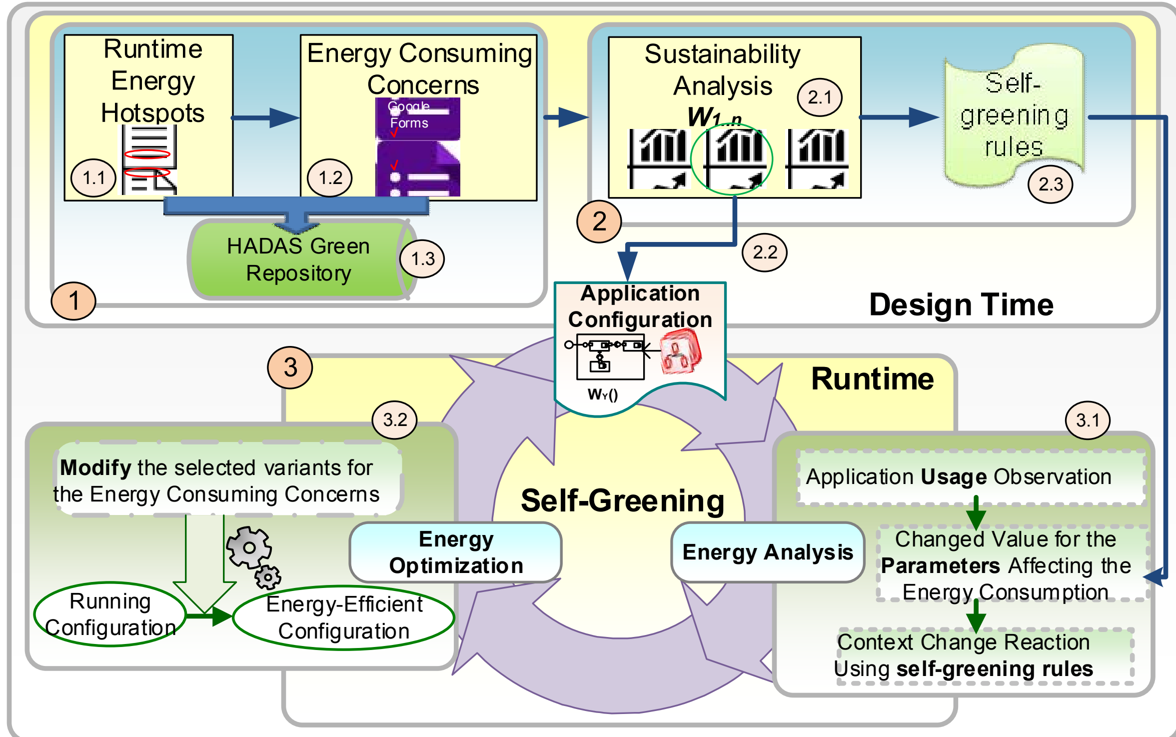

Figure 1. The HADAS Approach

Now, imagine that we wish to calculate the energy of sending a compressed file through a TCP socket. This entails two different concerns, the compression and the communication concerns, which have a clear dependency between them. Note that by compressing the file, the number of bytes sent through the network is lower. In this case, HADAS should help the developer decide whether compressing and sending the compressed file through the TCP socket consumes more or less energy than sending the file without compression. In order to do this kind of reasoning, HADAS formally specifies the dependency relationships between energy consuming concerns using the cross-tree constraints supported by the variability model. For example, the Remote Storage concern depends on both the Communication and the Compression concerns, so we define a constraint associated with the Remote feature as: Remote implies Compression and Communication (Figure 2). With HADAS, designers do not need to be aware of the inter-dependencies between the concrete solutions of different energy consuming concerns. HADAS will enable and disable variants of different hotspots automatically, as the designer selects the desired options. In the example, HADAS automatically incorporates the Communication and the Compression concerns because they are also energy consuming. HADAS then helps developers make informed decisions about the energy consumption of the selected concerns, and generate the initial application configuration (Figure 1, label 2.2).

V. ANALAZING AND SELECTING ENERGY-EFFICIENT CONFIGURATIONS

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 HADAS Green Repository in the development of self-greening applications is to provide the necessary means to make an energy-efficiency analysis, at design time, about the possibilities of optimizing energy consumption at runtime for a given application (Figure 1, label 2.1). This means that the HADAS Green Repository can be used to see whether it is worthwhile specifying a reconfiguration rule to replace, at runtime, a specific concern implementation with another after, for instance, a drastic change in user behavior. So, the HADAS toolkit helps developers carry out a comparative analysis of the power consumption of different solutions for a given runtime energy hotspot. For example, in Figure 2 we can see that for a file size equal to 4MB all the codecs consume similar energy, so we can deploy the LAME codec, but when this size increases up to 64MB, then the codec Vorbis is greener. Since both the file size and quality depend on what the user needs at each moment, it is not enough to just generate an initial energy-efficient application. 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 use conditions (e.g., audio codec LAME by Vorbis).

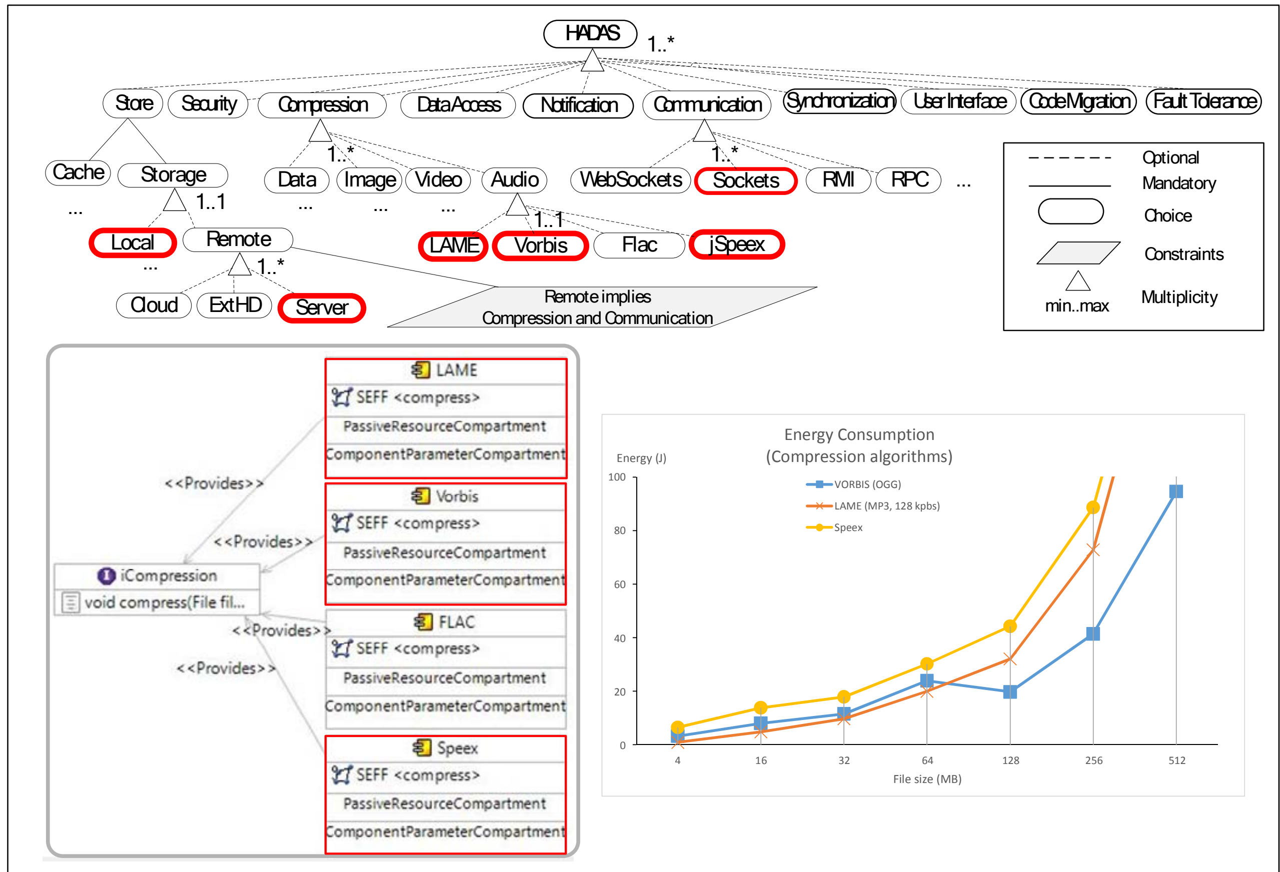

Figure 2. Models of the HADAS Repository and Power Consumption graphic of audio codecs

With HADAS, the developer is aware that the decision of choosing an audio codec can only be made considering the expected use of the application. This reasoning may be described using Event Condition Action (ECA) rules [16], a simple but efficient reconfiguration mechanism that consumes less than other computationally more complex approaches like, for example, optimization algorithms. In our case, the event will be a variation in the parameter value that affects the energy expenditure of a given concern (e.g., file size); the condition will be the concrete value that makes the current energy consuming concern implementation no longer optimal (e.g., size > 64Mb); and, the action will be to replace the current component configuration with a more eco-efficient solution (e.g., replace LAME with Vorbis).

However, this 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. In the MS application, as we have already said, the developer is also interested in exploring the possibility of uploading the audio files to a server. In this scenario, audio files must first be compressed and then uploaded to a server. In this case, we need to know the total power 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. HADAS will help developers jointly reason over different concerns, by showing the graphics with the power consumption for the entire configuration. The configuration is generated according to the dependency relationships previously defined in CVL (Figure 2). For our example, Figure 3 shows the power consumption considering the two concerns used in the remote-server configuration, Compression and Communication. It shows that for a file size of 4 MB, the energy consumption of three audio codecs plus communication is similar, but as the file size increases, the greenest codec is Speex.

With all this information, the developer can now complete the reconfiguration rules for the MS. Since the majority of MS users will store typical song audio files of 4 MB, the developer can select the local feature (i.e., store audio files in the device) and the LAME codec (i.e., the greenest according to Figure 2) for the initial configuration. However, at some point some users may wish to store audio files with a size greater than 64 MB (e.g., a journalist who wants to record an interview), so the greenest codec in this case would be Vorbis.

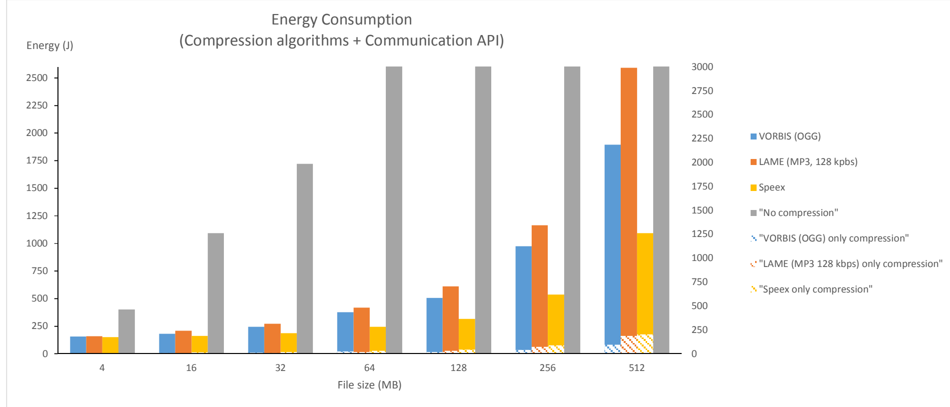

Figure 3. Energy consumption graphic of compression and communication concern

Finally, as the file size increases the device memory becomes full, so it is time to upload the audio files to a server. However, according to the results showed by HADAS (Figure 3), the energy consumption of sending the file to the server increases exponentially in function of the file size, and thus, a greener solution is to replace the compression algorithm with another, with a bigger compression rate, the Speex codec. Note that the Speex codec is the compression algorithm that consumes the most if it is used locally (see Figure 2). There are different green solutions because the file size affects communication to a greater extent than it does in compression. So, reducing the file size as much as possible before sending it to the server drastically decreases the energy consumption of the global solution. This means that we need an additional reconfiguration rule that specifies that if the user or the system decides to upload the audio files to the cloud, the greenest codec is Speex. We have identified three energy saving scenarios at runtime, each one recommending a different audio codec. In the following section, we show a possible implementation of a self-greening application written in Java.

VI. ENERGY-AWARE RECONFIGURATION

Once the initial system configuration has been deployed, the system has to monitor and reconfigure the current system, pursuing true energy efficiency at runtime (Figure 1, label 3). How can we implement a self-greening application without overloading the system with heavy-energy monitoring 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?

Indeed, 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 [15]) is not recommended. In addition, since eco-efficient concerns crosscut several application components it makes sense to follow an AOSD approach [10] to implement energy-related concerns separately from the application's functional components, to facilitate 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 Java events. We have found one solution, which is not intrusive and also eco-efficient, which is the AspectJ language, an Aspect-Oriented (AO) extension of Java. With this language, 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 (i.e., pointcuts in AspectJ terminology implemented as Java annotations) we can inject code related to self-greening functionality separately from the core application code. The AspectJ annotations are interpreted at compile time by the ajc compiler that weaves the "aspect" code with the application classes at the bytecode level, so there is no overhead at runtime.

Figure 4 shows an example of an aspect-oriented design solution for implementing self-greening applications in AspectJ. One possible solution would be to implement the monitoring of events that trigger a reconfiguration as separated code which would then be injected in 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 appear in the ECA rules defined above (the file size in our case). So, we propose implementing a Monitor for each of the parameters to be observed as an aspect, i.e., annotated with @Aspect (stereotyped as <<aspect>>).

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. This component directly interacts with the energy consuming concerns by enabling/disabling them and reconfiguring their internal behavior. The runtime energy consuming concerns are also implemented as aspects (i.e., stereotype <<eco-aspect>>) and are non-intrusively injected into the base application code. This provides a light solution in terms of energy consumption and allows an easier reconfiguration of the energy consuming concerns, because aspects do not call nor are called by the base application.

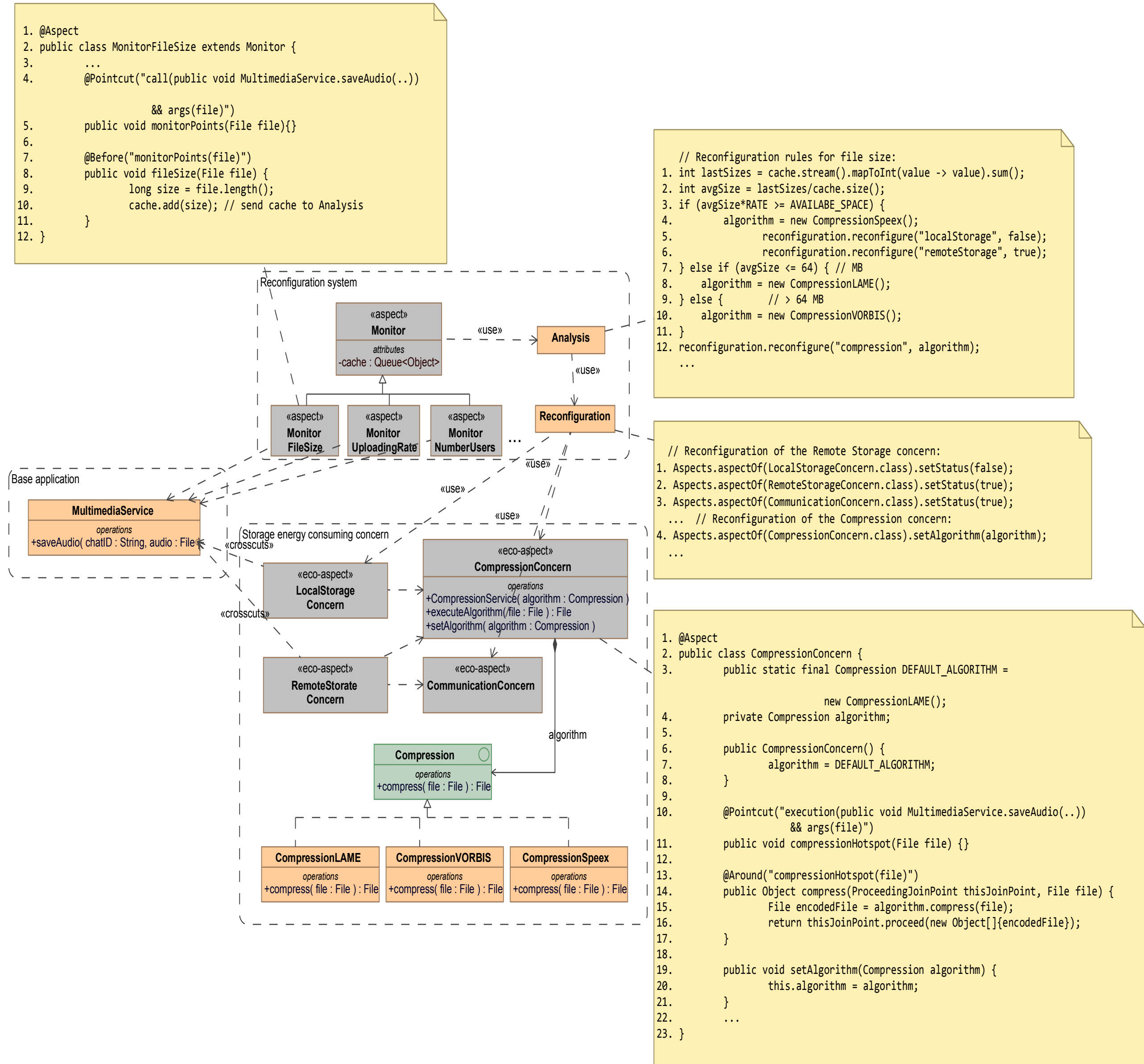

Figure 4. Implementation of the media store using HADAS self-greening approach

Returning to our example, we have defined the MonitorFileSize that monitors the size of the audio files processed by the MultimediaService.saveAudio() method (line 4 in the MonitorFileSize aspect), among others. Each new parameter value is stored in a cache (line 10), the Analysis component can have information about the most recent activity of the user and thereby make more accurate decisions. In our example, the Analysis considers the average size of the latest files processed (line 2 in the Analysis component) in the reconfiguration rules associated with the file size. The rest of the code of this component shows the implementation of the ECA rules defined in the previous step. Lines 7-8 correspond to the ECA rule for files stored locally and with a size less than or equal to 64 MB, setting the LAME codec. Lines 9-10 implement the second ECA rule that sets the Vorbis algorithm for file sizes greater than 64 MB. However, when the space available in the local repository is almost full, the system will be reconfigured to store the files in a remote server and thus, change the compression algorithm for Speex (lines 3 to 6).

The Reconfiguration component will activate and/or deactivate the appropriate concerns, changing from the local to the remote storage configuration (lines 1 to 3 in Reconfiguration component). In addition, it is responsible for

changing the current configuration of the activated concern, for example, changing the compression algorithm (line 4). The energy consumption concerns crosscut the base application to inject the appropriate functionality in the correct place. For instance, the CompressionConcern aspect crosscuts the base application to compress the audio before saving it (lines 10 to 17).

We tested our implementation and the AspectJ mechanism and the results showed that the energy consumption of the proposed implementation is insignificant compared to the total amount.

VII. CONCLUSIONS

We have presented HADAS, 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 could vary at runtime, according to the user interaction or on other context information (e.g., available memory, battery level). 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 concerns. 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. In our example, if the initial codec LAME is maintained and the user starts producing files greater than 64 MB we miss the opportunity to save between 48% (128 MB) and 65% (512 MB). In addition, if audio files have to be uploaded to a server at a certain moment, setting the codec to Speex could save between 52% (difference with LAME) or 43% (difference with Vorbis) for files greater than 64 MB and as the file size increases the saving could reach 81% (difference with LAME) or 54% (difference with Vorbis).

ACKNOLEDGEMENTS

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

REFERENCES

[1] Q. Li and M. Zhou. The survey and future evolution of green computing. In Proceedings of the IEEE/ACM International Conference on Green Computing and Communications, GreenCom'11, pages 230–233, 2011.

[2] K. Grosskop, J. Visser. Identification of Application-level Energy-Optimizations. In Proceedings of the conference on ICT for Sustainability – ICT4S'13, pages 101-107, 2013

[3] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause. An empirical study of practitioners' perspectives on green software engineering. In Proceedings of the 38th International Conference on Software Engineering - ICSE '16, pages 237–248, 2016.

[4] C. Pang, A. Hindle, B. Adams, and A. Hassan. What do programmers know about software energy consumption? IEEE Software, 33(3):83–89, may 2015

[5] K. Grosskop and J. Visser. Identification of application-level energy optimizations. Proceeding of ICT for Sustainability (ICT4S), pages 101–107, 2013.

[6] E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet, "Extending software architecture views with an energy consumption perspective: A case study on resource consumption of enterprise software," Computing, pp. 1–21, 2016.

[7] A. Noureddine and A. Rajan. Optimising energy consumption of design patterns. In Proceedings of the 37th International Conference on Software Engineering - Volume 2, pages 623–626, 2015.

[8] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle. Energy profiles of Java collections classes. In Proceedings of the 38th International Conference on Software Engineering - ICSE '16, pages 225–236, 2016.

[9] S. Hallsteinsen, M. Hinchey, S. Park, and Klaus Schmid. "Dynamic Software Product Lines". Computer 41, 4 (April 2008), 93-95.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and et. al. Aspect-oriented programming. In: ECOOP – Object Oriented Programming, vol. 1241. 1997. p. 220–42.

[11] S. Götz, C. Wilke, S. Cech, and U. Aßmann, "Runtime variability management for energy-efficient software by contract negotiation," in *CEUR Workshop Proceedings*, 2011, vol. 794, pp. 61–72.

[12] A. El Kouche, L. Al-Awami, and H. Hassanein, "Dynamically Reconfigurable Energy Aware Modular Software (DREAMS) Architecture for WSNs in Industrial Environments," Procedia Comput. Sci., vol. 5, pp. 264–271, 2011.

[13] S. J. Chinenyeze, X. Liu, and A. Al-Dubai, "An Aspect Oriented Model for Software Energy Efficiency in Decentralised Servers," in 2nd International Conference on ICT for Sustainability - ICT4S, 2014, pp. 112–119.

[14] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. K. Olsen and A. Svendsen. Adding Standardized Variability to Domain Specific Languages. In Proceedings of the 12th International Software Product Line Conference, SPLC'08, pages. 139-148, 2008

[15] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann. Modeling and Simulating Software Architectures - The Palladio Approach. MIT Press, Cambridge, MA, October 2016.

[16] N. Bencomo, R. France, B. H. Cheng, U. Aßmann (eds.). Models@run.time, LNCS, vol. 8378, pages 279–318. Springer, Heidelberg, 2014