DEEPC: Dynamic Energy Profiling of Components

Samuel J. Chinenyeze, Xiaodong Liu and Ahmed Al-Dubai
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
{s.chinenyeze, x.liu, a.al-dubai}@napier.ac.uk

***Abstract*—Many software projects are built using reusable *components* (i.e. reusable objects - as per *component* and connectors in software architectures). During component selection in CBSD, components are evaluated on the criteria of required quality attribute prior to integration into a system. Current green software research exploring software energy efficiency as a quality attribute adopts conventional counter-based white box energy measuring approach. Although the conventional approach provides results at fine granularity, as with its adoption in component selection, the challenge is that to test software energy of each component, the test has to be done prior to integration, which means implementing multiple counters or multiple versions of the system – thus inefficient, especially when involving much components. In this paper, we present an approach and tool for dynamic energy profiling of components for software systems (DEEPC). The proposed approach employs AOP concepts to expedite the energy measurement of components and improve accuracy, by; i) dynamically loading related components for evaluation (load-time weaving) into the base system, as a way to circumvent manual counter implementation, ii) using pointcuts to facilitate power measurement of loaded components. An evaluation of DEEPC approach presents it to be more time and resource efficient with better profiling accuracy compared to its counterpart.**

Keywords-Component Energy Evaluation; Software Energy Profiling; Dynamic Energy Evaluation; Aspect-Oriented Profiling; Green Software

I. INTRODUCTION

With the increasing impact of IT services on the environment and the emerging need to incorporate green policies into IT processes [1], [2], there has been some progressive work in green software. Green in this context refers to energy-efficiency (EE), or improved energy usage.

To achieve green software in software design process, current research explore EE as a software quality attribute [3]–[5]. Consequently, software energy measurements are in line with existing software testing approaches. For example software energy measurements can either adopt a black box approach or a white box approach [3], [6]. While black box approach focuses on testing the external system features, white box approach involves testing internal features (e.g. code blocks) by use of counters i.e. instrumentation code (thus access to the source code is required). Counters can be implemented either statically (at compile time) or dynamically (at runtime, e.g. the bytecode for the Java classes are instrumented at the time that they are loaded by the JVM). The work on software energy evaluation largely focuses on static instrumentation [3] – involving manual implementation of counters for specific code sections for evaluation. Although beneficial for fine-grained software energy measurement, the counter-based approach is inefficient (development time) for component evaluation process, since it requires several implementation of a set of counters at different code sections being measured. Moreover, our evaluation (section IV) shows that the multiple counter approach is time (execution time) and resource consuming for component evaluation process. Generally, counter-based approach incur the challenge of application and monitor counters being out of sync (further discussed in following section). To the best of our knowledge, software energy measurement have not been well explored within the context of component reuse and dynamic profiling.

Consequently, this paper proposes a time and resource efficient approach known as DEEPC (Dynamic Energy Profiling of Components) based on aspect-oriented programming (AOP) technique. DEEPC aims at providing a less expensive approach for measuring the reusable components of an application at runtime. Through dynamic instrumentation DEEPC approach circumvents manual counters, this is achieved by applying AOP load-time weaving (LTW) on components, and facilitating accurate power monitoring through the use of AOP pointcuts. The benefit of DEEPC in using AOP load-time weaving of components into target system is that multiple components can be evaluated in a single application implementation; thus saving resource, execution and development time. While by using pointcuts to facilitate power monitoring as opposed to counters, consistency and accuracy of profiling results is ensured. Furthermore, as opposed to the use of application counters within code which usually results in code clutters when instrumenting large codebase, DEEPC separates the functional (core) logic from the non-functional logic (i.e. energy profiling process) by use of LTW and pointcuts. In particular, this paper makes the following contributions:

1. DEEPC: An AOP-based approach for efficient software energy profiling, presented in section III.
2. A tool to facilitate the DEEPC approach and boycott the structural limitation of the AOP library adopted, also in section III.
3. An evaluation of DEEPC approach, in comparison with existing counter-based approach, presented in section IV.

The order of the rest of the paper is as follows: motivation, the proposed approach and tool, experiment, related work and conclusion.

II. MOTIVATION

A. Component Reuse and Counter-based Measuring

In software development, reusable components or objects often fulfil unique system functionality. A reusable component of a given functionality can be composed of different counterpart varieties existing off-the-shelf.

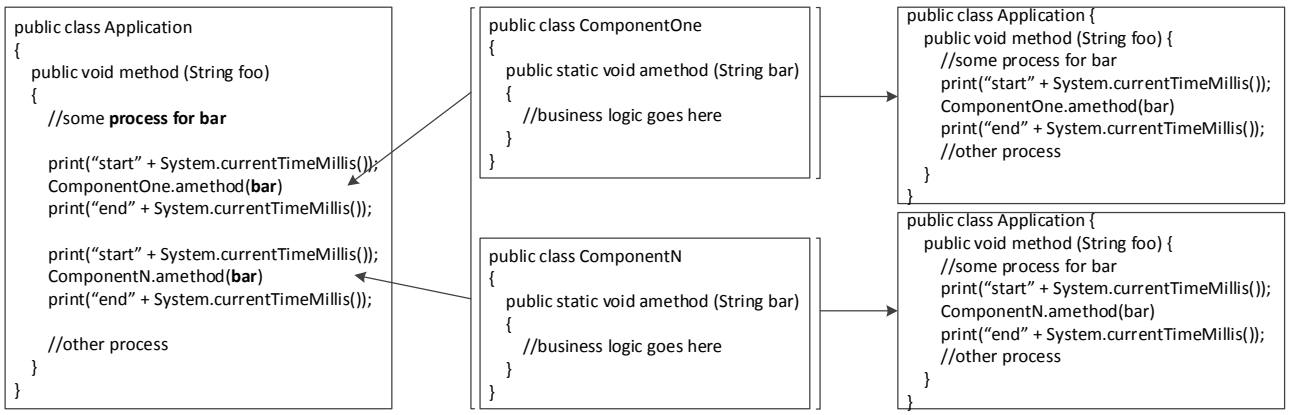

Figure 1. Component evaluation by conventional counter-based measuring.

Consequently CBSD specifies activities for software development (using reusable components) which are fundamentally different from that of the traditional approaches [7]. A unique CBSD activity is component selection which involves sub-activities such as qualification and evaluation of components to ascertain which components (from varying flavors) meet the quality requirements of the target (or base) system.

Current green software research exploring software energy efficiency as a quality attribute adopts conventional counter-based (white box) energy measuring approach [3], [8]. The counter-based approach [3]; involves use of application counters within the source code of an application – specifically at the entry and exit of the segment of code to be measured (i.e. start and end counters in Fig. 1). For example as shown in Fig. 1, Java timestamp utility can be used to obtain a timestamp to mark the start of code execution and the end of the execution – which can be matched to logs from a power monitor for analysis (i.e. estimation of execution time and used energy).

Implementing application counters to be in sync or of the same granularity (of format and log frequency) with the power monitor is a *sine qua non* condition for analysis in counter-based approach. *Granularity of format* deals with coherence of the used timestamps, e.g. if the power monitor logs data in a timestamp of HH:mm:ss:SSS, application counters must be implemented similarly. Whereas, *log frequency* is the rate at which the power monitor logs its data. The smaller the set frequency, the finer the granularity of results – thus improving the accuracy of sync between the power monitor's log counter and application counters. However as the monitor logging and application counters are different processes, there is no guarantee of always perfect sync. The accuracy of results based on counters is dependent on these granularity factors comprising *sync challenge*.

Fig. 1a and b depicts the scenario of implementing the counters in component selection phase of CBSD (in order to ascertain the more EE component). The challenge of Fig. 1a is that the implementation is out of context (*non-contextual challenge*). In other words, evaluating different components at the same time changes the execution context of the application and also gives a false (and increased) energy reading and execution time. Also if a process were to provide parameters for the components prior to evaluation, the outcome of each component for Fig. 1a will be unreliable. Irrespective of dependency on previous (caller) process, components will still be better evaluated within overall runtime context of the system (rather than independent of the target system) as [9] shows that most software induced energy consumption is caused by runtime bloat (from overutilization of resources).

Fig. 1b however will produce a reliable outcome and exact energy readings however multiple versions of the application will be required to be implemented for the evaluation (*multi-implementation challenge*). And given that only energy efficient component will be used (while others discarded), multiple implementation is therefore waste of development time as well as system resource.

In order to address the *sync*, *non-contextual* and *multi-implementations* challenge, illustrated in Fig. 1, we propose the use of AOP concepts – specifically; dynamic crosscutting and dynamic or load-time weaving (LTW).

B. *Why AOP?*

Aspect Oriented Programming (AOP) is a programming technique which provides an effective means for crosscutting of concerns in programs. The concept of code injection at points of execution is one of the core features of AOP. AOP provides two types of crosscutting; dynamic crosscutting – which modifies the behavior of the program using pointcuts, and static crosscutting – which modifies the static structure of the types (classes, interfaces, and other aspects) and their weave-time behavior [10]. In AOP a weaver is used to weave crosscutting concerns into an application in either of two ways; before compilation (i.e. static weaving) or after compilation (i.e. dynamic weaving/LTW) – hence useful for reloading objects during execution.

Thus, application counters which are typically implemented at different code section can be treated as crosscutting concerns and consequently addressed using an aspect. Furthermore, since the purpose of application counters in energy measurement is to compare with power monitoring counters, AOP can be used to boycott the sync challenge by not only replacing application counters with aspects but also facilitating the power monitoring using pointcuts. AspectJ provides a before and after pointcut which can be used in place of start and end counters of Fig. 1, as presented in next section (Fig. 2). Furthermore, the non-contextual challenge (which requires calling different components at various sections of the application), and the multiple implementation challenge, all depicted by Fig. 1, can be curtailed by use of AOP LTW to reload objects at runtime – as shown in DEEPC approach.

III. DEEPC

DEEPC approach (Fig. 2) is derived by addressing the three challenges identified earlier as crosscutting concerns.

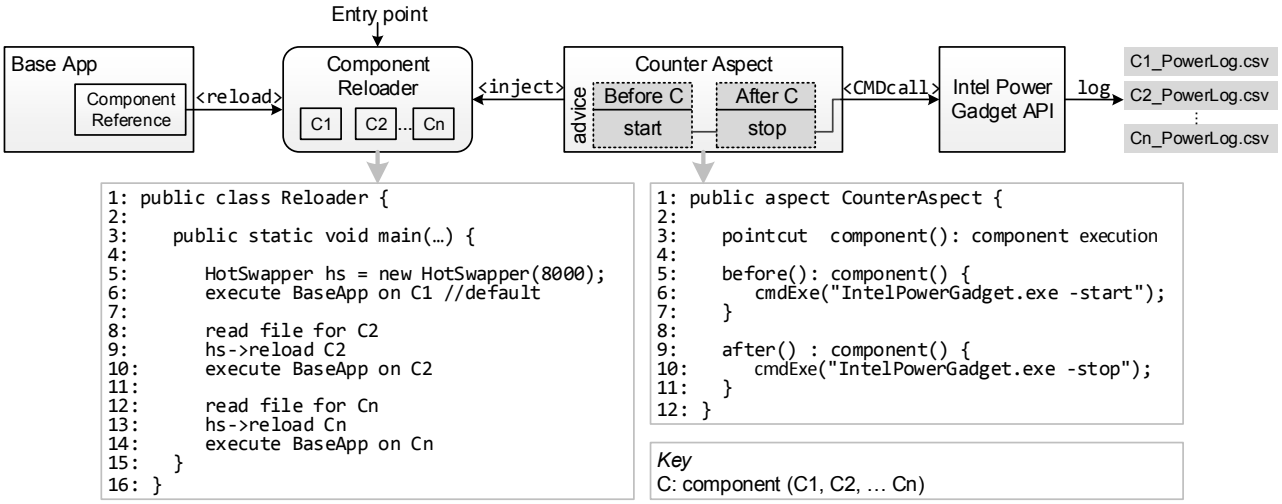

Figure 2. DEEPC Approach.

Thus, AOP APIs such as AspectJ [10] and Javassist [11] are used to implement the DEEPC approach. In DEEPC, AspectJ is used to overcome the sync challenge of 'application to power log' counters by dynamic crosscutting – i.e. through the use of pointcuts rather than counter timestamps. While Javassist is used to dynamically load components to be evaluated, to overcome the challenge of multi-implementation and out of context single implementations. Although AspectJ also can be used for LTW, we found Javassist (Hotswapper class) to be practically more effective and simple to implement (requiring lesser configurations).

DEEPC profiling approach is composed of four main components; *Base App*, Component *Reloader*, *Counter Aspect* and *Monitoring API*. The novelty of the approach is in the simplicity of the architectural setup and the use of AOP for component energy evaluation. Furthermore, a tool has been developed, which uses reflection technique – i.e. ability to change class implementation at runtime – to synchronize classes of different structural compositions (as shown in Algorithm 1). Within the tool (Fig. 3), the components and the Base App is required to be loaded (as source or class files). First the components are loaded and synced. Then the Base App is loaded prior to starting the measurement. When the measurement is started the tool automatically generates the Reloader and CounterAspect classes (shown in Fig. 2), which is then used to start the actual measurement.

Javassist API terminologies used in Algorithm 1: CtClass (compile-time class) object is a handle for dealing with a class file. Similarly, an instance of CtMethod represents a method.

Algorithm 1 Synchronization of Components.

Let *cn* represent a defined name for components
//— Generating Sync List —
1: load all components
2: *d=0*; // the directory for *c*
3: for *c* in components do: // class of component
4: *d*++;
5: get *ctc* from *c* // CtClass of *c*
6: replace class name of *ctc* with *cn*
7: write updated *ctc* to *d* directory
8: add *d* to *dirList* // list of *c* directories
9: for *ctm* in *ctc* declared methods // CtMethod in *ctc*
10: add *ctm* to *methodList* // list of all *c* methods
11: end for
12: end for
13: for *ctm* in *methodList*
14: if frequency of *ctm* in *methodList* != *d*
15: add *ctm* to *toSyncList* // list of methods to sync
16: end if
17: end for
//— Syncing Components —
18: for *d* in *dirList*
19: for *ctm* in *toSyncList*
20: Get *ctc* from *c* in *d*
21: if *ctc* declared methods does not contain *ctm*
22: add *ctm* to *ctc*
23: write updated *ctc* to *c* in *d*
24: end if
25: end for
26: end for

A. Base App

Base App is the target component based application for which components are being evaluated. Thus; the base app may have other application code or not. However, importantly the base app must have a reference to the component being evaluated – this can be method call to the component's class. From Fig. 2 the component reference will be made to the default component C1 (line 6 of Reloader class in Fig. 2). In order to identify C1 as the default component, and also avoid compile time errors, the class for C1 is required to be in the same directory as the base app. The other components will be in uniquely assigned directory – lines 4 and 7 of Algorithm 1.

B. Component Reloader

The Component *Reloader* is the entry point of the evaluation process (i.e. main method in Java). The objective of the reloader is to specify the components which are to be evaluated. The reloader employs Javassist's Hotswap API to perform the LTW of components into the base app. During the initial loading of the base app the component reloader executes the default component normally as a single program. And if further components are declared in the reloader, the Javassist runtime agent first reads the component file, reloads the reference in the virtual machine and executes the *base app* from the point of reference (i.e. the input of the *Entry* text field in Fig. 3).

Currently, the Javassist Hotswapper API requires the components to be of the same class structure; in other words all components must have the same number of methods and method names. To overcome the above structural limitation, DeepC tool (Fig. 3) has been implemented to synchronize components by using reflection to implement empty method bodies from heavier components into lighter components, see Algorithm 1. Heavier components being components which has more

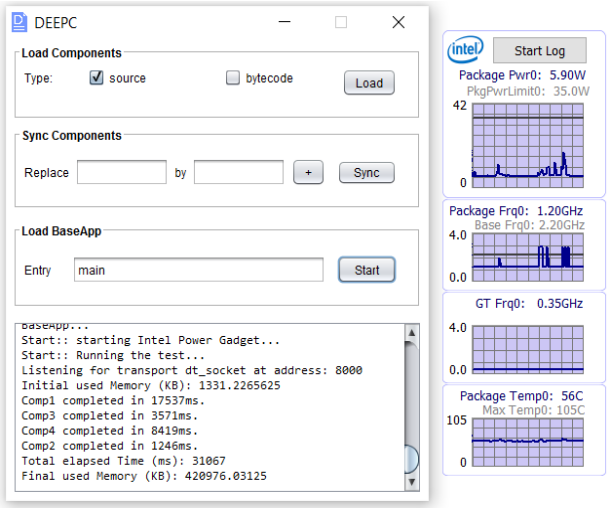

Figure 3. DEEPC Tool.

methods, some of which are different from the lighter components. As shown in Algorithm 1 (line 15), first a list of such methods are generated. Subsequently, the list is used to sync other classes (lines 18-26). The sync is achieved automatically (*Sync* button of tool). And can also be done manually (+ button) by using the *Replace* - *by* text fields of the tool.

C. Counter Aspect

The profiling process works through the use of AOP aspect – implemented using AspectJ – which defines a before and after advice on a dynamic pointcut, for facilitating power monitoring of components. The pointcut references the execution point of the components. Since all components are of similar structure the pointcut applies to all components. Moreover, pointcuts can also be used with wild cards for abstraction.

Intel Power Gadget provides a simple API to start and stop the monitor from command line. The before advice is used to execute the command to start the monitor. The after advice executes the command to stop the monitor. The stop command also saves logs to csv file.

By controlling the monitor using aspect, power data is uniquely logged in accordance to the order in the *Reloader* specification, e.g. first saved log is for component C1, etc. – thus eliminating the need for implementing (named) counters.

D. Power Monitoring API

DEEPC approach uses a monitoring API – Intel Power Gadget (IPG) API [12], [13] as shown in Fig. 3. For any monitored task, IPG provides the elapsed time with actual CPU frequency, wattage, and energy usage readings. By use of monitoring API (rather than external monitor tool) the monitoring process is integrated into the component evaluation process, and thus; eliminating the need for manual counter implementation.

Furthermore, since IPG API computes and writes the energy and execution time of a completed monitoring process to csv files, the need for manual (or post-evaluation) data analysis is eliminated.

IV. Experiment

A. System Implementation

The experiment uses a sorting application as the component based system, and sorting algorithm types – bubble, insertion, selection and merge – as the components being evaluated for the system. Each component sorts 200,000 randomly generated integer values. For individual instance of experiment three parameters are investigated;

- *Distinct Used Energy*, is the energy usage data from IPG log of each component. IPG is CPU-based monitor, and relevant for the sorting experiment; which are computation intensive.
- *Elapsed Time*, is the total elapsed time of each approach measured using Java timestamp facility.
- *Used Memory*, is the memory consumption of each approach measured using Java Runtime API.

The experiment is conducted in three scenarios: i) *DEEPC* is the proposed approach, ii) *CounterS* is the single implementation counter-based approach in Fig. 1a, and iii) *CounterM* is the multiple implementation counter-based approach in Fig. 1b. To reduce bias results we use the command line to run all approaches. Thus; leaving the DEEPC tool evaluation for future work. This is because the tool may contribute additional overall runtime overhead, although not impacting specific component performance.

B. Deployment Settings

The aforementioned scenarios were deployed using the same platform settings, as follows: Windows 10 64bit OS, Intel Core i7 processor and 8GB RAM. A Java JDK 1.8 is used with: 134MB InitialHeapSize, and 2GB MaxHeapSize. IPG power monitor is used with a default log sampling resolution of 100ms. Furthermore, the following factors were considered to control possible threats to validity of results;

Single VM invocation for all scenarios. For each scenario in the experiment we perform multiple runs within a single VM invocation, as a way to focus on steady (not startup) performance and energy of the scenarios.

Standard JIT compilation. JIT compilation is compilation done during execution of a program. For this research, standard mixed mode is used (as it is the default development settings) – which lets the compiler handle necessary optimizations rather than depending on the evaluated program. As JIT compilation is done at runtime, we complement rigor by repeating runs.

Rigorous method of analysis: For a reliable experimental result a rigorous approach is applied for data analysis by use of computed mean from 10 sample readings, with data rounded to 4 d.p. Furthermore, deviation values are also analysed – in following section.

C. Results and Discussion

In this section we discuss the results of our study – presented in Tables I and II. The results are split into two key points of discussion. The first point evaluates the accuracy and consistency of the profiling approaches by studying the energy results of the evaluated components. The second point evaluates the performance and resource efficiency of the approaches based on memory usage and time.

1) Profiling Accuracy and Consistency.

For the counter-based approach, application counters were implemented in the format of the IPG logger, which is HH:mm:ss:SSS and the default logging frequency of 100ms is used for the monitor (i.e. log sampling resolution in IPG). However, despite the consideration of these

TABLE I. RESULTS OF EXPERIMENT

Component	Distinct Used Energy (J)		
	DEEPC	***CounterS***	***CounterM***
Bubble	171.6058	214.1325	213.9645
Insertion	14.5986	14.1734	14.5608
Merge	49.7143	49.6732	51.1224
Selection	93.0567	94.9073	94.4249
Elapsed Time (ms)	28206.7	30336.7	30595.8
Used Memory (KB)	311734.3	325264.6	349557.1

TABLE II. SAMPLE DEVIATION FOR COMPONENTS

Component	Distinct Used Energy (J)		
	DEEPC	***CounterS***	***CounterM***
Bubble	1.2376	1.1399	1.7814
Insertion	0.3032	0.3547	0.9423
Merge	1.5838	3.4219	1.8831
Selection	0.7848	1.6072	0.9943

granularity factors, Table I and II show inaccuracies in the counter-based scenarios.

From Table I the counter (sync) overhead caused as shown in Bubble component is as much as 22% difference (42J), this is a significant overhead considering that other components are within or just over twice the value of this overhead (e.g. Insertion, Merge and Selection). Furthermore, Bubble sort component gives a clear cut demonstration of the inaccuracies in CounterM and CounterS caused by coarse synced counters as opposed to DEEPC. By coarse synced counters we mean that; not only was the application counter non-existent in the power logs, but also, the most fitting (enveloping entry and exit) counters of the logs were considerable much timestamps (milliseconds in our case) apart. For DEEPC approach however, the accurate reading is achieved since the monitor is launched and terminated at the precise entry and exit point of component code, using AOP aspect.

It is worth noting however, that when counters (of application and power log) are in much closer sync, the fine granularity of result can result in accuracy of counter based approach. This is demonstrated in CounterS results for Merge which compared with DEEPC, is only 0.08% difference; thus showing accuracy in CounterS. The mean accuracy however only proves CounterS to contain some accurate samples. In other words, by proper statistical inference, not all Merge samples of the CounterS can be counted as overcoming the sync overhead considering the deviation values of Table II. To expatiate the claim, consider the values: sample standard deviation is 3.4219 for CounterS and 1.5838 for DEEPC, yielding a significant difference of approx. 73.4%. Thus, CounterS sample range is 49.6732±3.4219 (i.e. from 46.2513J to 53.0951J) and DEEPC; 49.7143±1.5838 (i.e. from 48.1305J to 51.2981J). Thus, three points can be deduced;

Point 1: as mentioned earlier, when in a finer counter sync, the counter-based approach (CounterS) can achieve accurate readings. And as shown by the range, these readings could achieve finer precision than DEEPC (reason in point 3), however, this is rare, which leads to point two.

Point 2: as the CounterS shows a wider range of values (by at least ±2 in the example) compared to DEEPC, DEEPC is more consistent in accuracy. This is further demonstrated by the deviation values of Table II, for which DEEPC has a lower deviation for all evaluated components.

Point 3: although DEEPC results are more accurate (as it excludes the sync overhead of counter), it incurs a load time weaving overhead (i.e. from Reloader component of DEEPC). Hence, the reason for the difference between the lower bound values of CounterS (46J) and DEEPC (48J). This yields 4.3% difference overhead from DEEPC, which is however, almost annulled by 3.4% overhead of CounterS, considering the upper bound values.

2) Performance and Resource Efficiency.

As shown in Table I, DEEPC approach has a better performance and memory usage in comparison to the counter-based counterpart. From the sample; DEEPC presents approx. 7% performance improvement/time savings and from approx. 4% to 11% memory savings compared to counter-based approach. Also, based on the observations on the profiling accuracy and consistency, we can deduce that DEEPC is also more energy efficient compared to its counterparts.

Furthermore, with reference to the elapsed time and used memory columns, next to DEEPC in the efficiency trail is CounterS before CounterM. This is because DEEPC architecture is similar to that of CounterS which are both single implementation except that CounterS is constrained by context and counter granularity, whereas for DEEPC is LTW overhead – which is insignificant compared to the overhead of counter inconsistencies, as shown earlier.

CounterM which is not constrained by context (explained in motivation section), is most inefficient of the three scenarios due to the separate startup of components. The total elapsed time of CounterM is obtained by summing up the individual component elapsed time. In addition to being resource inefficient, CounterM approach demands more development time – less productivity. Independent component evaluation (CounterM) approach, could be beneficial in a case where components are immensely different in structure and composition, and there is no target system to integrate components.

D. Limitations

Structural limitation of Javassist library used (addressed by the DEEPC tool). In our approach the Reloader component is restricted to components of the same structure – as required by Javassist Hotswap API. The structure refers to the class name, method names and number of fields. This requirement was also considered in the sorting experiment. For example; all sorting components had similar class structures of three implemented methods – sort(), printArray() and sortName(). Merge however had two additional implemented methods – sortArray() and merge(). Consequently, the extra Merge methods were declared with empty method body for Bubble, Insertion and Selection so as to overcome the structural limitation.

To overcome the structural limitation to the applicability of the approach, the DEEPC tool was implemented – which also expedites the profiling process.

The other important considerations to take into account when choosing to implement the DEEPC approach is that it is dependent on power monitor API. We have only tested the approach on a software based monitor (which is a CPU-based monitor, as our example is compute intensive), and therefore cannot speculate the outcome on a hardware based monitor. For example start and stop logging which is used in DEEPC software monitor may or may not incur additional overhead when used in hardware monitor – taking note that a hardware monitor will require a logging server, as illustrated by [3].

V. Related Work

In terms of dynamic instrumentation techniques and approaches, aspect-based profilers are popular in research and practice. For instance, commercial solutions which are evolving and complex in nature employ AOP for crosscutting concerns such as performance, fault-tolerance, transactions and error logging [10], as aspect-based profilers allow for program instrumentation with few lines of code while abstracting the instrumentation code from the main application. Like many existing aspect-based profilers, [14] promotes and builds a flexible and efficient profiler using high-level AOP, however the approach only explores the use of aspects and proposes an inter-advice communication mechanism for enhancing the efficiency of aspect-based profiling. However, load-time weaving is not explored in this case and the approach does not take into consideration instrumentation for energy usage evaluation. [15] proposes a framework (AOP Hidden Metrics) based on AOP to dynamically instrument existing code with some metric. Although the framework is based on dynamic crosscutting of object properties and functional test, it does not fulfill dynamic/LTW weaving of objects. Moreover the framework is useful for profiling applications as a single entity. DEEPC however focuses on evaluating different entities – the components, based on a single target using LTW. Furthermore, [15]'s framework is more congruent to AspectJ than DEEPC, as it implements AOP concepts by introducing extension APIs to the AspectJ base library. We did not explore the AOP Hidden Metrics frameworks or other AspectJ extension frameworks, because their extensions were not germane to the components evaluation goal – moreover, basic AspectJ library sufficed, and would not introduce extra profiling overhead.

[3], proposes a method for software energy profiling (alongside energy efficiency metric) which uses power monitoring API with counters for white box measurement. In this approach, the instrumented application writes the application counters as well as the monitor logs to file. Subsequently, the logs are analysed by a tool based on recorded counters – we show the challenge of this approach in the motivation and experiment. Conversely, [16] proposes Joulemeter to eliminate the need for implementation of counters within the application. This is achieved by pointing the tool to the application's process in execution, and therefore only the entire system is evaluated – black box. This cannot be applied to components evaluation, since there is no way to reload the application for evaluation of other components.

To the best of our knowledge AOP has not been explored in the context of software energy profiling – especially for component evaluation. Consequently, DEEPC approach have been proposed for the improvement of software energy profiling process for component evaluation through use of AOP concepts (dynamic crosscutting and LTW) and APIs (e.g. Javassist and AspectJ) – demonstrated with IPG API.

VI. Conclusions and Future Work

DEEPC has been proposed as a new software energy profiling approach for components which aims to improve the current counter-based approach. DEEPC is presented as a white box measuring approach similar to the counter-based approach. However, our experiment have demonstrated that unlike counter-based approach which are challenged with counter granularity overhead, DEEPC maintains consistency and accuracy of energy profiling results irrespective of observed load-time weaving overhead. Some types of instrumentations are likely to cause dramatic increase in execution time and resource usage, the case studies presented in the paper have shown that DEEPC is more resource efficient and saves evaluation time compared to its counterpart. DEEPC approach is highly beneficial in comparing between components, especially in a situation where there are a large number of components with similar structure. However, in the case of components with dissimilar structures, DEEPC tool; consisting component synchronization algorithm, have been presented to ensure structural coherence of components during evaluation.

Future work for the DEEPC system would include comparing the tool against other GUI profiling tools implementing counter-based approach. Also we envisage evaluating the proposed approach on a hardware monitor to test its robustness.

References

[1] M. Kazandjieva, B. Heller, O. Gnawali, P. Levis, and C. Kozyrakis, "Measuring and analyzing the energy use of enterprise computing systems," *Sustain. Comput. Informatics Syst.*, vol. 3, no. 3, pp. 218–229, Sep. 2013.

[2] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester, "Overall ICT footprint and green communication technologies," *2010 4th Int. Symp. Commun. Control Signal Process.*, pp. 1–6, Mar. 2010.

[3] T. Johann, M. Dick, S. Naumann, and E. Kern, "How to measure energy-efficiency of software: Metrics and measurement results," *2012 First Int. Work. Green Sustain. Softw.*, pp. 51–54, Jun. 2012.

[4] B. Steigerwald and A. Agrawal, "Developing Green Software | Intel® Developer Zone," 2011. [Online]. Available: http://software.intel.com/en-us/articles/developing-green-software. [Accessed: 18-Jan-2016].

[5] B. Zhong, M. F. M. Feng, and C.-H. L. C.-H. Lung, "A Green Computing Based Architecture Comparison and Analysis," *2010 IEEEACM Intl Conf. Green Comput. Commun. Intl Conf. Cyber Phys. Soc. Comput.*, pp. 386–391, Dec. 2010.

[6] E. Capra, C. Francalanci, and S. a. Slaughter, "Is software 'green'? Application development environments and energy efficiency in open source applications," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 60–71, Jan. 2012.

[7] G. Pour, "Moving toward Component-Based Software Development Approach 1 : Introduction 2 : Component-Based Software Development," *Syst. Eng.*, pp. 296–300, 1998.

[8] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 42–49, Nov. 2013.

[9] G. Software, "Software Bloat and Wasted Joules : Is Modularity a Hurdle to Green Software?," no. September, pp. 97–101, 2011.

[10] R. Laddad, *AspectJ In Action: Enterprise AOP with Spring Applications*, Second Ed. Manning Publications Co., 2010.

[11] D. M. Sosnoski, "Java programming dynamics, Part 6: Aspect-oriented changes with Javassist," Mar-2004. .

[12] J. De Vega, "Intel® Power Gadget," 2014. [Online]. Available: https://software.intel.com/en-us/articles/intel-power-gadget-20. [Accessed: 18-Jan-2016].

[13] J. De Vega, "Using the Intel® Power Gadget 3.0 API on Windows* | Intel® Developer Zone," 2014. [Online]. Available: https://software.intel.com/en-us/blogs/2014/01/07/using-the-intel-power-gadget-30-api-on-windows. [Accessed: 18-Jan-2016].

[14] W. Binder, D. Ansaloni, A. Villazón, and P. Moret, "Flexible and efficient profiling with aspect-oriented programming," *Concurr. Comput. Pract. Exp.*, vol. 23, no. 15, pp. 1749–1773, Oct. 2011.

[15] W. Cazzola and A. Marchetto, "AOP - Hidden metrics: Separation, extensibility and adaptability in SW measurement," *J. Object Technol.*, vol. 7, no. 2, pp. 53–68, 2008.

[16] A. Kansal, F. Zhao, and A. A. Bhattacharya, "Virtual Machine Power Metering and Provisioning," pp. 39–50.