GEMMA: Multi-objective Optimization of Energy Consumption of GUIs in Android Apps

Mario Linares-Vásquez[1], Carlos Bernal-Cárdenas[2], Gabriele Bavota[3]
Rocco Oliveto[4], Massimiliano Di Penta[5], Denys Poshyvanyk[2]
[1]Universidad de los Andes, Bogotá, Colombia
[2]The College of William and Mary, Williamsburg, VA, USA
[3]Università della Svizzera italiana, Lugano, Switzerland
[4]University of Molise, Pesche (IS), Italy
[5]University of Sannio, Benevento, Italy

***Abstract*—This tool demonstration describes** GEMMA**, a tool aimed at optimizing the colors used by Android apps, with the goal of reducing the energy consumption on (AM)OLED displays while keeping the user interface visually attractive for end-users.** GEMMA **has been developed as a distributed architecture to ensure scalability. It is composed of a Web-based client and processing nodes that are capable of analyzing multiple requests (apps) concurrently. The underlying approach makes use of power models, color theory, and multi-objective genetic algorithms. The empirical evaluation of** GEMMA **indicated its ability to reduce energy consumption while producing color combinations pleasant enough for the users. Also, a qualitative analysis conducted with app developers highlighted the potential applicability of the tool in an industrial context.**
VIDEO: https://www.youtube.com/watch?v=k-5ReMVwK0c

I. Introduction

Mobile applications are playing a crucial role in everyday peoples' life, and, because of the features they offer, they are becoming more demanding in terms of required computational resources and, ultimately, energy. Reducing apps' energy consumption is important to improve the mobile device battery duration, and avoid undesirable "running out of battery" situations when the device is needed. In recent and past years, researchers have investigated the major causes for energy consumption, including energy bugs [1]–[4] energy greedy APIs [5], [6], and other aspects such as the impact of code obfuscation [7], adds [8], and HTTP requests [9].

Moreover, some hardware components play a major role in the energy consumption such as the GPS, WiFi, phone module, and the device's display. Concerning the latter, there are two categories of displays, the Liquid-Crystal Display (LCD) ones, for which the energy consumption is almost constant and it does not depend on the colors displayed on the screen, and the Organic Light-Emitting Diode (OLED) or Active-Matrix OLED (AMOLED) displays, for which darker colors ensure lower energy consumption. In recent years, some authors have proposed approaches to optimize color palettes for both Web apps [10], and mobile apps in general [11], [12], when running on OLED displays. While such approaches produce solutions successfully reducing energy consumption, the color palettes might not be as appealing as the original one, and in general, might deviate from the developers' original choices. Also, the optimization does not take into account the amount of time different screens are being shown during the app's usage and the resulting designs are not consistent across several GUIs in the same app.

In this paper we describe the architecture, implementation, and usage of GEMMA(Gui **E**nergy **M**ulti-objective opti**M**ization for **A**ndroid apps), a novel tool for generating color compositions for Android apps that reduce energy consumption and are also visually attractive. GEMMA is based on power models, pixel-based engineering, color theory, dynamic analysis, and multi-objective optimization, to produce a Pareto-optimal set of design solutions (*i.e.,* GUI color compositions) across three different objectives: (i) reducing energy consumption, (ii) increasing contrast, and (iii) improving the attractiveness of the chosen colors by keeping the palette close to the original one. Further details about the approach behind GEMMA can be found in a previously published technical paper [13].

In our original research paper [13] the approach was implemented as a command line-based pipeline of Java programs and scripts, that worked in a single thread mode. Hereby, we describe the implementation of GEMMA as a distributed, cloud-based architecture, composed of (i) Web clients from which the user can upload an app with screens that need to be optimized, (ii) processing nodes in which the APK is analyzed, the apps are executed over a pool of emulators using systematic exploration, and in which screens are optimized using the GEMMA's approach; and (iii) a NoSQL engine supporting asynchronous communication between clients and processing nodes. Noticeably, GEMMA's architecture is designed to achieve horizontal scalability with the possibility of replicating processing nodes.

II. GEMMA's Implementation

This section describes GEMMA's underlying approach, architecture and implementation details.

A. Architecture

GEMMA's architecture is outlined in Fig. 1. GEMMA is composed of three main components: (i) the user interface implemented as a Web client, (ii) the Execution Engine (EE) that

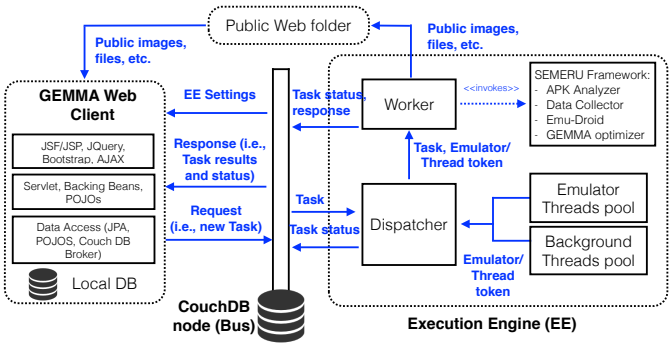

Fig. 1. Architecture of the GEMMA Web Client and the Execution Engine.

executes the requests for GUI optimizations, and (iii) a NoSQL database enacting asynchronous communication between the Web client and the EE. The GEMMA's architecture has been designed with the following design drivers in mind:

- *Asynchronous and decoupled communication* between the clients and the processing nodes. This is in particular important because some GEMMA tasks (*e.g.,* genetic algorithm execution) might be lengthy and computationally intensive;
- *Hybrid storage strategy:* NoSQL document-oriented storage for the requests and EE tasks, and independent relational databases for the client. Images and files generated by the EE, that are rendered/painted in the client afterwards, are stored in a public Web folder;
- *Potential horizontal-scalability*, by clustering/sharding at the DB level (provided by the DB engine), and potential load balancing at the EE level. Both design decisions will allow scalability of the EE;
- *Vertical scalability*, achieved by pools of background threads and emulator threads, and queues of tasks. The latter is used to keep the tasks requests from the clients;
- *Data privacy:* The EE should not store any information of the tasks or the proposed solution after completing the optimization.

We decided to use a "Message Bus" architectural style—without relying on a bus implementation such as OpenESB [14] or a message broker like RabbitMQ [15]—to achieve loose coupling and asynchronous calling between clients and the EE while keeping the possibility of horizontal scalability. In particular, we used a NoSQL database management system, namely CouchDB [16], as the "bus", which contains a queue of requests to be executed by the EE, and a collection of tasks with all their attributes and corresponding statuses. The Message Bus style allows for easy addition of new clients and EEs to the whole infrastructure. Also, CouchDB: (i) provides us with a REST-based interface that allows for easy connection/invocation from any type of client (*i.e.,* it is not coupled to specific languages or drivers), (ii) stores documents in JSON format, which promotes data-model freedom when extending the execution engine to support more processing tasks, (iii) provides replication/clustering features for horizontal scalability, and (iv) is easy to deploy on any operating system.

B. The GEMMA Web Client

Android developers and GUI designers can access GEMMA's features by means of a Web client. The goal of the Web client is to provide users with an easy way to: (i) request the execution of GEMMA tasks in the EE for optimizing the energy consumption of the GUIs in target apps, (ii) explore the Pareto front of solutions generated by GEMMA, and (iii) visualize the differences of the proposed solutions in terms of energy savings, contrast improvement, distance from the original design, and color palettes. A request is a set of attributes describing the optimization task, *e.g.,* APK to analyze, task name, number of GUIs to analyze. Therefore, the requests for optimizations are saved by the Web client in the CouchDB engine as JSON documents. The APKs of each request are stored in the CouchDB engine by using its "attachments" feature [17]. Then, when the request is dispatched by the EE to an internal worker (more details later), the request is removed from the CouchDB engine, and a document is created for the task with all the information generated during the processing (*e.g.,* status, solutions, energy improvements). The Web client checks on demand for updates of the tasks running on the EE, and locally updates the status of the non-finished tasks. When tasks are complete, all task info is synchronized locally (*i.e.,* copied to the local DB) and removed from the CouchDB engine.

As far as technologies used and implementation details are concerned, the GEMMA's client is a Java Web application relying on Java Server Faces, Bootstrap, and JQuery for the presentation layer. The Web client uses a local MySQL database to store the information of the tasks once they are processed by the Execution Engine. The data access layer is realized using the EclipseLink implementation of JPA. The communication between presentation and data access layers is done with a Servlet, backing beans (JSF), and POJOs.

C. The GEMMA Execution Engine

The Execution Engine (EE) is the heart of GEMMA. The EE is in charge of executing the optimization tasks. In particular, the EE: (i) automatically executes the APK on a virtual device by using systematic exploration similarly to what has been done in previous work [18], [19]; (ii) selects the GUIs to optimize according to the number of GUIs requested by the user and an execution time-based heuristic (*i.e.,* the GUIs are ranked according to the number of visits during the systematic exploration)[1]; (iii) analyzes the selected snapshots (*i.e.,* GUI screenshot and GUI hierarchy tree) to identify GUI components, containers, and salient colors on the components; (iv) builds the data structures required for the Genetic Algorithm (GA) execution [13]; (v) executes the multi-objective GA; and finally (vi) saves the results to the CouchDB client and copies screenshots illustrating the suggested color compositions into a folder that can be accessed by the Web client.

[1]Note that conversely to our original approach [13] in which users selected manually the GUIs to analyze, GEMMA Web automatically detects the salient GUIs.

The EE runs as a Java daemon that queries the requests in the "bus", and then dispatches the requests to workers (*i.e.,* the units in charge of running GEMMA's tasks) following a FIFO policy. During the dispatching process, the EE verifies the availability of free Emulators and background threads. If no emulators or threads are available in the EE, the dispatcher keeps the tasks on queue. Otherwise, the task is assigned to a worker that requires one background thread and one emulator. Once a worker is dispatched with a task, an emulator, and a thread, start to run (asynchronously) the GEMMA tasks listed before. During the execution, the workers update the status of the assigned tasks directly to the "bus", and after completion, the generated artifacts (*i.e.,* solutions) are updated in the task document (in the CouchDB engine), and the solution screenshots are copied into a public Web folder. Note that, because there are no synchronous messages between the workers and the Web client notifying when a task is finished, the clients should query the "bus" and synchronize their local databases to reflect the responses/results generated with each task. The workers rely on components (*i.e.,* APK-Analyzer, Data-collector, Emu-droid, and the GEMMA's optimizer) previously developed by the authors for static and dynamic analysis of Android apps and used in previous work [13], [18], [19].

The GEMMA's optimizer, thoroughly described in our previous paper [13]—is based on a multi-objective GA, and namely a Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) [20], implemented in the *jMetal* library [21]. The GA generates Pareto-fronts of solutions while optimizing the following objectives: (i) energy consumption, estimated through regression models, (ii) contrast between adjacent component (a minimum contrast value is also added as problem constraint), and (iii) color distance with respect to the original color composition. The algorithm produces solutions considering, as available colors, the original ones, black, white, and sets of colors achieving equidistant harmony. Noticeably, the fitness function weights the energy consumption of each app screenshot based on the estimated proportion of time it is being displayed during an usage scenario. Also, colors are changed consistently (*e.g.,* all yellow components are painted blue) to ensure a pleasant and consistent result across the GUIs in a target app.

III. GEMMA ON ACTION

After logging, a user can (i) request for a new GEMMA task, (ii) check the status of her requests/tasks, or (iii) check the solutions generated for a finished GEMMA task. GEMMA web client has a form for submitting a new request that includes (i) the name of the "request", (ii) the name of the app, (iii) the APK to analyze, and (iv) the number of GUIs to analyze in the APK. After a request is submitted, the user can check the status of the execution on the GEMMA EE. Fig. 2 depicts the window in GEMMA for listing the requests, which groups finished and unfinished tasks. For example, the list of requests in Fig. 2 shows that the user "Carlos Bernal" has three unfinished requests for the apps *Diabetes Plus*, *Walmart*, and *Fit Brains Trainer*, and one finished task for *Learn Music Notes*. For

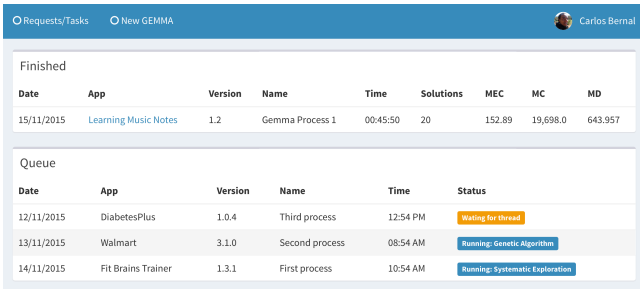

Fig. 2. User requests (*i.e.,* GEMMA tasks) in the GEMMA web client.

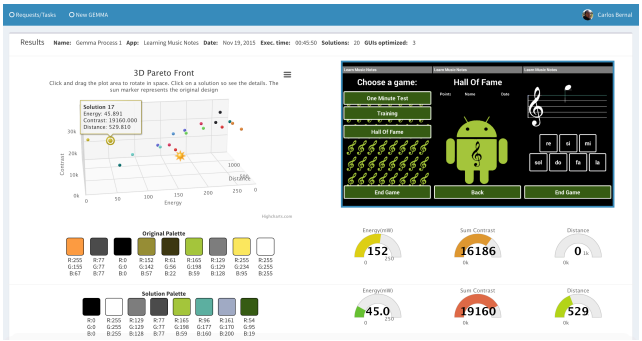

Fig. 3. Dashboard of GEMMA's solutions for the *Learn Music Notes* app.

the unfinished requests, the web client shows that one is still waiting for dispatching (*i.e.,* "Waiting for thread"), another is in execution mode and running the "Systematic Exploration", and the last one is on "Running: Genetic Algorithm". If errors/exceptions occur during processing, the status of the request is set to "Error" with a link to the complete description of the error. The list of requests is updated on-demand, whenever the user refreshes the window.

For finished tasks, GEMMA has a link to the name of the corresponding app, which leads to the details of the generated solutions (dashboard). The GEMMA dashboard, depicted in Fig. 3, includes the following visual artifacts: (i) A 3D Pareto front chart (navigable) with the optimized solutions; (ii) color palettes of the original design and a specific solution selected on the Pareto front; (iii) an image of the visual appearance of the selected solution; and (iv) a gauge-style visualization comparing a solution to the original design of the three optimization objectives (*i.e.,* energy, total contrast of the GUI components, and distance to the original design). The Pareto front (Fig. 4) uses a 3D space to plot the energy consumption of the solutions in the x-axis, the contrast in the y-axis, and the distance to original design in the z-axis. The Pareto front also shows the original design using a sun marker. For example, Fig. 4 depicts the results of a real execution of GEMMA for the *Learn Music Notes* app, in which 20 solutions were generated.

When a user moves the mouse pointer over a solution in the Pareto front, GEMMA displays a contextual window (Fig. 4) listing the values of the solution for energy consumption, contrast, and distance from the original design. In addition, when clicking on a solution, GEMMA updates the other visual artifacts in the dashboard, to plot the information corresponding to the selected solution. The Pareto front can be downloaded as an image. Further details of GEMMA on action can be

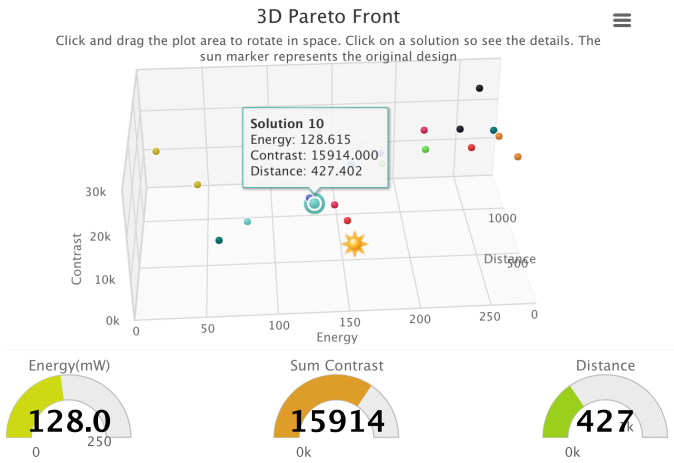

Fig. 4. Pareto Front (top) and gauge style visualizations (bottom) in GEMMA for solutions of the *Learn Music Notes* app.

seen in the video accompanying this paper.

IV. Evaluation Summary

We empirically evaluated the energy savings that could be achieved by running Android apps when adopting the GUI color design recommended by GEMMA (RQ_1), the colorfulness of the GUIs that GEMMA produces as assessed by mobile apps users (RQ_2), and GEMMA's suitability in an industrial context, when applied to minimize GUI energy consumption of existing commercial apps (RQ_3).

RQ_1. We applied GEMMA on 25 apps and we measured the energy consumption, the contrast ratio, and the distance from the original design of the solutions it generated. On average, the solutions having the lowest consumption resulted in a mean energy saving of 66% while also improving the contrast ratio of the original design (+16% on average).

RQ_2. We involved 85 app users asking for their opinion about the look and feel of the original GUIs and of the GUIs generated by GEMMA for the same set of 25 previously mentioned apps. Users expressed a slight preference toward the original design. While this result was quite expected (*i.e.,* minimizing energy does have a cost in terms of visual aesthetics as perceived by users), the solutions generated by GEMMA and representing a compromise between the three objectives (*e.g.,* the solutions having the median value for energy consumption among the generated ones) were quite appreciated by the users.

RQ_3. We conducted semi-structured interviews with the project managers of the three software companies. We ran GEMMA on five apps developed by the three companies and asked for their feedback about the solutions generated by GEMMA. The interviews indicated that the three managers are ready to account for GEMMA's recommendations in future app releases.

Further details about the GEMMA's evaluation are available in our previous research paper [13].

V. Conclusion

We presented GEMMA, a Web-based tool for generating color compositions for Android apps able to reduce the energy consumption while being visually attractive. GEMMA has been implemented as a distributed, cloud-based architecture ensuring high scalability and extensibility.

Results of our evaluation [13] showed GEMMA's ability to generate energy-saving color compositions that are also acceptable from the end users' perspective. Also, a qualitative evaluation conducted with three managers of app development companies, indicated the applicability of GEMMA in a real development context.

Future work will be mainly devoted to the GEMMA's dissemination and extensive evaluation in the Android development community.

References

[1] A. Pathak, A. Jindal, Y. Hu, and S. P. Midkiff, "What is keeping my phone awake? characterizing and detecting no-sleep energy bugs in smartphone apps," in *MobiSys'12*, 2012, pp. 267–280.

[2] Y. Liu, C. Xu, and S. C. Cheung, "Where has my battery gone? finding sensor related energy black holes in smartphone applications," in *PerCom'13*, 2013, pp. 2–10.

[3] J. Zang, A. Musa, and W. Le, "A comparison of energy bugs for smartphone platforms," in *MOBS'13*, 2013.

[4] Y. Liu, C. Xu, S. Cheung, and J. Lu, "GreenDroid: Automated diagnosis of energy inefficiency for smartphone applications," *IEEE TSE*, vol. 40, no. 9, pp. 911–940, 2014.

[5] A. Pathak, Y. Hu, and M. Zhang, "Where is the energy spent inside my app? Fine grained energy accounting on smartphones with Eprof," in *EuroSys'12*, 2012, pp. 29–42.

[6] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy API usage patterns in Android apps: an empirical study," in *MSR'14*, 2014, pp. 2–11.

[7] C. Sahin, M. Wan, P. Tornquist, R. McKenna, Z. Pearson, W. G. Halfond, and J. Clause, "How does code obfuscation impact energy usage?" *Journal of Software: Evolution and Process*, vol. 28, no. 7, pp. 565–588, 2016.

[8] J. Gui, D. Li, M. Wan, and W. G. Halfond, "Lightweight measurement and estimation of mobile ad energy consumption," in *Proceedings of the International Workshop on Green and Sustainable Software*, May 2016.

[9] D. Li, Y. Lyu, J. Gui, and W. G. Halfond, "Automated energy optimization of http requests for mobile applications," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, May 2016.

[10] D. Li, A. H. Tran, and W. Halfond, "Making web applications more energy efficient for OLED smartphones," in *ICSE'14*, 2014, pp. 573–538.

[11] M. Dong and L. Zhong, "Power modeling and optimization for OLED displays," *IEEE TMC*, vol. 11, no. 9, pp. 1587–1599, 2012.

[12] M. Wan, Y. Jin, D. Li, and W. G. J. Halfond, "Detecting display energy hotspots in Android apps," in *ICST'15*, 2015.

[13] M. Linares-Vásquez, G. Bavota, C. E. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Optimizing energy consumption of GUIs in Android apps: a multi-objective approach," in *ESEC/FSE'15*, 2015.

[14] "Openesb. http://www.open-esb.net."

[15] "Rabbitmq. https://www.rabbitmq.com."

[16] "Apache couch db. http://couchdb.apache.org."

[17] "Couchdb wiki - attachments. https://wiki.apache.org/couchdb/HTTP_Document_API#Attachments."

[18] M. Linares-Vásquez, M. White, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, "Mining Android app usages for generating actionable GUI-based execution scenarios," in *MSR'15*, 2015, pp. 111–122.

[19] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for Android applications," in *ESEC/FSE'15*, 2015, pp. 673–686.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective Genetic Algorithm: NSGA-II," *IEEE TEC*, vol. 6, no. 2, pp. 182 – 197, 2002.

[21] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, 2011.