ABSTRACT
Given the limited tool support for energy-saving strategies during the design phase of android applications, developing battery-aware, location-based android applications is a non-trivial task for developers. To this end, we propose eGEN, consisting of (1) a Domain-Specific Modeling Language (DSML) and (2) a code generator to specify and create native battery-aware, location-based mobile applications. We evaluated eGEN by instrumenting the generated battery-aware code in five location-based, open-source android applications and compared the energy consumption with non-eGEN versions. The experimental results show 188 mA (8.34% of battery per hour) of average reduction in battery consumption while showing only 97 meters of degradation in location accuracy over three kilometers of a cycling path. Hence, we see this tool as a first step in helping developers write battery-aware code in location-based android applications. The GitHub repository with source code and all artifacts is available at https://github.com/Kowndinya2000/egen, and the tool demo video is available at https://youtu.be/Iadfh4cCw8I.

CCS CONCEPTS
• Software and its engineering → Development frameworks and environments; Domain specific languages.

KEYWORDS
domain-specific language, code generator, energy-saving location-sensing, android applications

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ESEC/FSE ’22, November 14–18, 2022, Singapore, Singapore
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9413-0/22/11...$15.00
https://doi.org/10.1145/3540250.3558914
The subject applications were instrumented using the generated battery-aware code and executed on the Nokia C3 smartphone (with a battery capacity of 3040mAh) running on the Android 10 platform. Overall, the eGEN version of the subject application shows a reduction of 188 mA in battery consumption and 97 meters of degradation in location accuracy. The reduced battery and GPS usage with negligible degradation of location accuracy by the generated code show initial promise and the need for further research.

2 EGEN DESIGN AND DEVELOPMENT

2.1 Overview

The essential idea of energy-saving in location-sensing is to enforce energy-saving policies in the following situations: (1) when the battery is discharging, and the battery level is critical, and (2) when the app is running in the background. Therefore, eGEN is designed to assign appropriate values for critical battery level, and location-sensing interval based on the application requirements. eGEN has been developed, with the help of Xtext and Xtend. As shown in Figure 1, the usage of eGEN consists of seven steps:

1. The Eclipse editor is used to specify the energy-saving location-sensing policies using the textual domain-specific modeling language.
2. The editor creates the .egen model.
3. The validator module of Xtext checks whether the .egen model is following the eGEN DSML grammar.
4. The code generator takes the validated .egen model as the input.
5. It then generates the Java code using model-to-text transformation.
6. The generated code can be added to existing android applications to make it energy-aware.
7. The updated android project can be built and installed on the android device by the user.

2.2 DSML Grammar and Code Generator

According to the grammar shown in Figure 2, .egen model starts with the definition of self-adaptive location-sensing policies. Each adaptation policy requires domain-specific elements such as context information and application features (lines #4-5). Each context and feature specification has its own allowed range of values. With respect to context information, the BatteryLevel might have the values High, Medium, Low (lines #31-32). The domain analyst can set a numerical value for battery levels using Threshold_High and Threshold_Medium (lines #34-38). The BatteryState can have the values charging and discharging (lines #28-29) and AppState can have the values foreground and background (lines #40-41).

With respect to features, the SensingInterval (in milliseconds) and DecreasingFactor can have values depending on the applications’ requirements (See line #43-46). Further, the BatteryAware function might have the values exponential or linear (lines #49-50). The domain analyst can set a suitable location-sensing interval for different contextual situations. As shown in lines #4-5, the adaptation policy definition starts with the keyword AdaptationPolicy and a unique ID. Subsequently, the four pieces of context information are defined using the keyword condition. Each context information and its values must be specified (mandatory) and separated by the AND operator. The keyword then is used to separate the block of conditions and the block of features. The keyword Adaptation is used to define three features separated by the AND operator. The code generator uses the following android APIs to achieve self-adaptation: Battery Manager API, Fused Location API, and Android Activity Life Cycle. The code generator is defined by mapping each element in the eGEN DSML to the corresponding android library. The code generator creates Java files that contain the self-adaptive location-sensing artifacts, which can be appended to the existing android projects. The underlying conceptual details of DSML and code generator are beyond the scope of this tool paper.

3 EVALUATION OF EGEN

We have designed an empirical study based on controlled experiments [16] to evaluate the effectiveness of eGEN.
3.1 Study Design

We selected five open-source subject applications from F-Droid\(^4\) and Google Play Store\(^5\), that are suitable for instrumentation and conducting controlled experiments.

The first and second authors have investigated the subject applications’ source code to identify the modules responsible for location listening activity. We looked for the location API, sensing interval, and support for battery awareness. We found that the subject applications use static sensing intervals. Hence, we defined suitable adaptive location-sensing policies using eGEN. We then generated a sample android application with battery-aware location-sensing code based on these policies. We instrumented the generated code to the subject applications for battery awareness. We have selected Nokia C3 as a test device for conducting the experiments. It comes with a 5.99-inch display, and its hardware packs 3GB RAM and 32GB in-built storage with a 3040 mAh battery. This smartphone runs on Android 10 and comes with a cleaner version of android (Stock android), simplifying the process of disabling the system apps to create an isolated environment for conducting controlled experiments. All other user applications, including Google Play Store, have been disabled during experiments. Initially, we conducted cycling from the specified source to the destination using Google Fit\(^6\) to measure the total distance. The three trials of distance measurement with Google Fit are depicted in Figure 3. Figure 3[A] shows the direction of a round trip travel. Figure 3[B] and [C] show the duration of the trial and distance covered, respectively. For trial duration that lasted for 12-14 minutes, the average battery drop was found to be approximately between 1.5% to 4.8%.

The distance measured with Google Fit (3060 m) is considered a reference distance used to analyze subject applications’ location accuracy. The same path (as used in the Google Fit trail) was used to perform the controlled experiments for eGEN and non-eGEN versions of the subject applications. We have conducted three trials for both versions of the five subject applications, thus conducting 30 trials each of length of three kilometers (approx.). The generated bug report has been uploaded to the Battery historian for further analysis on completing each controlled trial.

3.2 Experimental Results

The eGEN and non-eGEN versions of the subject applications were executed on the test device. The variables such as BatteryConsumption and DistanceCovered were captured and analyzed. BatteryConsumption has been calculated using Google Battery Historian (bug reports are uploaded here\(^7\) ) and the reported results are the average values obtained for the three trials. The DistanceCovered is calculated from the location coordinates recorded by the application.

3.2.1 Reduction in Battery Consumption: We conducted the trials when the battery level was either low or transitioning from medium to low or high to medium to cover all battery-critical situations. The battery consumed by non-eGEN and eGEN version was compared, and the difference is reported in Table 1. The positive value of Energy Savings in mA implies the significant reduction in battery consumption by the eGEN version. Since the trial time for each experiment varied from 12 to 14 minutes, for each trial for both eGEN and non-eGEN versions of subject applications, we decided to present the battery consumption in mA which is independent of time instead of presenting values in mAhr (a.k.a battery capacity) which are dependent of time. However, for the savings obtained by each subject application, we presented the battery savings in the percentage of battery consumed per hour as per the formula\(^8\).

The results from subject applications demonstrate that the eGEN version shows energy savings ranging from 89 to 269 mA (approx) with a mean reduction in battery consumption of 188 mA. As shown in Table 1, the code generated by eGEN can bring a maximum of 268.76 mA battery savings (in OpenTracks).

3.2.2 Degradation in Location Accuracy: For calculating the distance covered, we collect the coordinates of the previous and current locations and cumulatively add the distances using Haversine formula\(^9\). In Table 2, the distance covered by eGEN and non-eGEN versions of the subject applications is presented. The degrade is calculated by subtracting the distance measured by non-eGEN version from eGEN. In Table 2, the negative degrade values mean less deviation, and positive values signify more deviation in location accuracy. As observed from Table 2, OSMTracker showed equal distance measurement when compared to that of non-eGEN while resulting in battery savings of 149.94 mA. Distinctly, KinetiE-Speedometer brought in a more accurate distance measurement of

\(^1\)https://github.com/Kowndinya2000/egen/tree/master/bug-reports
\(^2\)https://github.com/Kowndinya2000/egen#calculating-battery-consumption-in-percentage
\(^3\)https://www.movable-type.co.uk/scripts/latlong.html

| Subject Application | Distance Covered in km | Degradation in Accuracy in m. |
|----------------------|------------------------|-------------------------------|
| KinetiE-Speedometer  | 2.79                   | -80                           |
| OSMTracker           | 2.990                  | 0                             |
| OpenTracks           | 2.983                  | 0                             |
| RunnerUp             | 2.99                   | 0                             |
| GPSLogger            | 3.05                   | 54                            |
| Google Fit(Benchmark Application) | 3.00km | |
80 meters while reducing battery consumption. The apps such as Open Tracks, RunnerUp, and GPSLogger showed degraded accuracy of about 7, 10, and 54 meters, respectively. Overall, eGEN versions interestingly reported a thin margin of degradation in accuracy while reducing the battery consumption when compared to the non-eGEN versions of the subject applications.

4 DISCUSSION AND LIMITATIONS

Given the use of Xtext in designing DSMLs and Eclipse ecosystem in the context of Model Driven Development[11, 12], we developed eGEN as a plugin for Eclipse. However, it can further be extended to IntelliJ IDE, and Android Studio. The eGEN generated code cannot be used for iOS or Windows smartphone platforms. The generated battery-aware boilerplate code for self-adaptive location sensing in Android applications could be reused as it is written in Java. As we decided to consider a smartphone with stock android, we selected the Nokia C3 smartphone for experimentation. However, we believe any other smartphone can also be selected for conducting trials with the same experimental setup.

The current version of eGEN allows developers to create adaptation policies with explicit trade-off analysis of the target applications’ energy and accuracy requirements. For instance, the AdaptationPolicy 01 ensures high accuracy at the expense of using more battery, and this policy could be applied in scenarios where the battery level is high. Similarly, the AdaptationPolicy 03 tends to minimize battery consumption over achieving accuracy and is triggered when battery levels are low. While eGEN shows an initial promise for considering domain-specific energy savings during the early stages of android development, there is immense scope to improve DSML grammar and code generator. Further, the tool could be evaluated with real-world mobile application developers.

5 RELATED WORK

Energy Estimation Language (EEL) [2] is a textual domain-specific language written using the Xtext language development platform and considers explicit specification of energy-related properties at design time. Unlike EEL, eGEN supports code generation and specifying energy properties during design time.

MD2 [6] is an approach for cross-platform apps with model-driven development methods following the Model-View-Controller (MVC) pattern for android and iO. ADSML [9] is an adaptive domain-specific modeling language for cross-platform mobile app development. DSL-Comet [17] is the active DSL that assists the user in creating graphical DSLs for smart city applications. RAPPT [1] aids the developers in specifying the characteristics of mobile applications using domain-specific visual languages and graphical notations. MoWeBA Mobile [13] is a model-driven approach that covers the data layer of mobile apps. The existing tools target textual and graphical models to specify mobile apps for cross-platform app development. However, they do not cover modeling the energy-aware self-adaptive behaviors of location-based applications. In contrast, eGEN aims to develop a domain-specific language and code generator for location-based applications.

6 CONCLUSION

This paper presents eGEN tool for modeling energy-aware self-adaptive behaviors of location-based mobile applications. The domain analyst may use the textual DSML to specify the energy-saving adaptation policies. The developer may use the generated battery-aware code in the existing repositories. The preliminary evaluation presented in this paper demonstrates that the instrumented code shows a considerable reduction in battery consumption for the trials that last for 12-14 minutes. Hence, we believe that the eGEN generated code might lead to significant battery savings when the apps are used for a longer time in real-world scenarios. Therefore, writing battery-aware code might help developers to reduce unwanted battery consumption in location-based android applications.

REFERENCES

[1] Scott Barnett, Iman Avazpour, Rajesh Vasa, and John Grundy. 2019. Supporting multi-view development for mobile applications. Journal of Computer Languages 51 (2019), 88–96.
[2] Thibault Béziars, Issofe, Massimo Tisi, Jean-Marie Mottu, and Gerson Sunye. 2020. Annotating executable DSLs with energy estimation formulas. In Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering. 22–38.
[3] Nicholas Capurso, Tianny Song, Wei Cheng, Jiguo Yu, and Xuuben Cheng. 2017. An Android-Based Mechanism for Energy Efficient Localization Depending on Indoor/Outdoor Context. IEEE Internet of Things Journal 4, 2 (2017), 299–307.
[4] Taehwa Choi, Yohan Chon, and Hyoung Cha. 2017. Energy-efficient WiFi scanning for localization. Pervasive and Mobile Computing 37 (2017), 124–138.
[5] Xiaodong Gong, Jingbin Liu, Sheng Yang, Gege Huang, and Yu Bai. 2021. A Usability-Enhanced Smartphone Indoor Positioning Solution Using Compressive Sensing. IEEE Sensors Journal 22, 3 (2021), 2823–2834.
[6] Henning Flekötter, Tim A Majchrzak, and Herbert Kuchen. 2013. Cross-platform model-driven development of mobile applications with md2. In Proceedings of the 28th Annual ACM Symposium on Applied Computing. 526–533.
[7] Haozheng Huang, Georg Gartner, Jukka M Krips, Martin Raubal, and Nico Van de Wege. 2018. Location based services: ongoing evolution and research agenda. Journal of Location Based Services 12, 2 (2018), 63–93.
[8] Mohamed Ibrahim and Mousta faYoussel. 2012. CellSense: An accurate energy-efficient GSM positioning system. IEEE Transactions on Vehicular Technology 61, 1 (2012), 286–296.
[9] Xiapeng Jia and Christopher Jones. 2015. An approach for the automatic adaptation of domain-specific modeling languages for model-driven mobile application development. In ICSOFT. Springer, 365–379.
[10] Dohee Kim, Sooyoon Lee, and Hyokyung Bahn. 2016. An Adaptive Location Detection scheme for energy-efficiency of smartphones. Pervasive and Mobile Computing 31 (2016), 67–78.
[11] Mohamed Lachgar and Abdelmoumouin Abdali. 2014. Generating Android graphical user interfaces using an MDA approach. In 2014 Third IEEE International Colloquium in Information Science and Technology (CIST). IEEE, 80–85.
[12] Carlos Alberto Medeiros, Alan Bandeira, Paulo Henrique M Mta, and Mathus Paixao. 2020. MDE in the Wild: An Exploratory Analysis on What Developers are Discussing from Q&A Platforms. In Proceedings of the 34th Brazilian Symposium on Software Engineering. 157–166.
[13] Manuel Nunez, Daniel Bonhaeure, Magali Gonzalez, and Luca Cernuzzi. 2020. A model-driven approach for the development of native applications focusing on the data layer. Journal of Systems and Software 161 (2020), 110489.
[14] Thomas Olutoyin Oshin, Stefan Poslad, and Athen Psaila. 2015. Improving the energy-efficiency of GSM based location sensing smartphone applications. In 2015 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, 1698–1705.
[15] Jeongyup Park, Jongheeon Kim, and Ramesh Govindan. 2010. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In Proceedings of the 8th international conference on Mobile systems, applications, and services. ACM, 299–312.
[16] Rui Pereira, Tiago Carção, Marco Couto, Jácome Cunha, João Paulo Fernandes, and João Saravia. 2020. SPEl1ing out energy leaks: Aiding developers locate energy inefficient code. Journal of Systems and Software 161 (2020), 110463.
[17] Diego Vaquero-Melchor, Javier Palomares, Esther guerra, and Juan de Lara. 2017. Active domain-specific languages: Making every mobile user a modeler. In 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, 75–82.