A Generative Approach for Android Sensor-based Applications

Davi Tabosa[1], Paulo Duarte[1,2], Rafael Carmo[2], and Windson Viana[1,2]

[1] Group of Computer Networks, Software Engineering and Systems (GREat)
[2] Federal University of Ceara (UFC), Fortaleza – CE – Brazil
davitabosa12@alu.ufc.br, pauloduarte@great.ufc.br,
carmorafael@virtual.ufc.br, windson@great.ufc.br

Abstract With the popularisation of smartphones, mobile devices became a crucial technological element in Information Systems. Beyond be the end-user platform, smartphones have several sensors to capture and characterise the user's context. Current Context-Aware Information Systems (CAIS) use this data to improve user experience by filtering data, services, and, even, adapting the system behaviour. The design and development of the capture, inference, and action parts of CAIS can be challenging for mobile developers, due to the diversity of mobile hardware ecosystem and sensor APIs' complexity. This paper presents EasyContext, a visual tool for designing and generating contextual rules for mobile devices, using the Google Awareness API as its contextual information provider. Mobile developers specify what to capture and when to react to context-changes in the visual tool and these contextual rules are exported to Android projects. We evaluated our approach with eight developers, which were selected to participate in a quasi-experiment. The experiment involves developing two mobile applications using both EasyContext and Google Awareness API. Preliminary results show that it is possible to build applications with fewer lines of code compared to the Google Awareness API. According to the developers, the proposed approach has better readability and makes it easier to design contextual rules.

Keywords: Context-Aware Information Systems · Android · Generative Programming · Sensor-based.

1 Introduction

Current smartphones have several built-in sensors (e.g. gyroscopes, accelerometers, GPS) at affordable prices. Also, their increase in memory, processing, and communication capabilities set the basis to the rise of context-aware systems. Due to smartphones popularisation, many software developers have started to use information obtained from sensors to improve user's experience[11]. The context in this domain can be defined as *"any information that can be used to characterise the situation of an entity. An entity is a person, place, or object*

that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [8] [5].

Context-Aware Information Systems (CAIS) use context information to improve user experience by filtering data, services, or adapting the overall system behaviour [17]. Smartphones are now the most important end-user platform for CAIS. They represent a hub of sensor data for these systems. CAIS encompasses the gathering and use of contextual information for various purposes, from general utility applications like map-based navigation systems, smart home assistants to public service applications [14][10][11]. Many challenges exists in the design and development of CAIS challenges [1], such as: sensor data accuracy and availability [15][9][12], specification of the context dimensions [7], the complexity of the sensor access code [13][15][9][16][6], and the modelling and execution support of contextual rules [10][9][16].

In the Android ecosystem, one of the most widely used solution to reduce sensor access complexity is the Google Awareness API[3]. This API provides an interface to access a set of contextual information common to many Android smartphones. For example, Air4People[14] is an application that uses Awareness API to get user location information and current weather. With that data, this information system sends notifications about the air quality to the user.

As with other context-aware platforms, creating contextual rules using Awareness API remains a complex and verbose process. The developer must have advanced knowledge of the Android internal communication protocols, in addition to specific features of each version update of the platform. For example, creating a contextual rule on Android 7.0 is different from creating one on Android 8.0. This heterogeneity can make it challenging to implement and maintain systems that use the Awareness API as a data source.

Based on these challenges, we investigated the following research question: "How to assist the modelling and implementation of contextual rules for sensor-based systems on the Android platform?"

Given this scenario, we propose EasyContext, a tool for modelling contextual rules and generating its behaviour with the Awareness API as a context provider. EasyContext aims to facilitate the modelling and use of contextual rules, especially for novice developers, through a visual interface. In this paper, we also present an evaluation with eight mobile app developers who used the tool. After analysing the codes generated and the results of a qualitative survey, we found that EasyContext is easier to use when compared to Awareness API. Also, it helps to reduce the number of lines written by the mobile developer and improves the readability of the code.

The remainder of the paper is organised as follows: Section 2 presents Awareness API. Section 4 introduces EasyContext, while Section 3 describes the methodology used in its development. Our evaluation approach is available in Section 5, with the results shown in Section 5.2 and discussed in Section 5.3. Related Work is presented in Section 6. Finally, Section 7 presents the final considerations.

[3] https://developers.google.com/awareness

2 Google Awareness API

Google released Awareness API in 2016 as part of Google Play Services suite[4] trying to reduce the Sensor API complexity and aiming to provide high-level user's context information. It contains two APIs, named Snapshot and Fence. The Snapshot API obtains the current information about the user's context from the smartphone's sensors (e.g., user's activity, location, headphone state). The Fence API represents a contextual rule[18], making the Android device execute a code if a previously established condition becomes true or false (e.g., if the user's activity is "walking" then Android executes the X method). For this action code to run, developers must configure and register it using the Broadcast component [5] of the Android platform. Awareness API offers seven categories of contextual information: Time (local time of day), Location (geographic coordinates), Place (geocoded information), Activity (detected user activity), Beacons (nearby Bluetooth beacons), Headphones (headphone state) and Weather (climatic conditions). Some categories that are present in the Snapshot API, such as Weather and Place, are not present in the Fence API, making it impossible to create contextual rules for these categories with the API. The Google Awareness API is an initiative to mitigate CAIS code complexity. However, this approach is still somehow counter-intuitive as it requires the developer to perform a series of steps that increase the complexity of coding[10].

3 Research Methodology

Figure 1 shows the sequence of steps for we adopted in the design and development of EasyContext. At first, we performed a requirements elicitation with emphasis on the research question. Then, we studied the Awareness API; its strengths and weaknesses were highlighted. After that step, we developed Version 1 of EasyContext, which was then used to build two context-aware applications. They illustrated the use of the approach in a real scenario. We were able to find the existing limitations of Version 1. We then refactored and improved the approach, generating Version 2 (described in the last section). Finally, we evaluated it with developers, which we explain in details in Section 5

3.1 Application for redesign

We made two applications to test and refactor the EasyContext Version 1. The first one is a music player, where the user creates playlists with songs of their choice. The user can also connect a playlist to one or more contextual rules of activation. (e.g., if it's raining, play *Umbrella*). When launched, the application detects the user's current context and suggests the playlists created by the user, ordered by the similarity between the current context and the context recorded

[4] https://developer.android.com/distribute/play-services/
[5] https://developer.android.com/guide/components/broadcasts

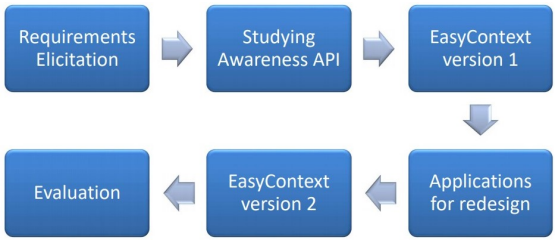

Figure 1. Flowchart of EasyContext's development methodology

in the playlists. This mobile app has a unique contextual rule and a comparison algorithm between contexts. At the time when the user puts the headphone, the application sends a notification asking if he or she wants to listen to music at that moment. If the user clicks on the notification, the app opens and recommends playlists according to the user's current context. To sort the playlists according to the user's current context, we have developed an algorithm that calculates the similarity between two contexts. The music player always recommends the playlist with the most detailed context (i.e., if it matches the current user's situation with more context elements).

The second one is a reminder application based on contextual change. The user registers a message that they want to remember and incorporates a contextual rule. When the user's current context matches the recorded context, the application displays a notification with the message registered previously. This application has no fixed contextual rules. When the user composes a reminder, they must link it to a context rule. These rules cannot be made in the Web Tool, because the user has full control over when the reminder should be triggered.

3.2 Refactoring

During the development of these two applications, several difficulties happened regarding what the tool offered that time. EasyContext version 1 was more based on the Fence concept and gave limited functionality for Snapshots. Also, the programmatic creation of Fences was still quite verbose with EasyContext, with its creation methods receiving many sophisticated attributes. Actions were very similar to Broadcast Receivers and needed to be instantiated by the programmer in the code. This fact affected the rule life cycle that could be destroyed by the Java virtual machine's garbage collector at any time, making it unstable.

After that, we have created a class that specialises in delivering Snapshot information. Since the Snapshot class delivers contextual data asynchronously, the developer must create a listener to retrieve the information. To get the updated context data, the developer must invoke the *updateContext* function, which retrieves the context data by packaging it into a *CurrentContext* object. The developer can decide which context providers should be updated. The *CurrentContext* class is the default way to request current context information. It

contains information that would be in the Awareness' Snapshot API but as a single object. The two applications we developed use customised contextual rules for the end-user, which requires the persistence of these rules. To get around this problem, now, all the EasyContext rules can be serialised and deserialised in JSON. With that, we facilitate the storage of contextual rules on data persistence systems such as *SQLite* and *Shared Preferences*. The *GeneralReceiver* (Figure 5) is a BroadcastReceiver that resides in an application with EasyContext. It is the intermediary between the *Intents* sent by the Awareness API and the Action written by the developer. When *GeneralReceiver* receives a message from the Awareness API indicating that this condition has become true, the *GeneralReceiver* creates an instance of the Action that has been registered with the rule (on the Web tool) and executes the *doOperation* method. This enables the application to go to standby and maintain functionality in the background, even with the thread of the main application is not executing.

The *FenceManager* is a mediator between the *fences* registered in the Awareness API and EasyContext. The developer can create new context rules at runtime and register them using this class. This is an alternative to Web Tool if the developer needs to register a rule defined by the end user. It is also responsible for linking a rule to an Action. When the *GeneralReceiver* is activated, with the name of the *fence* in hand, it queries FenceManager to know which Action should execute at the moment.

4 EasyContext

EasyContext is a tool to help developers insert context-aware behaviour into Android applications in a less verbose and more straightforward way. We divided the approach into two parts: a Web Tool for modelling contextual rules and an Android library for interpretation and management of these rules. This library acquires its contextual information by using Google Awareness API, providing abstractions for the invocation of Awareness API methods.

Figure 2 presents an overview of the process of developing a context-aware application using EasyContext. The developer models the contextual rules using EasyContext Web, which generates a JSON document that represents the modelling made and the context providers that should be activated (e.g., headphone state monitoring). After that, developers must include the JSON document into their Android projects. After embedding it, the development process follows a standard Android project. In EasyContext, a contextual rule has two parts: Condition and Action. A Condition is a clause or set of clauses that define a context situation. Action is a set of instructions that Android will execute if a contextual condition is satisfied. Expression 1 describes a contextual rule of our approach, using the Backus-Naur form [2]. An example of a rule would be: "If the user enters a hospital, set the mobile phone to silent mode". In this case, the contextual condition would be "entering a hospital" and the resulting phenomenon, if this condition is true, would be "configuring the mobile phone in silent mode".

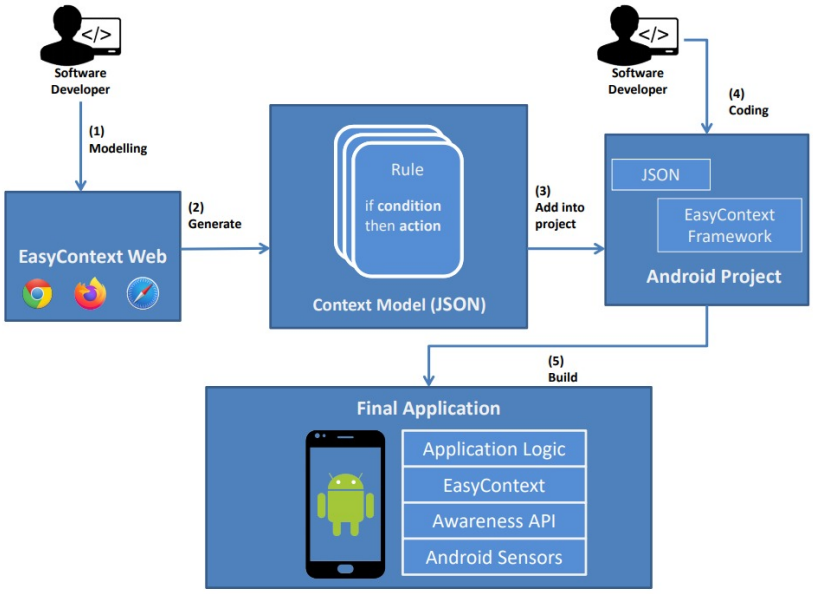

Figure 2. Application development flow using EasyContext.

$$
\begin{aligned}
\langle \text{contextual rule} \rangle &\models \langle \text{name} \rangle \ \langle \text{clause} \rangle \ \langle \text{action} \rangle \\
\langle \text{clause} \rangle &\models \langle \text{situation} \rangle \ \langle \text{situation} \rangle \ \mid \ \langle \text{situation} \rangle \\
\langle \text{situation} \rangle &\models \textit{a contextual situation e.g. standing still} \\
\langle \text{name} \rangle &\models \text{A} \ldots \text{Z} \ \mid \ \text{a} \ldots \text{z} \ \mid \ 0 \ldots 9 \\
\langle \text{action} \rangle &\models \textit{method to be called}
\end{aligned}
$$

Expression 1. Grammar of a contextual rule from EasyContext

4.1 EasyContext Web Tool

Many approaches proposed tools for configuring contextual rules as plugins for the Integrated Development Environment (IDE) [9]. Attaching a modelling tool to one third-party IDE results in risks such as the cease of IDE development or the limitation of the target platform. To avoid this, we constructed the EasyContext configurator as a web app independent of third-party IDEs.

Using a Web tool to customise or improve an Android project is not an uncommon practice among software developers. For instance, the Google Map's *Styling Wizard*[6] allows the configuration of visual styles of maps in a web editor. All the modifications made by the developers are updated in real-time in the online map. This tool generates a JSON document. Developers must incorporate

[6] https://mapstyle.withgoogle.com/

it into the application project that makes use of Google Maps. This flow is similar to those of other tools supporting mobile app development, such as Firebase[7].

Figure 3 presents the main screen of EasyContext Web tool. By accessing the page, the developer visualises a screen for contextual rule management. He can create, edit, and delete several rules in a single document. All rules must be correctly named to be used in the Android project posteriorly.

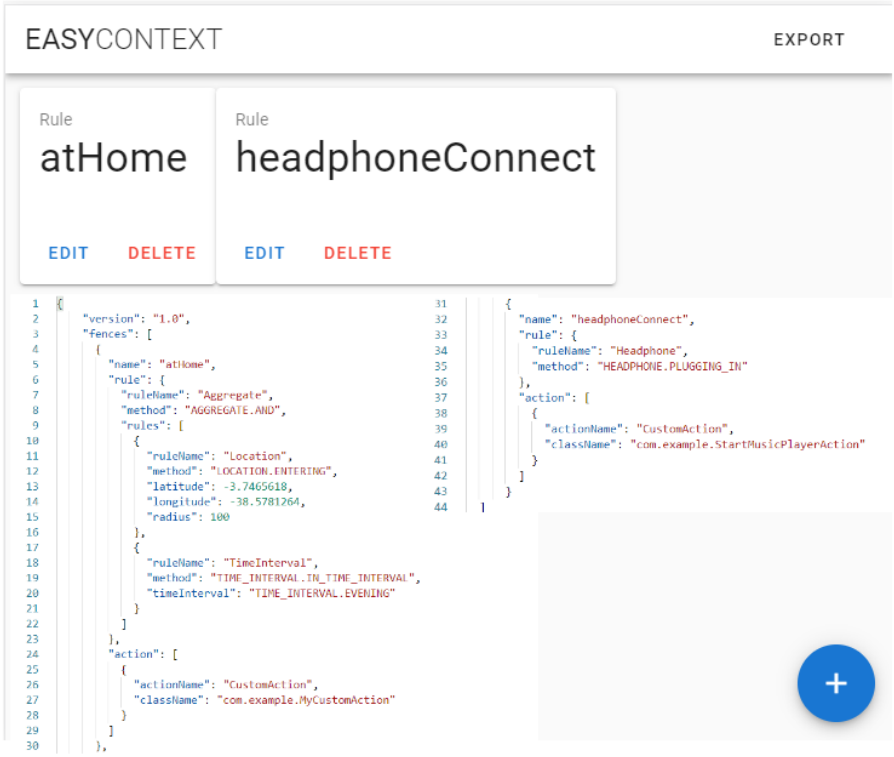

Figure 3. Main screen of EasyContext containing two rules. At the bottom, one image showing the JSON code related to those rules.

EasyContext Web tool uses a card-based metaphor for representing both action and contextual clauses. The cards are interactive, and their values are modified dynamically according to the user input. In the left menu, there are the cards categories that users can select, divided into *Conditions* and *Actions*. *Conditions* represent the EasyContext's contextual clauses, which are four in total: Headphone, Activity, Time, and Location. Their functionalities are identical to

[7] https://firebase.google.com/

those of the Awareness API. Figure 4 shows the main screen of EasyContext web with two modelled rules. The JSON code bellow represents those rules.

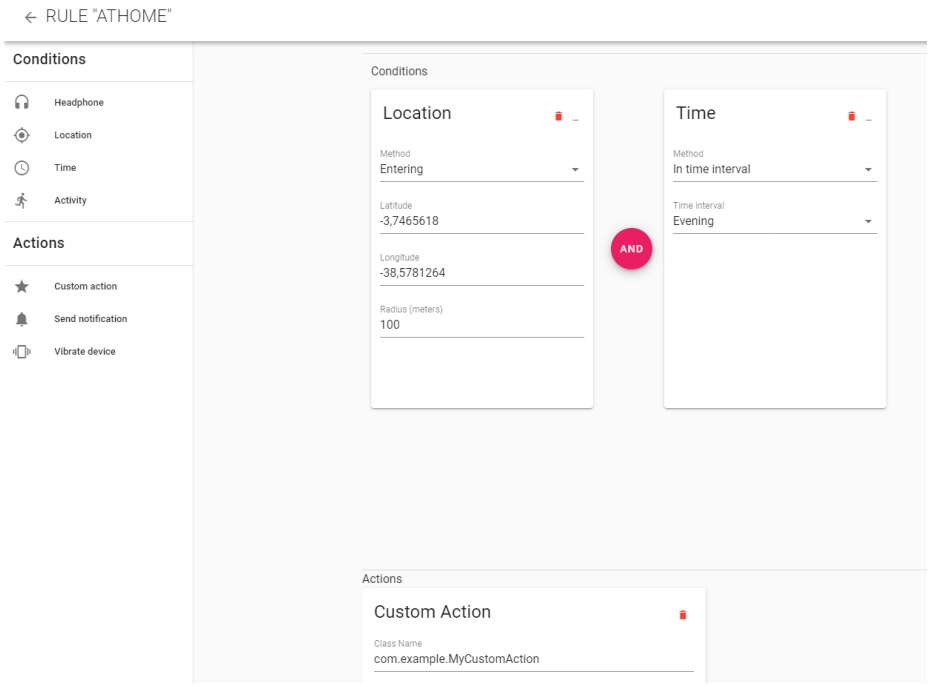

Figure 4. Screen of Contextual Rule modelling

The final generated artefact is a JSON document to be incorporated in the developer's Android project. Figure 3 presents an example of a generated document with two configured rules.

4.2 EasyContext Android Library

The EasyContext Android library is the approach core, acting as an interpreter of the high-level commands specified on the Web Tool and serving as an abstraction layer of the Awareness API features. Similarly to the Awareness API, the library contains two parts: Snapshot and Rule. The Snapshot class has static methods to perform context acquisition. It incorporates some methods from the Awareness API. In terms of design patterns, it works as a *Facade*, encompassing all the methods from the Snapshot API. The *Rule* part of EasyContext encompass other support classes, as shown in Figure 5. The interface FenceAction is a facilitator to the process of writing contextual rule actions, i.e., the app behaviour in response to the user's context changes. The developer can implement this class

and override the method *doOperation* to specify this the action, to favour the actions reuse.

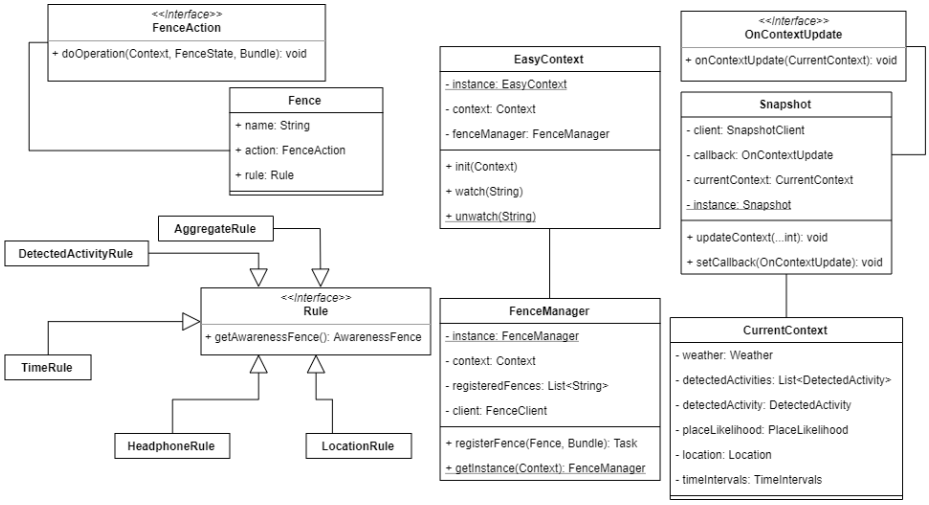

Figure 5. Class diagram of the EasyContext Android Library

Rule classes help to specify the contextual rules in a friendlier way. Also, they allow categorising and encapsulating context acquisition methods. For instance, the FenceManager is a *Singleton* object that manages the contextual rules. This class has the objective of registering and removing contextual rules in the Awareness API stack. It performs the registration of *Broadcasts* in parallel, without the need of the programmer intervention. Due to this control, we reduced the code necessary to be written by the developer for implementing a context-aware behaviour with Awareness API. The coding effort is reduced since there is no need to rewrite several lines of code that deal with the management of these *Broadcasts*. The *EasyContext* class provides control to the developer in the task of observing contextual rules. Making use of the names of the rules modelled in the Web Tool, the programmer can activate and deactivate them.

5 Evaluation

We evaluated our research aiming to gather the acceptance of EasyContext and to compare its use with the direct coding with Awareness API. For that, we used code complexity metrics analysis in both approaches during an experiment with mobile developers. In two rounds of experiments, we evaluated EasyContext with a group of 08 participants in the area of Computer Science, in which 04 are undergraduate students, and 04 are graduate students. All of them had previous experience in Android programming using the Java language in the Android

Studio environment. However, none of them had experience in the creation of context-aware information systems or had in-depth knowledge of the Awareness API features. However, all of them have followed Awareness API theoretical and practical classes.

5.1 Procedure

The evaluation consisted of developing two toy applications, making use of both the Awareness API and EasyContext. Pre-made projects were given to all participants. They had to code the context-aware behaviour of those apps. The first application is a fitness tracker. Once activated, the application counts how long the user has been sitting or moving. The second application is a reduced version of the reminder app, which send messages to the users whenever the users enter in certain places at a given time of day.

First, we made an initial levelling to make it possible to measure the direct impact of the usage of the Awareness API and EasyContext. After that, we randomly divide the students into two groups, called Group A and Group B. The experiments were performed in a controlled environment, and the participants developed the applications using the IDE Android Studio IDE version 3.2 in iMac computers. Group A developed the fitness tracker making use of EasyContext, and the reminders app making use of the Awareness API. Group B developed the tracker using the Awareness API and the reminders application using EasyContext. Before the beginning of the evaluation, participants answered a questionnaire regarding their abilities in Android programming and Awareness API. We timed the development time every student took to develop each application. At the end of the activity, the students answered two other questionnaires [3] regarding that experience in developing applications using both tools.

5.2 Results

Table 1 presents the average results of the static code analysis for the eight subjects. The metrics collected were: average operation complexity (OCavg, lower is better), weighted method per class (WMC, lower is better) and lines of code (LOC, lower is better). This collection was performed through a *plugin* available for the Android Studio and IntelliJ IDEA IDEs called MetricsReloaded[8]. The static code analysis relative to the fitness tracker application reached 13 in WMC and 1.62 in average operation complexity when EasyContext was used; 14 in WMC and 1.75 in average operation complexity when Awareness API was used. The analysis of the codes related to the reminders application reached 21.75 in WMC and 2.08 in average operation complexity when EasyContext was used; 21 in WMC and 2.1 in average operation complexity when Awareness API was used.

Analysing the amount of lines of code written by the developer, EasyContext achieved an average of 97.75 and 131.5 lines written in the fitness tracker and

[8] https://plugins.jetbrains.com/plugin/93-metricsreloaded

Activity	Tool	OCavg	WMC	LOC
Fitness Tracker	EasyContext	1,62	13	97,75
	Awareness API	1,75	14	111
Reminders	EasyContext	2,1	21	131,5
	Awareness API	2,08	21,75	153,5

Table 1. Static code analysis of the codes written by the participants

reminders applications respectively. The Awareness API achieved an average of 111 and 153.5 lines written in the fitness tracker and reminders applications respectively.

Developers also answered nine questions for each approach. All items were constructed using the 5 point Likert scale (Strongly Disagree = 1, Strongly Agree = 5). We have elaborated these nine questions based on Cognitive Dimensions of Notations[3], by Blackwell et al. Figure 6 shows the analysis comparing the questionnaire results. When asked if the "tool fulfills best the requested specifications", the results were mostly neutral both to EasyContext and Awareness API, averaging 3.5 and 3.12 respectively. This neutrality persists when questioned if "less errors occur in the tool", averaging 3.12 for EasyContext and 3.25 for Awareness API. When questioned if "the code is more legible in the tool" the average of answers were 4.25 for EasyContext and 3.12 for Awareness API. In the question regarding if "the tool restricts my freedom as a programmer", the answer were similar, 2.5 for EasyContext and 2.75 for Awareness API. Regarding the theme "the tool doesn't have all features I need", results were also identical, averaging 2.87 for both tools.

5.3 Discussion

While observing the questionnaire analysis, EasyContext has behaved satisfactorily, especially in questions involving code readability and ease of design. Also, the developer writes fewer lines of code using EasyContext, which may have contributed to the readability of the code. The number of lines of code increases significantly using the Awareness API. While in the development activity of the Diary had an average of 97.75 lines of code using the model approach, developing with Awareness API increased the number of lines of code to 111 on average. When taken into account the complexity of the code written by the participants, there was no significant difference between the two approaches in both metrics.

Due to the approach used in the EasyContext development cycle, we infer some threats to the validity, both in methodology and evaluation, and seek to mitigate them. The same software engineer developed the two versions of EasyContext and the two application examples. This can result in a bias in the requirements elicitation for version 2 redesign. To mitigate this problem, we decided to evaluate with a broader group of developers, with the flexibility to comment on improvements to the tool. During the execution of the evaluation, we have registered the occurrence of problems in technological terms (e.g., failure to access Web Tool servers on two machines) and human conditions (e.g.,

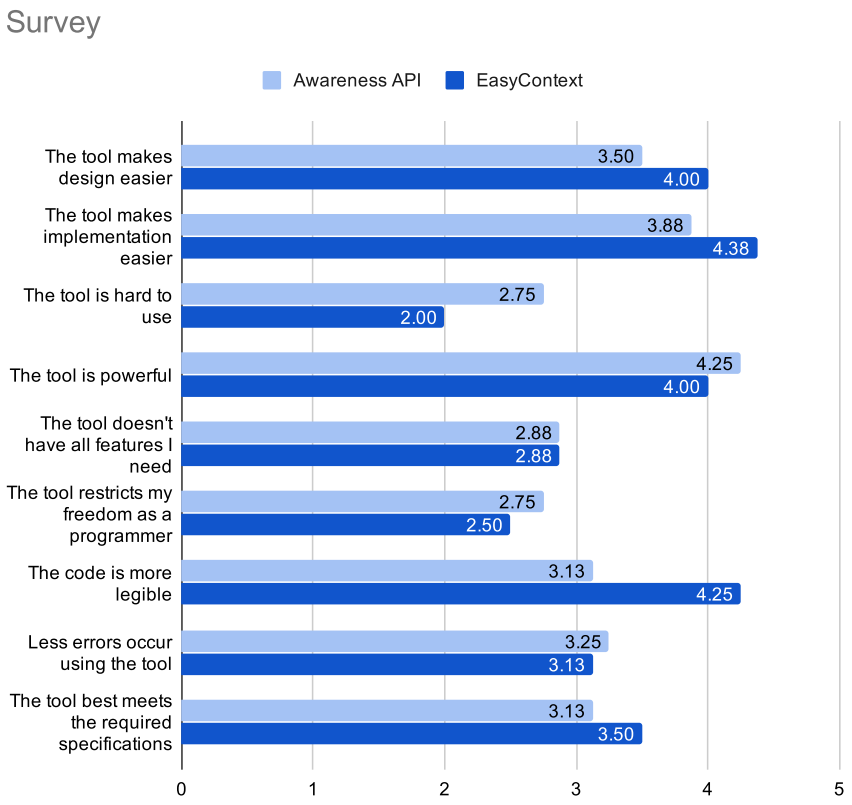

Figure 6. Acceptance Survey

less focused participants). Initially, 14 developers participated in the evaluation, adding up the two rounds. However, 06 participants were unable to complete the activities in the time stablished (2 hours) correctly. While this problem has been most significant in coding applications with the Awareness API, we decided to remove from the analysis of the 06 participants who failed to complete the activities. This decision may have influenced the results. However, for a comparison between the codes to be carried out, we decided to use in the evaluation only those developers who managed to carry out the two activities correctly.

6 Related Work

Several approaches in the literature use generative programming focused on the development of sensor-based mobile applications. In our previous research, we

investigated model-based approaches that integrate DSLs (Domain Specific Languages) and context-aware management middleware platforms [9]. In that research, we also proposed CRITiCAL[9], a visual tool for contextual modelling using ContextRuleML. It had a visual editor integrated to Eclipse for creating contextual rules using UML-based models. From the model, CRITiCAL generated an Android project with the contextual rules already configured, ready to be used in the project. CRITiCAL used the LoCCAM[13] middleware to acquire contextual information and execute the context rules. We observed from that research that integrating the modelling with Eclipse was a mistake since the Android Studio became the standard Android development environment. Also, applications developed with the help of CRITiCAL were limited to the scope of LoCCAM.

More recently, Likudie et al. proposed an approach[12] for generating Android applications without the need for the user to have experience with programming languages. Similar to EasyContext, they use a web app for modelling Android projects. This visual tool generates a JSON document that describes the modelled Android project and the web application servers perform all the binary code generation. However, that modelling approach has limited access to the device sensors and not support context-aware behaviour definition.

CONtroL[16] uses the *models@run.time*[4] approach to perform automatic testing of context-aware applications. The focus of CONtroL is to provide runtime checking of context-aware behaviour. CONtroL has a mobile framework that inspect the application in execution. Similar to EasyContext, CONtroL uses Google Awareness API as Context Provider. However, an application using CONtroL changes its behaviour depending on the models specified by the developer using a Dynamic Software Product Line approach. CONtroL focuses more on verification and also limits the context-aware application architectures.

JUSE4Android [6] focuses on building graphical user interfaces for mobile Business Information System applications. JUSE4Android has two syntaxes: graphical, using UML class diagrams and textual using Java Annotations. Although using a more advanced model system, this approach does not have any context acquisition system. Also, JUSE4Android targets Business Information System developers, while EasyContext targets CAIS developers.

Table 2 shows a comparison of the presented approaches. The analysis of the approaches allowed us to list the desirable characteristics for a contextual rules modelling tool with code generation focused on sensor-based applications in the Android platform.

7 Final Considerations and Future Work

The use of contextual information is a challenge for software developers [10]. So, based on the research question "How to assist the modelling and implementation of contextual rules for sensor-based systems on the Android platform?", we conducted a literature study about generative programming and context modelling approaches. After that, we propose a solution combining modelling and gener-

	Likudie's Approach [12]	**CRITiCAL [9]**	**CONTrol [16]**	**JUSE4 Android [6]**	**EasyContext**
Model Representation	JSON	Context-RuleML	JSON	UML / Java	JSON
IDE Independence	Yes	No	No	Yes	Yes
Target Platform	Android	Android	Android	Android	Android
Visual Language	Yes	Yes	No	Yes	Yes
Contextual Rule Modelling Support	No	Yes	Yes	No	Yes
Context Provider	—	LoCCAM	Awareness API	—	Awareness API

Table 2. Comparison among generative approaches and EasyContext

ative programming, and uses the Awareness API as a contextual information provider. EasyContext encapsulates the complexity of configuring and using the API. After a usability assessment with eight participants, who developed applications with and without EasyContext. We attested a usability gain when comparing the use of EasyContext to direct coding with Awareness API. Besides, the code complexity metrics analysis revealed that the lower number of lines of code in applications with EasyContext did not affect its cyclomatic complexity.

However, the approach can still be improved. There is still no support for creating more-complex context rules (*Rules* compositions) using Web Tool. The current metaphor, using cards to represent contextual rules, has the limitation of showing a few cards on the screen simultaneously. This metaphor makes it hard to create more elaborated contextual rules. As future work, we will add new context providers to EasyContext. Our objective is to turn EasyContext platform-independent since the Web Tool already is. We will also make improvements to the visual representation metaphor based on feedback from evaluation participants.

References

1. Aarab, Z., Saidi, R., Rahmani, M.D.: Towards a framework for context-aware mobile information systems. In: 2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems. pp. 694–701 (Nov 2014). https://doi.org/10.1109/SITIS.2014.89
2. Backus, J.W.: The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. Proceedings of the International Comference on Information Processing, 1959 (1959)
3. Blackwell, A.F., Britton, C., Cox, A., Green, T.R., Gurr, C., Kadoda, G., Kutar, M., Loomes, M., Nehaniv, C.L., Petre, M., et al.: Cognitive dimensions of notations:

Design tools for cognitive technology. In: International Conference on Cognitive Technology. pp. 325–341. Springer (2001)
4. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10), 22–27 (Oct 2009). https://doi.org/10.1109/MC.2009.326
5. de Carvalho, W.V.: Mobility and Context-awareness for Personal Multimedia Management: CoMMediA. Ph.D. thesis, Joseph Fourier University, Grenoble, France (2010), https://tel.archivesouvertes.fr/tel00499550
6. da Silva, L.P., Brito e Abreu, F.: Model-driven gui generation and navigation for android bis apps. In: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). pp. 400–407 (Jan 2014)
7. Daniel, F., Matera, M., Quintarelli, E., Tanca, L., Zaccaria, V.: Context-aware access to heterogeneous resources through on-the-fly mashups. In: International Conference on Advanced Information Systems Engineering. pp. 119–134. Springer (2018)
8. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. **5**(1), 4–7 (Jan 2001)
9. Duarte, P.A.S., Barreto, F.M., Gomes, F.A.A., Viana, W., Trinta, F.A.M.: Critical: A configuration tool for context aware and mobile applications. In: 2015 IEEE 39th Annual Computer Software and Applications Conference. vol. 2, pp. 159–168 (July 2015)
10. Föhr, J.: Context-awareness through Google Awareness API (2019), https://lutpub.lut.fi/handle/10024/159360
11. Gedeon, J., Himmelmann, N., Felka, P., Herrlich, F., Stein, M., Mühlhäuser, M.: vstore: A context-aware framework for mobile micro-storage at the edge. In: International Conference on Mobile Computing, Applications, and Services. pp. 165–182. Springer (2018)
12. Likudie, C.K.: Implementation of a web-based code generator for the android Mobile platform. Ph.D. thesis, Ashesi University (2018)
13. Maia, M.E.F., Fonteles, A., Neto, B., Gadelha, R., Viana, W., Andrade, R.M.C.: Loccam - loosely coupled context acquisition middleware. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. pp. 534–541. SAC '13, ACM, New York, NY, USA (2013)
14. Ortiz, G., Boubeta-Puig, J., Corral-Plaza, D.: Air4people: a smart air quality monitoring and context-aware notification system. Journal of Universal Computer Science **24**(7), 846–863 (2018)
15. Raento, M., Oulasvirta, A., Petit, R., Toivonen, H.: Contextphone: A prototyping platform for context-aware mobile applications. IEEE pervasive computing **4**(2), 51–59 (2005)
16. Santos, I.D.S., Santos, E., Andrade, R., Neto, P.: Control: Context-based reconfiguration testing tool. In: IX Tools Session of CBSoft 2018 (09 2018)
17. Wieland, M., Kopp, O., Nicklas, D., Leymann, F.: Towards context-aware workflows. In: CAiSE07 proc. of the workshops and doctoral consortium. vol. 2, p. 15 (2007)
18. Yurur, O., Liu, C.H., Sheng, Z., Leung, V.C.M., Moreno, W., Leung, K.K.: Context-awareness for mobile sensing: A survey and future directions. IEEE Communications Surveys Tutorials **18**(1), 68–93 (Firstquarter 2016). https://doi.org/10.1109/COMST.2014.2381246

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

T117g Tabosa, Davi Batista.
A Generative Approach for Android Sensor-based Applications / Davi Batista Tabosa. – 2019.
15 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2019.
Orientação: Prof. Dr. Windson Viana de Carvalho.
Coorientação: Prof. Me. Paulo Duarte.

1. Context-Aware Information Systems. 2. Android. 3. Generative Programming. 4. Sensor-based. I. Título.

CDD 302.23
