EasyContext: Facilitando o desenvolvimento de aplicações com a Awareness API

Davi Batista Tabosa
Universidade Federal do Ceará
Av. Mister Hull, s/n
Fortaleza, Ceará 60451-970
davitabosa12@gmail.com

Windson Viana
Universidade Federal do Ceará
Av. Mister Hull, s/n
Fortaleza, Ceará 60451-970
windson@virtual.ufc.br

ABSTRACT

Due to the popularity of smartphones and their built-in sensors, Context-Aware Mobile (CAM) applications are on the rise. Some libraries and APIs help developers to create CAM applications, such as the Google Awareness API. However, the initial configuration, the write of context-aware rules, and code readability persist being complex tasks, specially for beginner developers. This paper proposes an approach to ease the development of CAM applications using Awareness API. Our proposal, called EasyContext, includes a Web Configurator and an Android Framework, which hides code complexity and the Awareness API configuration. A PoC using the EasyContext is also presented.

KEYWORDS

Context-Awareness, Google Awareness API, Ubiquitous Computing

1 INTRODUÇÃO

Aplicações móveis sensíveis ao contexto (e.g., WAZE) e jogos móveis baseados em informações contextuais dos jogadores (e.g., Pokemon Go) tem tido ascensão notória nos últimos anos devido à popularização dos dispositivos móveis. Embutido nestes aparelhos, há vários sensores como giroscópios, acelerômetros, bússolas e GPS[10]. Este hardware nativo combinado com algoritmos de monitoramento e reconhecimento de situações permitem que aplicações e jogos móveis se adaptem conforme à situação do ambiente ou do usuário que as acessa. Essas aplicações, ditas CAM (*Context-Aware and Mobile* apps, ou Móveis e Sensíveis ao Contexto) executam a maior parte de sua interação com o usuário em um dispositivo móvel e adaptam o seu comportamento (e.g., conteúdo, interface do usuário) para o contexto atual do usuário e do ambiente [1][10]. O desenvolvimento de aplicações CAM tem sua gama de desafios, dentre os quais, podem ser citados: desafios inerentes à heterogeneidade dos dispositivos e plataformas, à complexidade do código voltado ao acesso de sensores e à dificuldade da definição de regras contextuais (e.g., comportamento que o sistema deve ter quando o usuário entrar em uma determinada região) [9][7][3][4].

Durante o desenvolvimento de um software é comum o programador encontrar estas situações problemáticas ao estruturar o projeto, sendo muitas delas reincidentes. Muitas pesquisas e ferramentas comerciais surgiram, então, com o objetivo de dirimir esses desafios, como a utilização de *frameworks* e/ou plataformas de *middleware* para uniformizar a sintaxe da aquisição e gerenciamento de informações contextuais[6][5][8]. Seguindo essa vertente, a Google desenvolveu uma API focada na aquisição de contexto na plataforma Android, a Google Awareness API[1]. Essa API permite ao desenvolvedor a aquisição de sete grupos de informações contextuais e a criação de regras contextuais que as combinem por meio do Google Play Services. Assim, para dispositivos da plataformas Android, a Awareness API oferece uma solução que não precisa da prévia instalação de outro software (como um *framework* ou *middleware*). No entanto, algumas funcionalidades do Google Awareness API, como a criação de regras contextuais, necessitam ainda de um processo complexo e verboso para serem implementadas. Dentro deste contexto, este trabalho propõe uma ferramenta de configuração da biblioteca do Google Awareness aliado com o uso de *framework* de desenvolvimento criado com práticas de padrões de projetos. A abordagem, chamada de EasyContext, visa facilitar o desenvolvimento de aplicações sensíveis ao contexto, em especial, para programadores iniciantes neste tipo de domínio. O trabalho mais relacionado a nossa abordagem é a Ferramenta CRITiCAL [3], que dispõe de um editor visual para a criação de regras contextuais. A partir do modelo, é gerado um projeto Android com as regras contextuais já configuradas, prontas para serem utilizadas no projeto. Contudo a CRITiCAL utiliza o LoCCAM [7] para a aquisição de informações contextuais que possui poucas informações contextuais implementadas. A seguir, detalha-se a fundamentação teórica deste trabalho e sua proposta principal.

2 SENSIBILIDADE AO CONTEXTO

Segundo Dey [2], "Contexto é qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade". Esta entidade pode assumir várias formas, como, "uma pessoa, lugar, ou objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o usuário e a aplicação em si". Uma aplicação sensível ao contexto é definida como "aquela que utiliza contexto para prover informações relevantes e/ou serviços para o usuário, onde a relevância depende da tarefa do usuário ". Uma das vantagens de utilizar sensibilidade ao contexto em aplicações móveis é a adaptação de seu comportamento dependendo do contexto onde o usuário está inserido (e.g., localização, temperatura), promovendo uma interação mais natural[3].

Um dos desafios pertinentes no desenvolvimento de aplicativos sensíveis ao contexto é a heterogeneidade de plataformas (e.g., sensores, dispositivos de acesso) [10]. Por exemplo, no mesmo ecossistema Android, existem vários tipos de smartphones, cada um com

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

In: XV Workshop de Trabalhos de Iniciação Científica (WTIC 2018), Salvador, Brasil. Anais do XXIV Simpósio Brasileiro de Sistemas Multimídia e Web: Workshops e Pôsteres. Porto Alegre: Sociedade Brasileira de Computação, 2018.

ISBN 978-85-7669-435-9.

hardware distinto. Certos modelos podem possuir ou não sensores que podem interferir na precisão da inferência de contexto. Além da própria complexidade inerente da programação deste tipo de aplicação. Uma estratégia para dirimir estes problemas é a criação de APIs e plataformas de middleware, como a Google Awareness API, que introduz uma camada de acesso ao hardware de forma uniforme, tornando transparente a aquisição de contexto além de realizar inferência a partir de dados brutos.

A Google Awareness API foi lançada em 2016, exclusivamente para a plataforma Android como parte da suite de serviços do Google Play Services[2]. Ela é dividida em duas APIs, chamadas Snapshot e Fence. A API Snapshot obtém os dados de sensores e realiza a inferência no momento em que seu método é executado.Já a API Fence representa uma regra de contexto[10], em que um trecho de código é executado caso esta regra torne-se verdadeira ou falsa. Para o trecho de código ser executado, ele deve estar configurado e registrado utilizando o componente Broadcasts[3] da plataforma Android. Esse tipo de abordagem é contra intuitivo, pois requer que desenvolvedor realize uma série de etapas que aumentam a complexidade de codificação.

A Awareness API oferece sete categorias de informações contextuais: *time* (hora local), *location* (coordenadas geográficas), *place* (lugar), *activity* (detecção de atividades do usuário), *beacons* (detecção de beacons próximos), *headphones* (detecção da presença de fones de ouvido) e *weather* (condições do clima). Algumas categorias que estão presentes na API Snapshot, como *weather* e *place*, não estão presentes na API Fence, o que impossibilita a criação de regras contextuais para essas categorias.

3 O *FRAMEWORK* EASYCONTEXT

Este trabalho propõe a criação do EasyContext, um *framework* para auxiliar desenvolvedores a inserir sensibilidade ao contexto em suas aplicações Android de forma menos verbosa e mais simples. Ele funciona como uma extensão do Google Awareness API. A abordagem é dividida em duas partes: um configurador web para a escrita de regras contextuais e uma biblioteca Java para a interpretação destas regras, além da gerência de registro de Fences e das abstrações para a invocação de métodos da Awareness API.

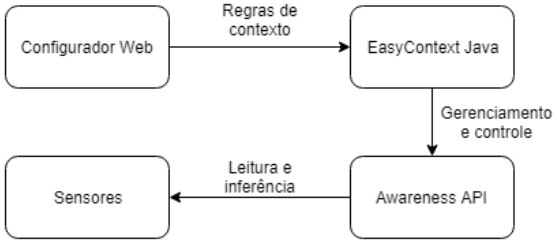

Figura 1: Arquitetura simplificada do EasyContext.

No EasyContext, uma regra contextual possui duas partes: Condição contextual e Ação. Uma condição contextual é uma cláusula ou conjunto de cláusulas que definem um contexto. Ação é um conjunto de instruções que serão executadas caso uma condição contextual ocorra. Um exemplo de regra conceitual seria: "Caso o usuário entre em um hospital, configure o celular em modo silencioso". Neste caso, a condição contextual seria "entrar em um hospital" e o fenômeno resultante caso esta condição seja verdadeira seria "configurar o celular em modo silencioso".

3.1 Configurador Web

Algumas ferramentas de configuração de regras de contexto são feitas como plugins de IDEs. Isso pode ser um problema caso a IDE em questão entre em desuso. Por exemplo, a Ferramenta CRITiCAL[3] foi desenvolvida como plugin para a IDE Eclipse. Porém, atualmente a IDE padrão para desenvolvimento Android deixou de ser o Eclipse e passou a ser o Android Studio, o que deixou a CRITiCAL obsoleta sem um grande retrabalho. Para evitar esta situação, o configurador do EasyContext foi feito como um aplicativo Web, deixando de ser dependente de IDEs. O artefato final gerado é um documento no formato JSON para ser incorporado no projeto Android do desenvolvedor. Na Figura 2, é exposto um exemplo de documento gerado com duas *Activities* configuradas para receberem comportamento sensível ao contexto.

```
{
    "activities":[
        {
            "name": "package.name.awarenesslib.MainActivity",
            "fences" : [
                {
                    "fenceName": "Headphone",
                    "fenceAction": "action1",
                    "fenceType":  "Headphone",
                    "fenceMethod": "Headphone.DURING",
                    "params": {
                        "headphoneState": 1
                    }
                }
            ]

        },
        {
            "name": "package.name.awarenesslib.ReconActivity",
            "fences" : [
                {
                    "fenceName": "Nome Qualquer",
                    "fenceAction": "complex_action",
                    "fenceType":  "DetectedActivity",
                    "fenceMethod": "DetectedActivity.STARTING",
                    "params": {
                        "activityTypes" : [0,1,2,7,8]
                    }
                }
            ]

        }
    ]
}
```

Figura 2: Exemplo de documento gerado pelo configurador web do EasyContext

O campo *name* é o nome da *activity* juntamente com o pacote do projeto. O campo *fences* é um arranjo de uma ou mais regras de contexto que essa *activity* possa ter. Em cada regra de contexto é necessário informar um nome único (*fenceName*), uma ação para ser executada (*fenceAction*) e a condição para ser observada, representada pelos campos *fenceType*, *fenceMethod* e *params*. Os campos *fenceType* e *fenceMethod* definem uma condição contextual

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

análoga à encontrada na Google Awareness API, enquanto *params* são parâmetros que configuram essas funções.

3.2 Biblioteca Java

Ao observar a Figura 1, nota-se que a biblioteca Java é a parte central do *framework*, agindo como interpretador de comandos de alto nível gerados pelo Configurador Web e servindo como camada de abstração das funcionalidades do Google Awareness API. Assim como a Awareness API, a biblioteca é dividida em duas partes: Snapshot e Fence. A classe Snapshot é composta de métodos estáticos para aquisição do contexto atual, além de atalhos para alguns métodos da Awareness API. Em termos de padrões de projeto, funciona como um *Façade*, incorporando todos os métodos da Snapshot API.

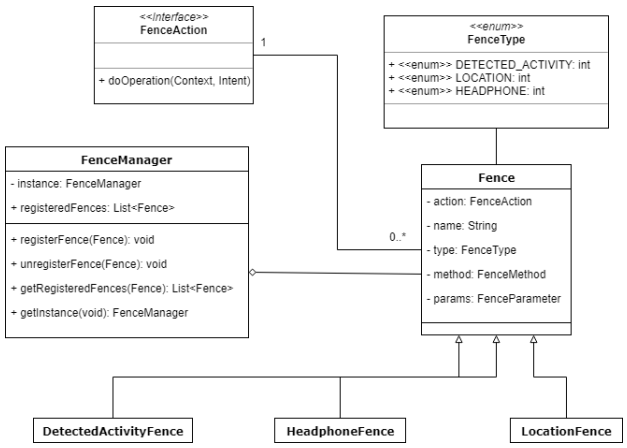

Figura 3: Diagrama de classe simplificado da EasyContext Fence

A parte Fence do EasyContext possui outras classes de apoio, como as mostradas na Figura 3. A interface FenceAction serve como um facilitador para escrita de ações de resposta à mudança de contexto. O desenvolvedor pode implementar esta classe e sobrescrever o método *doOperation* para especificar a ação, de forma a favorecer o reúso de ações. A classe Fence serve como uma representação de uma regra de contexto genérica. Sua instanciação é apenas utilizada pelo configurador de regras contextuais. As classes de apoio para Fence têm como função especificar a regra de contexto em uma forma mais amigável, categorizando e encapsulando os métodos de aquisição de contexto às categorias corretas. Cada classe de apoio possui um Builder para a especificação de parâmetros extras. O FenceManager é um *Singleton* para a gerência de regras contextuais. Esta classe tem como objetivo registrar e remover regras contextuais da pilha do Awareness API, além de realizar o registro dos *Broadcasts* em paralelo, fora do domínio do programador. Devido a este controle, o código que o desenvolvedor deve escrever é reduzido, pois não há mais a necessidade de reescrever diversas linhas de código que cuidam do gerenciamento destes *Broadcasts*.

A classe Configurator configura determinada tela do aplicativo e a deixa preparada para ser sensível ao contexto de forma automática. Contido, há um leitor JSON que lê o documento gerado pelo configurador web e cataloga todas as regras de contexto em um formato interpretável pelo Google Awareness API. Este documento deve estar localizado na pasta */res/raw/*. O Configurator também se utiliza de gerência de regras de contexto utilizando o FenceManager, que registra e remove regras quando sua observação não é mais necessária para a sensibilidade ao contexto. Um exemplo em que este fenômeno acontece é quando o usuário troca de atividade dentro da aplicação. Dependendo da *Activity* em que o Configurator é invocado, é catalogado registrado as regras que estão relacionadas com tal *Activity*. Abaixo se encontram duas listagens comparando as duas abordagens, a primeira utilizando o *framework* EasyContext para a configuração de regras de contexto. O documento de regras é o mesmo apresentado na Figura 2. A segunda listagem utiliza a API padrão do Google Awareness.

Listagem 1: Exemplo de código do Configurator EasyContext

```
1  @Override
2  protected void onCreate(Bundle
       savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5      Map<String, FenceAction> actions = new
           HashMap<>();
6      actions.put("action1", new MyCustomAction());
7      Configurator.init(MainActivity.this, actions);
8      }
9  }
```

Listagem 2: Exemplo de código padrão sem EasyContext

```
1  @Override
2  protected void onCreate(Bundle
       savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5      BroadcastReceiver action = new
           BroadcastReceiver{
6      @Override
7          public void onReceive(Context context,
               Intent intent){
8          FenceState state =
               FenceState.extract(intent)
9          switch(state.getCurrentState()){
10             case FenceState.True:
11                 System.out.println("Headphone
                       connected");
12                 break;
13             case FenceState.False:
14                 System.out.println("Headphone
                       disconnected");
15                 break;
16         }
17         registerReceiver(action, new
               IntentFilter("headphone");
18         PendingIntent pi =
               PendingIntent.getBroadcast(this, 0,
               new Intent("headphone"),
               PendingIntent.FLAG_CANCEL_CURRENT);
19         AwarenessFence fence =
               HeadphoneFence.during(
20         HeadphoneState.PLUGGED_IN
21         );
```

```
22          Awareness.getFenceClient(this)
23          .updateFences(new
                FenceUpdateRequest.Builder()
24          .addFence("headphone",fence,pi).build());
25          }
26      }
27  }
```

Ainda que o programador dedique uma classe que implemente BroadcastReceiver no seu projeto, ele deve realizar o registro do mesmo no escopo onde deseja sensibilidade ao contexto, prejudicando a legibilidade do código.

4 PROVA DE CONCEITO

Foi feito uma pequena aplicação como prova de conceito para ilustrar o uso do EasyContext (Figura 4). A aplicação utiliza o Easycontext para que o telefone se comporte nos seguintes cenários: vibrar quando o usuário estiver andando ou conectar um fone de ouvido e ligar a lanterna quando o usuário estiver andando e com o fone de ouvido conectado ao mesmo tempo. A Listagem 3 exibe um trecho de seu código.

Listagem 3: Prova de Conceito do EasyContext

```
1   protected void onCreate(Bundle
        savedInstanceState) {
2       super.onCreate(savedInstanceState);
3       setContentView(R.layout.activity_main);
4       HeadphoneFence pluggedIn = new
            HeadphoneFence("plugged",
5       HeadphoneMethod.HEADPHONE_PLUGGING_IN,
6       new VibeAction(),
7       null);
8       DetectedActivityFence walking = new
            DetectedActivityFence("walking",
            DAMethod.DA_STARTING, new VibeAction(),
9
10              new DetectedActivityParameter
11              .Builder()
12              .addActivityType
13              (DetectedActivity.WALKING)
14              .build());
15      AndFence and = new AndFence("composition",new
            LightAction(this),pluggedIn,walking);
16      FenceManager manager =
            FenceManager.getInstance(this);
17      manager.registerFence(pluggedIn);
18      manager.registerFence(walking);
19      manager.registerFence(and);
20      }
```

5 CONSIDERAÇÕES FINAIS

Este artigo propõe uma abordagem para facilitar o desenvolvimento de aplicativos CAM usando a Awareness API, o *framework* EasyContext, que encapsula a complexidade da configuração e uso da API. A abordagem ainda é passível de melhorias. A primeira é a falta de suporte para criação de regras de contexto agregadas (composições de Fences) utilizando o configurador web. A forma de se representar estas regras podem ser muito complexas e esta complexidade pode influenciar no tamanho do arquivo de configuração. O Configurator

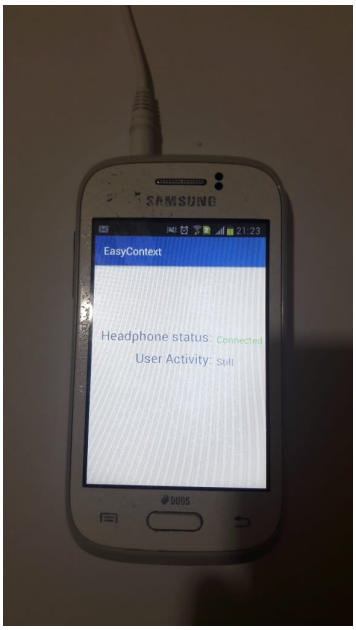

Figura 4: Prova de Conceito em um hardware Android

Java ainda leva um tempo para fazer a leitura e registro das regras de contexto. Uma solução seria otimizar partes do documento para serem lidas apenas pelas Activities as quais estas regras se referem. Uma avaliação baseada em técnicas de Engenharia de Software Experimental também está em andamento para comprovar os ganhos da abordagem.

REFERÊNCIAS

[1] Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. 2012. A Survey of Context Data Distribution for Mobile Ubiquitous Systems. *ACM Comput. Surv.* 44, 4, Article 24 (Sept. 2012), 45 pages.

[2] Anind K. Dey. 2001. Understanding and Using Context. *Personal Ubiquitous Comput.* 5, 1 (Jan. 2001), 4–7. https://doi.org/10.1007/s007790170019

[3] Paulo A. S. Duarte, Felipe M. Barreto, Francisco A. A. Gomes, Windson Viana, and Fernando A. M. Trinta. 2015. CRITiCAL: A Configuration Tool for Context Aware and mobiLe Applications. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, Vol. 2. 159–168.

[4] Eric Freeman, Elisabeth Robson, Bert Bates, and Kathy Sierra. 2004. *Head First Design Patterns: A Brain-Friendly Guide.* OReilly Media, Inc.

[5] José R. Hoyos, Jesús García-Molina, and Juan A. Botía. 2013. A Domain-specific Language for Context Modeling in Context-aware Systems. *J. Syst. Softw.* 86, 11 (Nov. 2013), 2890–2905.

[6] Fei Li, S. Sehic, and S. Dustdar. 2010. COPAL: An adaptive approach to context provisioning. In *2010 IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications.* 286–293.

[7] Marcio E. F. Maia, Andre Fonteles, Benedito Neto, Romulo Gadelha, Windson Viana, and Rossana M. C. Andrade. 2013. LoCCAM - Loosely Coupled Context Acquisition Middleware. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13).* ACM, New York, NY, USA, 534–541.

[8] André C. Santos, João M. P. Cardoso, Pedro C. Diniz, and Diogo R. Ferreira. 2013. Specifying Adaptations through a DSL with an Application to Mobile Robot Navigation. In *2nd Symposium on Languages, Applications and Technologies, SLATE 2013, June 20-21, 2013 - Porto, Portugal.* 219–234.

[9] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu. 2018. Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey. *IEEE Internet of Things Journal* 5, 1 (Feb 2018), 1–27. https://doi.org/10.1109/JIOT.2017.2773600

[10] Ö. Yürür, C. H. Liu, Z. Sheng, V. C. M. Leung, W. Moreno, and K. K. Leung. 2016. Context-Awareness for Mobile Sensing: A Survey and Future Directions. *IEEE Communications Surveys Tutorials* 18, 1 (Firstquarter 2016), 68–93. https://doi.org/10.1109/COMST.2014.2381246