Pós-Graduação em Ciência da Computação

Willamys Gomes Fonseca Araújo

UMA ABORDAGEM ORIENTADA A MODELOS PARA GERAÇÃO DE APLICAÇÕES BASEADAS EM INTERNET DAS COISAS PARA SMARTPHONES

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Willamys Gomes Fonseca Araújo

UMA ABORDAGEM ORIENTADA A MODELOS PARA GERAÇÃO DE APLICAÇÕES BASEADAS EM INTERNET DAS COISAS PARA SMARTPHONES

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Vinicius Cardoso Garcia*

RECIFE
2017

Eu dedico esta dissertação a todos os meus familiares,
amigos e professores que me deram o suporte necessário
para a conclusão com êxito desse sonho.

Agradecimentos

Primeiramente, gostaria de agradecer a Deus, por todas as oportunidades e graças recebidas nesse período. E por diversas vezes que achava que não tinha capacidade de conseguir e me mostrava o contrário.

A meus pais, Sr. Ione e Dona Maria das Neves, pela educação, simplicidade, acolhimento, amor e dedicação que me fizeram a pessoa que sou hoje.

A minha esposa e companheira, Carla Katiane, que sempre me apoiou com as palavras certas e me suportando nos momentos de estresse.

Ao professor/orientador Vinicius Cardoso Garcia, pela paciência e atenção nas correções, além de apresentar o melhor caminho para seguir nesta jornada.

Ao IFCE Campus Tianguá na figura do Diretor Jackson Nunes e Vasconcelos, pela oportunidade, compreensão e paciência para permitir que esse objetivo pudesse ser alcançado.

Agradeço também a toda turma do MProfRedes2014 por terem dividido comigo esta jornada, principalmente aos "Pôneis": Danyel, David, Leonardo e Rony. Aos amigos Patrício e Hugo por me apoiarem na estadia em Recife nos períodos de aula.

"Don’t let others define you. Don’t let the past confine you. Take charge of your life with confidence and determination and there are no limits on what you can do or be."

—MICHAEL JOSEPHSON

Resumo

A internet das coisas (IoT) traz à tona a ideia de dispositivos de computação identificáveis, que são conectados à internet e incorporados a objetos do nosso dia-a-dia ou ligados a animais ou pessoas (ATZORI; IERA; MORABITO, 2010), ou seja, poderiam ser vistos também pela ótica de que pessoas possam vir a atuar como sensores. Em consonância à IoT, a utilização de smartfphones tem tornado-se cada vez mais popular e acessível. De acordo com LEE; LEE (2015), a evolução das principais tecnologias IoT em relação a softwares e algoritmos com projeção para depois de 2020 são: softwares orientados para o usuário, o IoT invisível, easy-to-deploy de softwares IoT, colaboração things-to-humans, e IoT para todos. Nesse sentido, como forma de atender as premissas supracitadas, a utilização de técnicas de Programação Generativa (CZARNECKI et al., 2000) em que coloca seu foco na maximização da automação do desenvolvimento de aplicativos; e a abordagem da Arquitetura Orientada a Modelos (WARMER; KLEPPE, 2003), cujo diferencial está no fato do desenvolvimento ser baseado nas atividades de modelagem, trazendo flexibilidade e portabilidade para os softwares desenvolvidos; ambos podem ser vistos como uma alternativa para a criação de aplicações no âmbito de pessoas como sensores. Diante deste contexto, o presente trabalho apresenta uma abordagem orientada a modelos para o desenvolvimento de aplicativos na plataforma Android, dentro do domínio de pessoas como sensores, por meio da combinação de componentes de software reutilizáveis e os sensores presentes nos *smartphones*. Como forma de avaliar a abordagem, foi realizado um estudo de caso a fim de mensurar o reúso de software das aplicações. A abordagem propiciou bons resultados nas aplicações desenvolvidas, com bons índices reutilização de código, além de entregá-la pronta pra uso.

Palavras-chave: *Generative programming*, modelos, *Internet of Things*, *smartphones*

Abstract

The Internet of Things (IoT) brings to light an idea of identifiable computing devices, which are connected to the internet and incorporated into objects of our daily life or linked to animals or people (ATZORI; IERA; MORABITO, 2010); That is to say, to be well seen also by an optics of people with an actuator like sensors. In line with IoT, the use of smartphones has become increasingly popular and accessible. According to LEE; LEE (2015), an evolution of leading IoT technologies for software and algorithms projected beyond 2020 are user-oriented software, invisible IoT, easy to implement IoT software, things-to-human collaboration, and IoT for all. In this sense, as a way of meeting the above-mentioned aspirations, Use of Generative Programming techniques (CZARNECKI et al., 2000) in which its focus on maximizing the automation of application development; The model-oriented architecture approach (WARMER; KLEPPE, 2003), whose differential lies in the fact of development, based on modeling activities, bringing flexibility and portability to the developed software; Both can be seen as an alternative to creating applications to people as sensors. In this context, the present work presents a model-oriented approach to the development of applications on the Android platform, within the domain of people as sensors, through the combination of reusable software components and sensors in our smartphones. The approach provided good results in the developed applications, with good code reuse indexes, besides delivering it ready for use.

Keywords: Generative programming, Models, Internet of Things, smartphones

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1 Introdução

Neste capítulo são apresentadas as motivações e a problematização do tema que impulsionaram a realização deste trabalho. Serão descritos também os objetivos geral e específicos. Toda a estrutura utilizada para a elaboração deste trabalho será exposta ao final deste capítulo.

1.1 Contexto e Motivação

Atualmente, nossa rotina e senso de mundo tem mudado drasticamente pelo advento de novas tecnologias. Não abrimos mão de um dispositivo que tenha acesso a rede. Queremos saber qual a previsão climática para o dia, saber as mais novas notícias, se o ônibus que vamos pegar pra o trabalho já passou. Essas informações são obtidas através de uma variedade de sensores, atuadores, identificadores por Radio Frequência (RFID), *smartphones*, todas essas "coisas", estão conectadas e cooperando entre si, compondo um novo paradigma, conhecido como Internet das Coisas, do inglês *Internet of Things* (IoT).

As aplicações oferecidas pela IoT torna possível o desenvolvimento de um grande número de aplicações que podem vir a melhorar a qualidade de vidas das pessoas, em casa, no trabalho, quando doentes, numa viagem, dentre outros. Porém esses ambientes são equipados somente com inteligência primitiva, sem uma comunicação implementada entre eles. Uma vez permitida a troca de informações nos ambientes, uma gama de aplicações podem ser construídas agrupadas no âmbito do transporte e logística, saúde, ambientes inteligentes (casa, escritório, no campo), pessoal e social (ATZORI; IERA; MORABITO, 2010).

A internet das coisas (IoT) traz à tona a ideia de dispositivos de computação identificáveis, que são conectados à internet e incorporados a objetos do nosso dia-a-dia (ou coisas) ou ligados a animais ou pessoas (ATZORI; IERA; MORABITO, 2010), ou seja, poderia ser vista também pela ótica de que pessoas possam vir a atuar como sensores. A atuação das pessoas como sensores pode ser associada e/ou intermediada a utilização de *smartphones*, os quais possuem características permissivas para isso, como GPS, câmera, sensores de proximidade, luminosidade, temperatura, dentre outros, além de ter se tornado cada vez mais popular e acessível. Segundo ANNIE (2017), em sua retrospectiva 2016, o tempo gasto em aplicativos cresceu mais de

20%, para quase 900 bilhões de horas em 2016, além disso, em todo o mundo, os downloads aumentaram 15%, sendo assim em mais de 13 bilhões, tanto no iOS quanto no Google Play. Isso é apenas um sinal de que a economia de aplicativos globais viu um crescimento saudável durante o ano passado.

Para o domínio de atuação referido, é necessário que sejam desenvolvidas aplicações e o processo de criação destas deve ser ágil e de qualidade, a fim de permitir o seu uso o mais breve possível. Assim, o mercado impulsiona para o uso de ferramentas intuitivas, com baixa curva de aprendizado, baixo custo, customização, fácil manutenção e padronização. A Engenharia de Software (ES), diz que o desenvolvimento de sistemas de software deve possuir boa relação custo benefício, ocupando-se de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção deste sistema, depois que ele entra em operação (SOMMERVILLE, 2011). Assim a preocupação com a qualidade do software, agilidade e sua relação custo-benefício é um fato crucial.

Nesse sentido, como forma de atender as premissas supracitadas, a utilização de técnicas de Programação Generativa (CZARNECKI et al., 2000) em que coloca seu foco na maximização da automação do desenvolvimento de aplicativos; e a abordagem da Arquitetura Orientada a Modelos (WARMER; KLEPPE, 2003), cujo diferencial está no fato do desenvolvimento ser baseado nas atividades de modelagem, trazendo flexibilidade e portabilidade para os softwares desenvolvidos; ambos podem ser vistos como uma alternativa para a criação de aplicações no âmbito de pessoas como sensores.

Diante deste contexto, a proposta deste trabalho é apresentar uma abordagem de desenvolvimento de aplicativos na plataforma Android, voltada para a pessoas como sensores, por meio da combinação de componentes de software reutilizáveis e os sensores presentes nos *smartphones*, baseando-se nas técnicas de programação generativa e nos preceitos do *Model Driven Architecture* (MDA). A fim de evidenciar a sua viabilidade, uma ferramenta foi desenvolvida e aplicada em um estudo de caso.

1.2 Problematização

Historicamente, o desenvolvimento de software começou há décadas como um processo artesanal, desde então, o mercado vem incitando a transformá-lo em um processo quase que industrial. Isso é reflexo não só do avanço tecnológico, mas também do nível de exigência requerido.

Essa migração trouxe melhorias, mas algumas lacunas não estão totalmente preenchidas como: falhas na obtenção dos requisitos iniciais, documentação ineficiente, falta de treinamento das tecnologias utilizadas para o desenvolvimento (Linguagem de programação, *Integrated Development Environment* (IDE), ferramentas de modelagem), dentre outras. Conforme SOMMERVILLE (2011), os softwares devem possuir caraterísticas essenciais para se caracterizarem como bem projetados, para isso deve ser observado: a facilidade de manutenção (software deve

evoluir para atender as necessidades do cliente); nível de confiança (confiabilidade, proteção e segurança); eficiência (rapidez de reposta, tempo de processamento, memória) e facilidade de uso (interface apropriada, documentação adequada).

O mercado cada vez mais competitivo, impulsiona para o uso de ferramentas intuitivas, com baixa curva de aprendizado, baixo custo, customização, fácil manutenção e padronização. A ES, disciplina da engenharia, diz que o desenvolvimento de sistemas de software deve possuir boa relação custo benefício, ocupando-se de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção deste sistema, depois que ele entra em operação (SOMMERVILLE, 2011).

Diversas ferramentas foram criadas e continuam a aparecer com intenção de atender aos anseios de engenheiros e gerentes de projeto de software. De acordo com LEE; LEE (2015), a evolução das principais tecnologias IoT em relação a softwares e algoritmos com projeção para depois de 2020 são: softwares orientados para o usuário, o IoT invisível, *easy-to-deploy* de softwares IoT, colaboração *things-to-humans*, e IoT para todos.

A grande maioria das IDE's permitem o desenvolvimento de aplicações para sensores presentes em Arduíno, Raspberry e Intel, e utilizam a plataforma Android somente para controlá-los. Outro ponto é que as ferramentas disponibilizadas que exploram sensores presentes em *smartphones*, não apresentam uma *Graphic User Interface* (GUI) a fim de permitir a orquestração de componentes de software e/ou diagramação como a *Unified Modeling Language* (UML).

Nesse sentido, seguindo as tendências da IoT para um desenvolvimento mais simplificado (*easy-deploy*, diagramação) e ainda permitir a colaboração homem-máquina, a pergunta que fundamentou a pesquisa foi: Como realizar a combinação entre componentes de software reutilizáveis, baseado em técnicas de programação generativa e MDA para a criação de aplicativos Android voltados para pessoas como sensores?

1.3 Objetivos

Após um breve contexto e definição do problema, o objetivo geral deste trabalho é demonstrar e avaliar o reúso de software na abordagem proposta, a qual é baseado na composição de componentes reutilizáveis de software permitindo a transformação automatizada de modelos MDA em código fonte. O código fonte gerado, de forma automatizada, permitirá o desenvolvimento de aplicativos que fazem uso de sensores presentes em *smartphones*, e seguem os conceitos da IoT. Para que esse objetivo seja alcançado com êxito foram definidos os seguintes objetivos específicos:

1. Definir quais componentes de software reutilizáveis serão combinados para a elaboração da abordagem;
2. Selecionar técnicas e procedimentos que irão estruturar a abordagem;

3. Disponibilizar uma ferramenta que permita orquestração dos componentes definidos e geração do código fonte das aplicações;
4. Avaliar a abordagem por meio de um estudo de caso com aplicações construídas com a ferramenta.

1.4 Organização da Dissertação

A dissertação está organizada em cinco capítulos, além da seção referente às referências bibliográficas. O primeiro capítulo contém esta introdução.

No Capítulo 2, será abordado toda a fundamentação teórica que justifica o desenvolvimento da abordagem proposta, principalmente sobre o Desenvolvimento Orientados a Modelos e Programação Generativa. Alguns trabalhos relacionados foram analisados, a fim de extrair seus pontos fortes e fracos.

No Capítulo 3 é apresentado a abordagem definida, a qual combina algumas técnicas e procedimentos para a geração das aplicações dentro do domínio de pessoas como sensores.

No Capítulo 4 é apresentada a realização de um estudo de caso e seus resultados, para avaliar a abordagem.

E no quinto e último capítulo, as conclusões referentes ao trabalho, destacando as principais contribuições e trabalhos futuros.

2

Fundamentação Teórica

Neste capítulo serão apresentados os conhecimentos teóricos que orientaram a pesquisa, dando suporte à implementação do método de geração de aplicações de internet das coisas para *smartphones* e, consequentemente, ao alcance dos objetivos definidos no trabalho.

2.1 Internet da Coisas (IoT)

Na introdução foi falado de forma breve sobre a IoT. Esta refere-se a dispositivos de computação identificáveis, que são conectados a internet e incorporados a objetos do nosso dia-a-dia. A IoT também é vista como uma infra-estrutura de rede dinâmica com auto capacidade, baseada em padrões e protocolos de comunicação interoperáveis, onde coisas físicas e virtuais tem identidades, atributos físicos, personalidades virtuais e usam interfaces inteligentes e estão perfeitamente integrados na informação da rede (VERMESAN; FRIESS, 2014). Assim, dispositivos identificáveis na rede, que usam algum protocolo de comunicação e cooperam entre si, como o RFID, Tecnologias *Wireless*, como *Near Field Communication* (NFC), *WiFi*, *ZigBee* (ALLIANCE, 2015), podem ser considerados sistemas do paradigma IoT.

A IoT pode ser analisada seguindo a composição de três visões: orientada as coisas, a internet e a semântica. A primeira, foca nos "objetos", e em encontrar um paradigma capaz identificar e integrá-los; a segunda, onde a ênfase é sobre o paradigma de rede e em explorar o protocolo IP, para estabelecer uma eficiente ligação entre dispositivos, e ao mesmo tempo simplificando o IP, de modo que ele possa ser utilizado em dispositivos com uma capacidade muito limitada; a terceira e última, visa a utilização de tecnologias semânticas, descrevendo objetos e dados de gestão, para representar, armazenar, interconectar e gerenciar a enorme quantidade de informações fornecidas pelo crescente número de objetos da Internet das coisas. Na Figura 2.1 pode-se ter uma ideia de como é composto o paradigma IoT, de acordo com (ATZORI; IERA; MORABITO, 2010)

Contudo, essas três visões trazem à tona infinitas soluções que podem ser aplicadas a qualquer nicho de mercado, como na educação, saúde,segurança, transportes, comunicação, dentre outras que também permitirão melhorias para a população. Todas essas aplicações serão

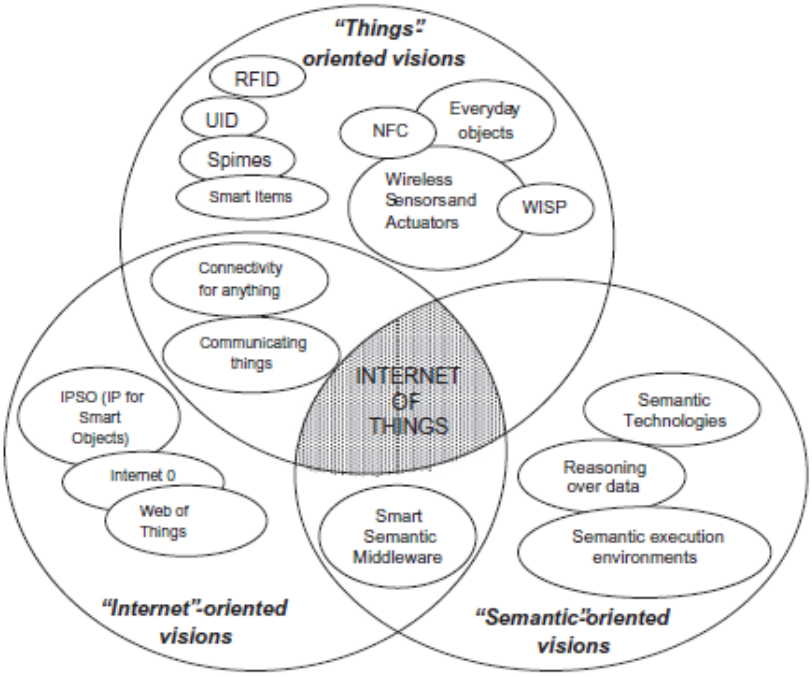


Figura 2.1: Paradigma da Internet das Coisas (IoT)
ATZORI; IERA; MORABITO (2010)

possíveis através de ferramentas que auxiliem na construção, contribuindo para que seja uma ambiente intuitivo, ágil e de qualidade.

Segundo LEE; LEE (2015) a IoT possui cinco tecnologias amplamente utilizadas para a implantação de produtos e serviços bem-sucedidos:

1. *Radio Frequency Identification* (RFID);
2. *Wireless Sensor Networks* (WSN);
3. *Middleware*;
4. *Cloud Computing*; e
5. *IoT Application software*.

Sobre o RFID pode-se entender como um das principais tecnologias para IoT, desde que o conceito de IoT foi originalmente concebido. Este foi utilizado para identificar qualquer objeto em torno do Mundo de uma maneira global e original. Segundo WANT (2006) RFID pode ser dividida em duas classes: ativa e passiva. Uma *tag* ativa requer uma fonte de energia (interna, como bateria; externa, rede elétrica); uma *tag* passiva, não requer fonte de energia, nem manutenção, porém necessita de uma antena leitora, responsável pela energia e comunicação com a *tag*. As aplicações para o RFID são inúmeras, por exemplo, usando uma *tag* ativa, é possível integrar tecnologia celular e GPS para localizar um carro roubado. A passiva pode ser utilizada para identificação de livros em uma biblioteca.

As redes de sensores sem fio (WSN) consistem em dispositivos com sensores autônomos distribuídos para monitorar condições físicas ou ambientais e cooperar com sistemas RFID para

melhor acompanhar o status de "coisas", como sua localização, temperatura, e movimentos (LEE; LEE, 2015). Uma de suas vantagens é que a comunicação se dá entre os sensores não precisando de um leitor entre eles, comunicação *machine-to-machine*(ATZORI; IERA; MORABITO, 2010).

Segundo COULOURIS et al. (2013) o termo *middleware* pode ser visto como uma camada de software que fornece uma abstração de programação, assim como o mascaramento da heterogeneidade das redes, do hardware, dos sistemas operacionais e das linguagens de programação subjacentes. Dessa forma, o *middleware* poupa o desenvolvedor de se atentar a problemas complexos que muitas vezes não estão ligadas as aplicações. Em BANDYOPADHYAY et al. (2011) comenta que *middleware* em IoT pode fornecer uma *Application programming interfacing* (API) para comunicações de camada física e serviços necessários para aplicativos, ocultando detalhes de diversidade. Dessa forma, uma infraestrutura complexa distribuída de IoT com inúmeros dispositivos heterogêneos, requer que o desenvolvimento de aplicações e serviços seja simplificado, e o uso de *middleware* é ideal para o desenvolvimento de aplicativos IoT (LEE; LEE, 2015).

Uma das premissas para que a IoT desponte ainda mais, é o acesso a rede e aos recursos disponibilizadas por ela. Para isso, a IoT tem se apoderado dos serviços oferecidos pela *cloud computing*. O *National Institute of Standards and Technology (NIST)* define computação em nuvem como um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (redes, servidores, armazenamento, aplicações e serviços). Para se ter uma ideia, são mais de 9 bilhões de dispositivos conectados e deverão crescer mais rapidamente e atingir 24 bilhões até 2020 (GUBBI et al., 2013). Nesse sentindo, a quantidade de dados geradas por esses dispositivos e que necessitam serem armazenadas é inimaginável. Um novo paradigma tem surgido pela integração da IoT e a *cloud computing*, e tem sido chamada de *Cloud of Things* (CoT) (AAZAM et al., 2014). Porém, nem tudo é perfeito nessa interação, problemas com suporte a protocolos, eficiência energética, qualidade de provisionamento de serviços, armazenamento de dados de localização, segurança e privacidade, armazenamento de dados desnecessários, além de outros pontos, ainda precisam ser melhorados. Acredita-se que a utilização de filtros ou condições específicas antes da sincronização com a nuvem possam trazer menor impacto para a *cloud*.

A aplicações desenvolvidas para IoT podem ser orientadas tanto para indústria quanto para usuários. De acordo com LEE; LEE (2015), a IoT pode ser categorizada em 3 grupos para aplicações empresariais: (1) monitoramento e controle, (2) grandes dados e análise de negócios e (3) compartilhamento de informações e colaboração. Em ATZORI; IERA; MORABITO (2010), o mesmo agrupa as aplicações no âmbito do transporte e logística, saúde, ambientes inteligentes (casa, escritório, no campo), pessoal e social. Já em GUBBI et al. (2013) categoriza as aplicações em 4 domínios: Pessoal e Casa, Empresas, Utilidades, Móvel. Observando de uma maneira mais geral, essas subdivisões impostas se completam, e às vezes absorvem umas às outras. Por exemplo, a categoria de monitoramento e controle pode incluir aplicações do âmbito de transportes, saúde e ambientes inteligentes. O domínio Pessoal e Casa pode absorver social,

pessoal e ambientes inteligentes, por exemplo.

Analisando o futuro das aplicações em IoT, SUNDMAEKER et al. (2010) trás projeções em diversas áreas, na Figura 2.2 temos algumas que foram destacadas. Em relação a redes, fator primordial para a IoT, existem tendências para que essa se torne mais autônoma, ou seja, se auto regule, corrija-se sem atuação humana. Sobre o desenvolvimento de software e algoritmos, o caminho vislumbra para aplicações orientadas a usuários, inteligência distribuída, colaboração *machine-to-machine* e *machine-to-human*, e ainda a fácil implantação de aplicações IoT (*easy-deploy*), a fim de permitir que o software evolua tanto quanto o hardware. As arquiteturas de tecnologia estão convergindo para arquitetura baseadas em contexto e/ou adaptativas, possivelmente pela heterogeneidade de dispositivos e interfaces existentes. Por fim, o hardware tende a evoluir para a miniaturização dos dispositivos e componentes destes, com o intuito de melhorar a eficiência energética e atuação em diversos ambientes.

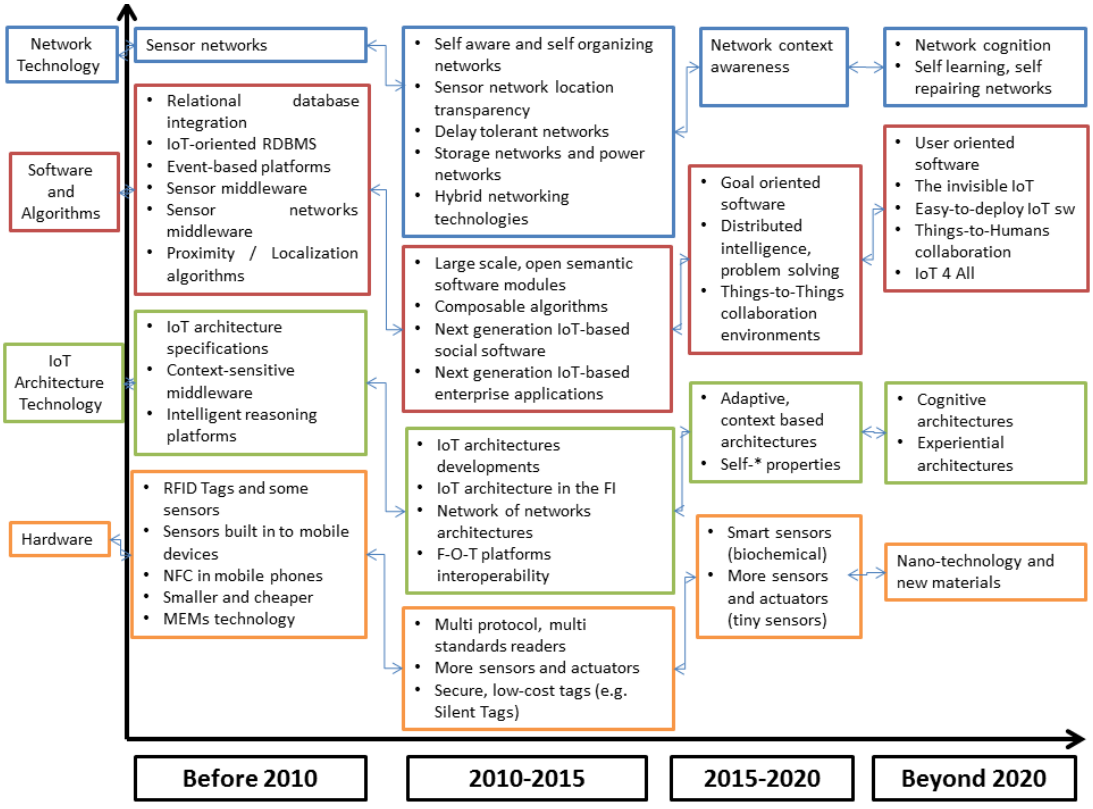


Figura 2.2: Tendências para o futuro da IoT
Adaptado de (SUNDMAEKER et al., 2010)

O presente trabalho observa essas tendências como oportunidade e está indo a favor dessa perspectiva, uma vez que a pesquisa tende a disponibilizar um método *easy-deploy* para o desenvolvimento de aplicações para IoT e ainda a interação homem-máquina (machine-to-human) por fornecer os insumos básicos para obtenção de informações através do sensores presentes nos *smartphones*, que pode ser conhecida como *people as sensors* ou *crowdsensing*.

2.1.1 Pessoas como Sensores

De acordo com RATTI; NABIAN (2010), dentre as formas de monitorar uma cidade, como forma de obter informações necessárias que podem auxiliar na tomada de decisões, destacam-se três mecanismos: sensoriamento viral, redes de sensores e baseado em pessoas (*crowdsensing*).

O primeiro está relacionado a algoritmos computacionais que tem a capacidade de detecção, e se infiltram em redes digitais, como vírus, alimentando e desenvolvendo-se através dos vestígios digitais deixados pelos usuários, de forma voluntária ou involuntária. Esses vestígios podem ser obtidos toda vez que um cartão de crédito é usado, uma mensagem de texto ou um e-mail é enviado, uma consulta do Google é enviada, permitindo construir um perfil ou mapa de uso da rede.

O segundo, sobre redes de sensores já foi abordado no tópico anterior. O terceiro e foco deste trabalho, baseia-se na utilização dos dispositivos móveis portados pelas pessoas, como forma de extrair informações (fotos carregadas de eventos populares, *tweets* enviados sobre novos acontecimentos em tempo real, dentre outros), tudo isso tanto a nível individual quanto coletivo e/ou comunitário chamada de *crowdsensing*. Similarmente a RATTI; NABIAN (2010), em GUO et al. (2014) *crowdsensing* é visto como um novo paradigma de percepção que capacita os cidadãos comuns para contribuir com dados detectados ou gerados a partir de seus dispositivos móveis, agrega e funde os dados na nuvem para extração de inteligência de multidão e entrega de serviços centrada nas pessoas.

Esse novo paradigma surge como alternativa aos sensores físicos, estes além de estáticos, angariam custos elevados para sua implantação e muitas vezes podem não condizer com as condições econômicas de uma cidade que pretende fazer uso dela (SILVA, 2014). Os sensores físicos geralmente estão espalhados em certas áreas de interesse, focando em propósitos de aplicações específicas (CAMPBELL et al., 2006), ou seja, usados único e exclusivamente, para aquela situação e/ou área. As pessoas, diferentemente dos sensores físicos, estão em constante movimento, presentes em diversos locais, possibilitando uma maior abrangência na área monitorada (MA; ZHAO; YUAN, 2014).

Estudos recentes apontam o potencial dos dados provenientes de dispositivos móveis atrelados a pessoas.Através do uso dos sensores presentes nos dispositivos móveis pessoais, todas as pessoas são convidadas a participar da coleta e compartilhamento de dados do seu cotidiano que são importantes para eles, e possivelmente, para outras partes interessadas como grupos comunitários, indústria local, ciências da computação, engenheiros, cientistas sociais, organizações de saúde ambiental, planejadores do espaço urbano, governos locais e nacionais, etc (PAULOS, 2009).

Nesse sentido, o presente trabalho busca identificar quais os sensores mais indicados para a obtenção dessas informações, e a partir disso, propor uma forma de desenvolver aplicativos que permitam interagir com essa realidade com qualidade e rapidez.

2.2 Smartphones

2.2.1 Sensores

Devido a estarem cada vez mais populares e com uma densidade per capita maior que os computadores, os *smartphones* estão aparecendo nos dois últimos anos como a chave para a porta de entrada dos serviços e produtos financeiros. Atualmente podemos dizer que os *smartphones* são dispositivos presentes em todos os lugares. Em uma pesquisa realizada por MEIRELLES (2016) a quantidade de *smartphones* presentes no brasil até maio de 2016, era de 168 milhões e a tendência para 2017/2018 é que chegue a 236 milhões,ou seja, estará ultrapassando a quantidade de habitantes.

Os *smartphones* de hoje estão equipados com capacidades de processamento computacional e sensores múltiplos que podem ser usados para monitorar atividades físicas, temperatura ambiente e umidade, pressão barométrica, estado de saúde, movimentos e outras condições ambientais que cercam os usuários (SALIM; HAQUE, 2015).

Nesta era smartphone, a interação com o mundo real é realizada através de várias tecnologias integradas em *smartphones* e plataformas móveis. Os *smartphones* são habilitados com recursos de geo-localização através de GPS (Figura 2.3), sistemas de localização em tempo real e geo-localização assistida, através das redes Wi-Fi implantadas. Outra tecnologia que também está presente em sua grande maioria é a Câmera. Assim, uma solução baseada nisso pode ser usada para digitalizar códigos de barras (1 dimensão) e códigos QR (2 Dimensões). Dessa forma, as coisas inteligentes podem ser identificadas lendo código de barras e códigos QR (JARA et al., 2014).

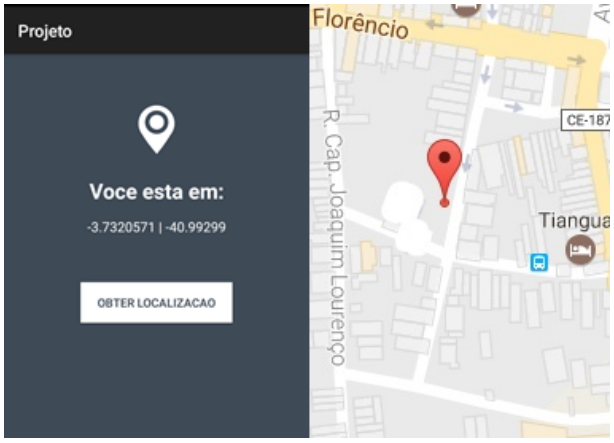


Figura 2.3: Localização usando GPS

Já foi mencionado que os *smartphones* possuem localização habilitada, câmeras estão disponíveis para ler códigos de barras / códigos QR e, finalmente, a última geração de plataformas móveis é alimentado com NFC que permite interação com apenas uma aproximação entre o *smartphone* e os objetos (JARA et al., 2014).

No presente trabalho foi dado ênfase a utilização do GPS e Câmera, que são os sensores mais populares nos celulares atuais, além de que dispositivos com NFC ainda estão com preço bastante elevado. Uma pesquisa mais aprofundada com a utilização do NFC pode ser realizada em breve.

2.2.2 Android

É a plataforma *mobile* mais popular do mundo. Atualmente possui centenas de milhares de dispositivos móveis em mais de 190 países ao redor do mundo. Foi desenvolvida pela Google, em colaboração com a Open Handset Alliance (ANDROID, 2016a).

O Android permite aos desenvolvedores criarem suas próprias aplicações na linguagem de programação Java, uma vez que a mesma possui um *Software Development Kit* (SDK) próprio o qual engloba os *packages* padrões do Java (como o gerenciamento de strings, controle de inputs e outputs, pacotes math, dentre outros), como os *packages* desenvolvidos pela Google.

Figura 2.4: Google Android e alguns aplicativos

Outro ponto interessante no Android é que não há diferença entre os aplicativos embutidos, que já vêm instalados no dispositivo, dos aplicativos que são criados com o SDK. Dessa forma, o usuário pode escrever aplicações poderosas para explorar os recursos disponíveis no dispositivo. Abaixo algumas características (ANDROID, 2016a):

- *Application framework* proporciona a reutilização e substituição de componentes;
- *Dalvik virtual machine* otimizada para dispositivos móveis;
- *Browser* integrado baseado no *Webkit engine*;
- Gráficos Otimizados possui uma biblioteca 2D e 3D baseada na especificação OpenGL;
- *SQLite engine* de banco de dados;
- Suporte multimídia para áudio, vídeo e formatos de imagem;
- Telefonia GSM;
- Bluetooth, EDGE, 3G, 4G LTE e Wifi, NFC;
- Câmera, GPS e acelerômetro;

- Rico ambiente de desenvolvimento, incluindo um emulador de dispositivo, ferramentas de depuração, memória, desempenho e um plugin para a IDE Eclipse;
- Atualmente desenvolveu sua ferramenta própria de desenvolvimento baseado na IDE IntelliJ IDEA, o Android Studio.

A plataforma Android esquematiza a sua arquitetura através de camadas bem divididas, conforme mostra a Figura 2.5, um diagrama com os componentes do Sistema Operacional Android.

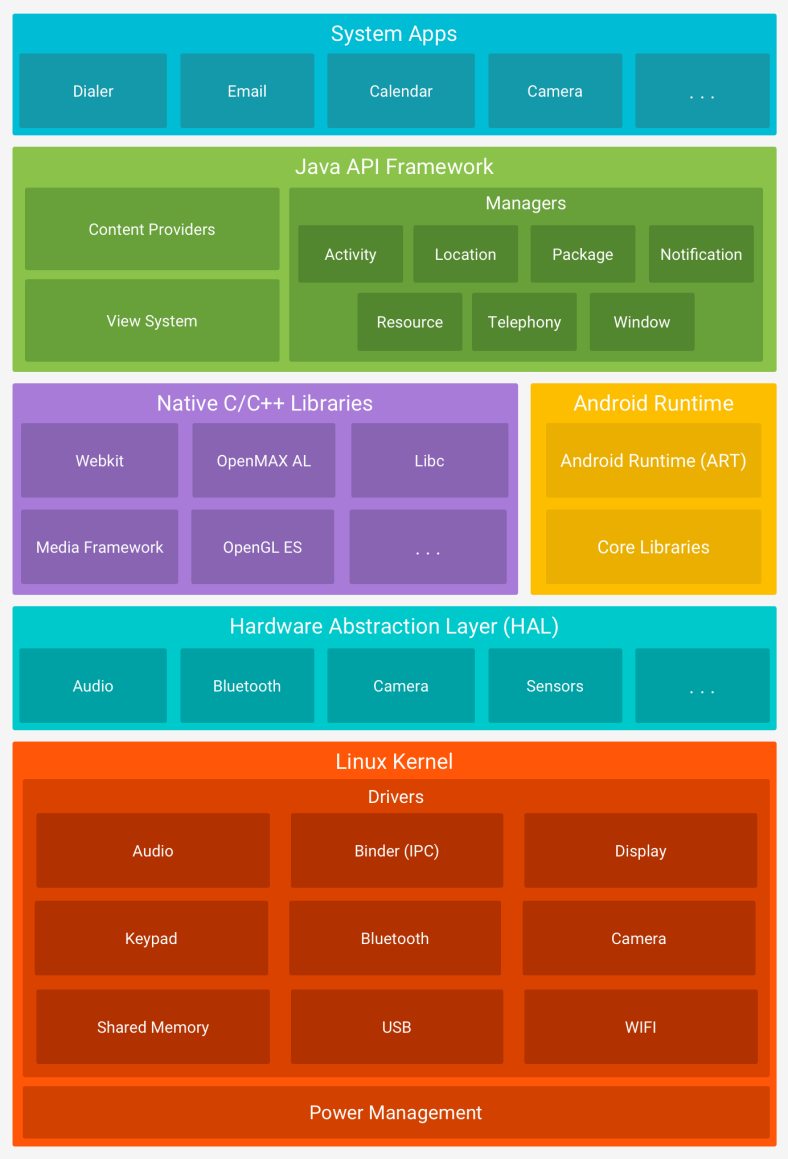


Figura 2.5: Plataforma Android

A base da plataforma Android é o *Linux Kernel*. Por exemplo, o *Android Runtime* (ART) depende do *kernel* do Linux para funcionalidades subjacentes, tais como *threading* e gerenciamento de memória de baixo nível. O uso de um kernel Linux permite que o Android aproveite os principais recursos de segurança e permita que os fabricantes de dispositivos desenvolvam *drivers* de hardware para um *kernel* bem conhecido (ANDROID, 2016b).

A camada de abstração de hardware (*Hardware Abstraction Layer* (HAL)) fornece interfaces padrão que expõem os recursos de hardware do dispositivo em alto nível para API do Java. O HAL consiste em vários módulos de biblioteca, que implementam interfaces para cada tipo específico de componente de hardware, como a câmera ou o bluetooth (ANDROID, 2016b).

Para dispositivos que rodam a versão 5.0 (API 21) ou superior do Android, cada app é executada em um processo próprio e com sua própria instância do ART. O ART é escrito para executar múltiplas máquinas virtuais em dispositivos com pouca memória, através da execução de arquivos DEX, um formato de *bytecode* projetado especialmente para o Android, otimizado para utilizar pouca memória (ANDROID, 2016b).

O Android inclui um conjunto de bibliotecas nativas escritas nas linguagens C e C++, que são usadas por diversos componentes e serviços básicos do sistema Android, como o ART e o HAL. A plataforma Android fornece APIs do *framework* Java para expor a funcionalidade de algumas dessas bibliotecas nativas a aplicativos (ANDROID, 2016b).

O Framework Java API é uma camada que oferece aos desenvolvedores, um conjunto completo de recursos do SO Android programadas na linguagem Java, permitindo a criação de aplicações ricas e inovadoras. Os desenvolvedores estão livres para aproveitar o hardware do dispositivo, as informações de localização de acesso, execução de serviços em *background*, definir alarmes, notificações para adicionar a barra de status, e outros, tudo isso de forma simplificada e prezando pela reutilização de componentes e serviços de sistemas modulares e principais. A seguir são descritos esses sistemas (ANDROID, 2016b):

- Um sistema de visualização (*View System*) rico e extensivo útil para programar a interface de usuário (UI) de um aplicativo, com listas, grades, caixas de texto, botões;
- Um gerenciador de recursos (*Resource Manager*), fornecendo acesso a recursos sem código como strings localizadas, gráficos e arquivos de layout;
- Um gerenciador de notificação (*Notification Manager*) que permite que todos os aplicativos exibam alertas personalizados na barra de status;
- Um gerenciador de atividade (*Activity Manager*) que gerencia o ciclo de vida dos aplicativos e fornece uma pilha de navegação inversa;
- Provedores de conteúdo (*Content Providers*) que permite que aplicativos acessem dados de outros aplicativos, como o aplicativo Contatos, ou compartilhem os próprios dados.

2.2.3 Firebase

Firebase[1] é uma plataforma móvel, que pertence a Google[2]. Esta foi adquirida em 2014, com a finalidade de ajudar aos desenvolvedores a criar aplicativos em tempo real para iOS,

[1] https://firebase.google.com/
[2] https://www.google.com/intl/pt-BR/about/

Android e a web podendo armazenar e sincronizar dados instantaneamente (LARDINOIS, 2014). Firebase é composto de recursos complementares que o desenvolvedor pode misturar e combinar para atender às suas necessidades (Figura 2.6).

Implementar aplicações para a plataforma Firebase é rápido e fácil. Com APIs intuitivas empacotadas em um único SDK, dessa forma o desenvolvedor não precisa perder tempo construindo infra-estrutura complexa para suas aplicações.

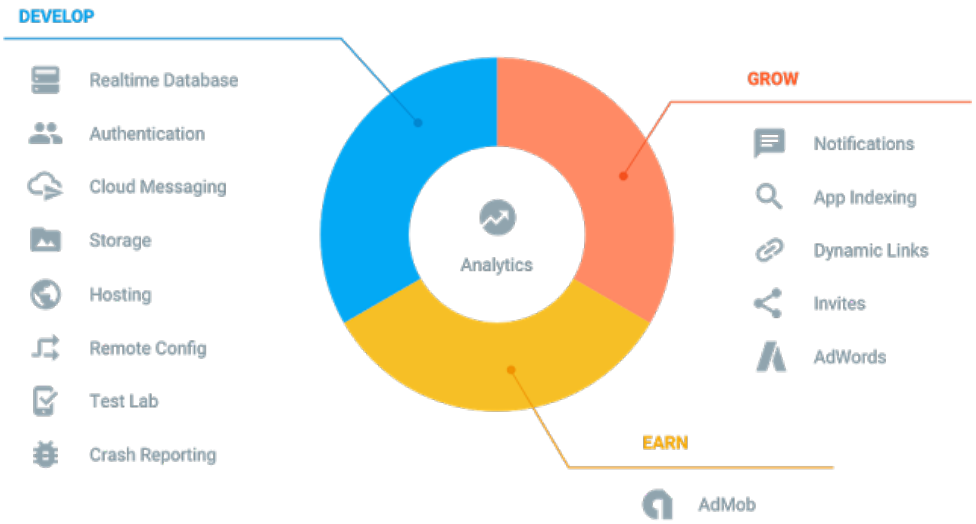


Figura 2.6: Firebase

Ao utilizar o firebase o usuário deve atentar-se a desenvolver sua aplicação e deixar as outras operações por conta do firebase. Para isso, o usuário do serviço tem ao seu alcance algumas *features*, são elas (FIREBASE, 2017):

- *Cloud Messaging* - Entregar e receber mensagens em todas as plataformas de forma confiável
- *Authentication* - Reduza o atrito com a autenticação robusta
- *Realtime Database* - Armazenar e sincronizar dados de aplicações em tempo real
- *Storage* - Armazene ficheiros com facilidade
- *Hosting* - Forneça conteúdo da Web mais rápido
- *Test Lab* - Teste no laboratório, não nos seus usuários
- *Crash Reporting* - Mantenha a sua aplicação estável

No projeto da ferramenta desenvolvida dentre as *features* do firebase foram utilizadas apenas 3: a *Authentication*, que permite o desenvolvedor criar sua aplicação com acesso restrito as informações por ele capturadas na aplicação, como também permite que outros usuários, também autenticados, possam dividir esses dados; O *Realtime Database* que permite que as informações adquiridas pela aplicação estejam disponíveis o mais rápido possível; e o *Storage*

que permite armazenar dados, como imagens que venham a ser capturadas pela câmera, por exemplo.

2.3 *Model Driven Architecture* (MDA) e Model Driven Development (MDD)

O MDA é um padrão aberto regido pelo OMG (Object Management Group) e surgiu como uma evolução natural do uso de modelos de software no que diz respeito a usar os modelos não apenas como ferramentas de comunicação e documentação, mas sim como elementos ativos e participantes do processo de desenvolvimento como um todo, ou seja, o MDA é um modelo de software, cujo diferencial está no fato do desenvolvimento ser baseado nas atividade de modelagem. Para o uso do MDA é necessário que se crie três modelos (KLEPPE; WARMER; BAST, 2003):

- O *Computacional Independent Model* (CIM), ou Modelo Independente de Computação;
- O *Platform Independent Model* (PIM), ou Modelo Independente de Plataforma;
- O *Platform Specific Model* (PSM), ou Modelo para Plataforma Específica.

O CIM é um modelo que descreve o domínio de uma aplicação e não tem nenhum detalhe sobre a estrutura e processamento do sistema, neste são definidos o contexto do negócio e os seus requisitos. Já o PIM é o resultado da transformação do CIM, porém no PIM é que deve ser modelado em relação entre as propriedades de uma entidade e as interações entre entidades, além dos serviços e interfaces para o negócio. Tais interações e relacionamentos são descritos no modelo CIM desde que ele não descreva detalhes sobre a estrutura e processamentos do sistema. Temos também a transformação do PIM em PSM. O PSM é um modelo associado a um sistema específico em termos de uma tecnologia de implementação específica. Após essa transformação tem início a última transformação que através de uma ferramenta o PSM é transformado em Code que é uma especificação do sistema em código fonte.

O processo do desenvolvimento do MDA é eficiente, porém muito rígido. Uma vez que as funcionalidades e restrições do projeto serão baseadas em UML (Unified Modeling Language, Linguagem de Modelagem Unificada) que é o padrão de modelagem. Dessa forma, uma vez definido a modelo UML no inicio do projeto, esse será usado para todo o processo de desenvolvimento. O ciclo de vida do MDA (Figura 2.7).

Como foi falado, o MDA é muito rígido e uma das suas restrições é a utilização da UML como padrão de modelagem. Já o *Model Driven Development* (MDD) é um paradigma de desenvolvimento que usa modelos, não necessariamente UML, como o artefato primário do processo de desenvolvimento. Geralmente sua implementação é (semi) automática e gerada a partir de metamodelos. Com o MDD é possível seguir seu paradigma abordando uma linguagem

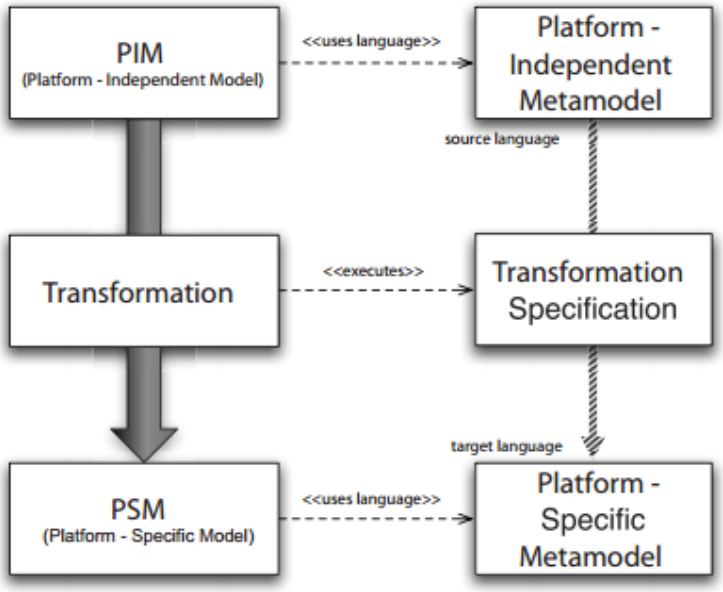


Figura 2.7: Ciclo de Vida do MDA

de modelagem de propósito geral (*General-Purpose Languages*, GPLs), ou adotando a utilização de uma linguagem de propósito específico (*Domain Specific Language*, DSL).

As Linguagens de modelagem de propósito geral (GPMLs, GMLs ou GPLs) representam ferramentas que podem ser aplicadas a qualquer setor ou domínio para fins de modelagem. O exemplo típico para este tipo de linguagem é o conjunto de linguagens UML, ou idiomas como redes de Petri ou máquinas de estado (BRAMBILLA; CABOT; WIMMER, 2012).

As DSLs são linguagens projetados especificamente para um determinado domínio, contexto ou empresa para facilitar a tarefa das pessoas que precisam descrever coisas nesse domínio. Se a linguagem for direcionado à modelagem, ele também pode ser chamado de linguagem de modelagem domínio específico (*Domain Specific Modeling Language*, DSML) BRAMBILLA; CABOT; WIMMER (2012).

Porém, conforme BRAMBILLA; CABOT; WIMMER (2012) criar DSL's que apenas reinventam a roda, uma e outra vez, deve ser desencorajado a todo o custo. Se seu DSL se assemelha a UML demais, talvez deva ser considerado usar apenas (um subconjunto da) UML e evitar inventar novas notações "quase-UML"e considerá-las novas DSLs. Normalmente, se estiver lidando com sistemas orientados a objetos, componentes ou processos, poderá reutilizar a UML devido ao suporte de modelagem já existente.

Nesse sentido, explorando as técnicas de metamodelagem ou os recursos de extensibilidade fornecidos dentro da própria linguagem UML, a ferramenta desenvolvida, utiliza-se do que a OMG denota como, UML Extensions. Isso é possível através da criação de perfis UML (UML *Profiles*) (BRAMBILLA; CABOT; WIMMER, 2012).

Em GIACHETTI; MARÍN; PASTOR (2009) apresenta que a abordagem MDD dentro da UML é representada automaticamente como UML *Profiles*. Assim o projeto da ferramenta,

se utiliza um pefil UML, então a abordagem utilizada é a MDD. Para a criação do perfil UML é necessário seguir 3 etapas, conforme a Figura 2.8):

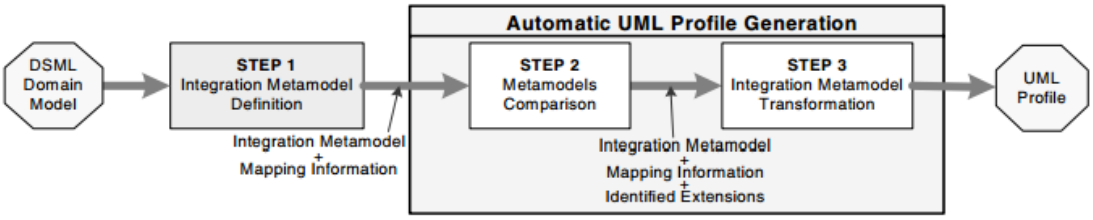


Figura 2.8: UML Profiles

- Etapa 1: Definição do Metamodelo de Integração a partir do metamodelo DSML tendo em conta o Metamodelo UML definido na Superestrutura UML.
- Etapa 2: Comparação entre o Metamodelo de Integração e o Metamodelo UML. Esta comparação identifica as extensões que devem ser definidas na UML usando as equivalências identificadas no Passo 1.
- Etapa 3: Transformação do Metamodelo de Integração de acordo com um conjunto de regras de transformação para obter o perfil UML final.

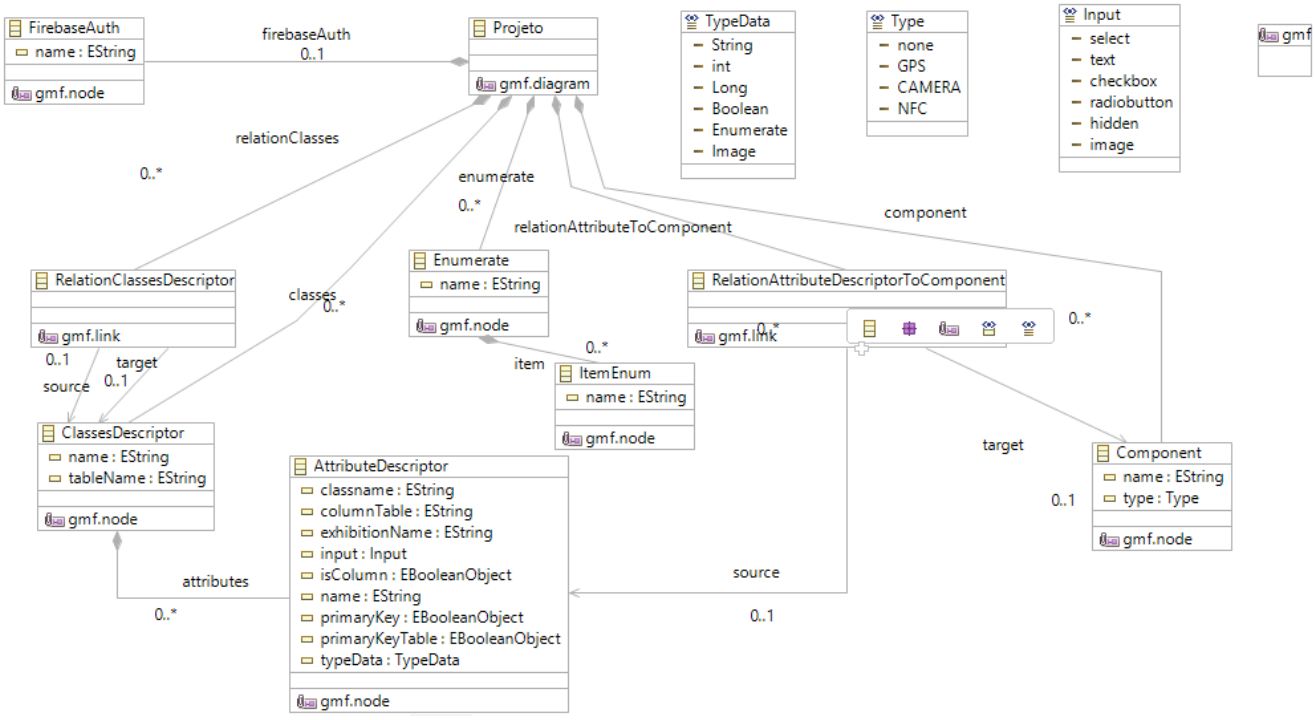


Figura 2.9: Metamodelo de Integração

De acordo com GIACHETTI; MARÍN; PASTOR (2009) a geração de perfis UML pode ser automatizada por meio de regras de transformação que são implementadas sobre a definição

XMI do Metamodelo de Integração. A definição EMOF também permite a implementação de editores de modelos específicos com ferramentas como o Eclipse GMF. Dessa forma, a definição do metamodelo de integração (etapa 1) é ideal, uma vez que o metamodelo foi definido e validado com uso do Eclipse GMF.

A Figura 2.9 apresenta o Metamodelo de Integração criado através do GMF, com as relações existentes entre as entidades, tipos de dados e cardinalidades. Pela figura é possível constatar, por exemplo, que o projeto criado através deste metamodelo deve possuir *ClassesDescriptor*, a qual pode deve possuir *AttributesDescriptor*. Outra informação, que para o *Component* necessita ser informado o tipo de sensor (GPS, Câmera, NFC), esses escolhidos em virtude do escopo dessa dissertação, sobre a utilização do *smartphone* como ferramenta de obtenção de dados.

Na etapa 2 deve ser feita a comparação entre Metamodelo de Integração (I.M.) e o elemento UML. Conforme GIACHETTI; MARÍN; PASTOR (2009) a coluna diferença deve mostrar quais são as diferenças indicando (quando necessário) os valores para o elemento Metamodelo de Integração (I.M.) e o elemento UML (UML) (Tabela 2.1).

Tabela 2.1: Comparação entre Integration Metamodel e UML Metamodelo

Integration Metamodel	**Diferença**
ClassDescriptor.newAttr1	Different type: I.M. = AtributeDescriptor; UML = integer
Component	Different type: I.M. = ClasseDescriptor; UML = Interface Component
RelationClassesDescriptorToComponent.target	Different upper bound: I.M. = 2; UML = *
RelationClassesDescriptorToComponent.source	Different upper bound: I.M. = 2; UML = *
RelationClassesDescriptor.type	Different type: I.M. = ClassDescriptor; UML = Type

A etapa 3 foi realizada a transformação do Metamodelo de Integração para perfil UML final de acordo com (GIACHETTI; MARÍN; PASTOR, 2009). Para a transformação ser realizada, devem ser verificadas algumas regras, essas são definidas considerando que os novos elementos e as diferenças entre elementos equivalentes identificados durante a comparação do metamodelo devem ser representados no perfil UML gerado. Além disso, essas regras levam em consideração a geração automática das restrições necessárias para assegurar a correta aplicação das extensões geradas. Os elementos avaliados são classes, propriedades(relação e atributos), enumerações, generalizações, regras OCL e tipos de dados. A Figura A.1 que representa a transformação encontra-se no Apêndice A.

2.4 *Generative Programming*

A Programação Generativa (*Generative Programming*) baseia-se na engenharia de sistemas e famílias (também referida como engenharia de linha de produtos) e coloca seu foco na maximização da automação do desenvolvimento de aplicativos, dada uma especificação do sistema, um sistema concreto é gerado com base em um conjunto de componentes reutilizáveis (CZARNECKI et al., 2000).

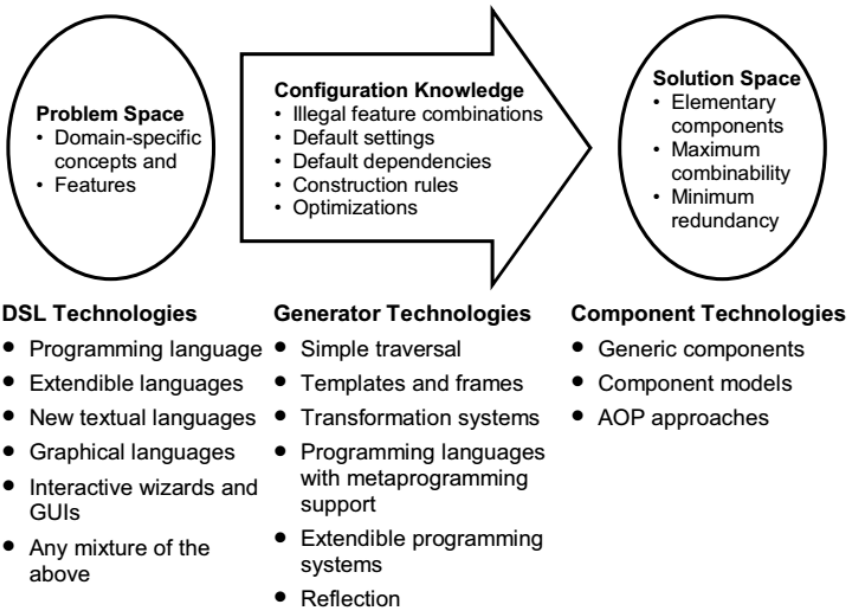


Figura 2.10: Modelo de domínio generativo e projeção de tecnologias (CZARNECKI et al., 2000)

Cada um dos elementos de um modelo de domínio generativo pode ser implementado usando Tecnologias diferentes, que dão origem a diferentes projeções tecnológicas, são elas:

- **DSL** podem ser implementadas usando recursos específicos da linguagem de programação (Como na metaprogramação de modelo em C++), linguagens extensíveis (Que permite extensões de sintaxe específicas do domínio, como OpenC++, OpenJava, programação baseada em palavras-chave Refill e Jasper, linguagens gráficas (por exemplo, UML), ou GUI e assistentes interativos.

- Os **geradores** podem ser implementados usando abordagens baseadas em templates, capacidades de metaprogramação embutidas de um idioma (por exemplo, metaprogramação de modelo em C++), sistemas de transformação.

- Os **componentes** podem ser implementados usando, por exemplo, componentes genéricos (como no STL (*Standard Template Library*)), modelos de componentes (por exemplo, JavaBeans, EJB, ActiveX ou CORBA) ou abordagens de programação orientadas a aspectos (por exemplo, AspectS).

2.4.1 DSL

As DSLs são linguagens projetados especificamente para um determinado domínio, contexto ou empresa para facilitar a tarefa das pessoas que precisam descrever coisas nesse domínio. Se a linguagem for direcionado à modelagem, ele também pode ser chamado de linguagem de modelagem domínio específico (*Domain Specific Modeling Language*, DSML).

As linguagens de modelagem são definidas através de três conceitos básicos (BRAMBILLA; CABOT; WIMMER, 2012):

- **Sintaxe Abstrata:** Descreve a estrutura da linguagem e a forma como as diferentes primitivas podem ser combinadas, independentemente de qualquer representação específica ou codificação.
- **Sintaxe Concreta:** Descreve representações específicas da linguagem de modelagem, cobrindo codificação e/ou problemas de aparência visual. A sintaxe concreta pode ser tanto textual quanto gráfica. Se a sintaxe é visual, o resultado da atividade de modelação consiste em um ou mais diagramas.
- **Semântica:** Descreve o significado dos elementos definidos na linguagem e o significado das diferentes formas de combiná-los.

Na Figura 2.11 é mostrado como os três conceitos se relacionam, conforme (BRAMBILLA; CABOT; WIMMER, 2012).

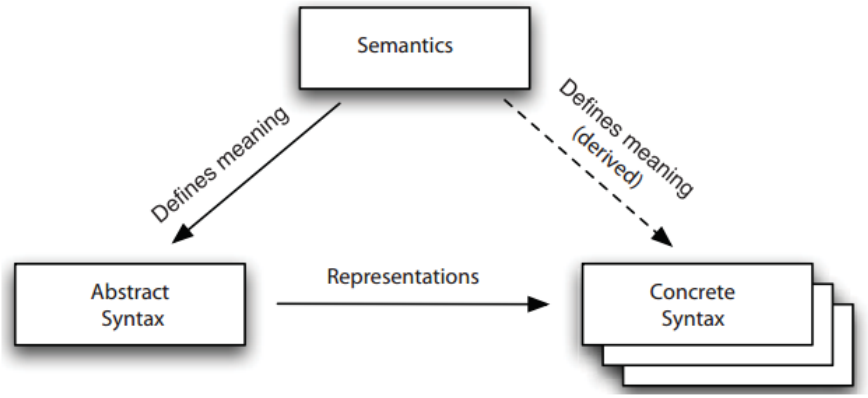


Figura 2.11: Os conceitos principais das linguagens de modelagem e suas relações

2.4.2 Geradores

Geradores de Código são basicamente programas que geram outros programas. Os geradores podem ser definidos como ferramentas tanto para formatar códigos simples quanto para gerar aplicações complexas a partir de modelos abstratos (*templates*). São muito utilizados para agilizar o processo de desenvolvimento, pois aumentam a produtividade e diminuem o tempo gasto na codificação da aplicação e, consequentemente, o custo final (AMBLER, 2009).

Alguns dos benefícios da utilização dos geradores de código são: *Qualidade*: grande volume de código escrito manualmente tendem a ter qualidade inconsistente, por que engenheiros encontram novas ou melhores abordagens para trabalhar. A geração de código cria um código consistente instantaneamente, dessa forma quando os templates são mudados, essas mudanças

refletem-se a todo o código. *Consistência:* o código construído com o gerador de código é consistente, quanto a API e aos nomes das variáveis, tornando o código fácil de entender e usar. *Ponto único de conhecimento:* uma mudança no arquivo de esquema é refletida em todo o resto da aplicação, não necessitando modificações manuais em cada arquivo gerado. *Arquitetura consistente:* O gerador de código usado para um projeto é a realização das decisões de arquitetura feitas pelos gerentes no ciclo de desenvolvimento, dessa forma encoraja os programadores a trabalharem com essa arquitetura, padronizando futuras codificações. *Desenvolvimento ágil:* a característica chave dos geradores de código é sua maleabilidade. Em nível de negócio, quer dizer que o software será mais fácil de ser mudado e melhorado ao longo do desenvolvimento (HERRINGTON, 2003).

A partir desses benefícios percebe-se a importância da utilização dos geradores de código em relação ao desenvolvimento e codificação de forma manual.

Geradores de Código baseados em *templates*

O processo que é conhecido como *Template Based Generation* (TBG) (Figura 2.12) é composto por três componentes: modelos, *templates* e o avaliador. O modelo é um artefato que descreve a aplicação em um alto nível de abstração. O *template* consiste em um código de linguagem de objeto com marcadores de posição, as *tags*, as quais contêm expressões para obter os dados do modelo. O avaliador tem a função de ler o conteúdo das *tags* do *template* e substituir os seus valores pelos do modelo, e enfim, gerar o código da aplicação (ARNOLDUS; BIJPOST; BRAND, 2007).

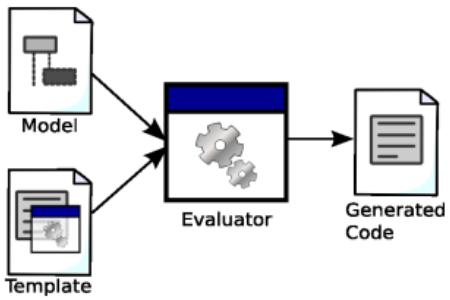


Figura 2.12: Template Based Generation
(ARNOLDUS; BIJPOST; BRAND, 2007)

Dentre as ferramentas que utilizam o TBG, temos o Xdoclet[3], que é uma ferramenta de código aberto para geração de código. Ela é baseada no *Attribute-Oriented Programming* para Java, isto significa que o programador pode adicionar mais significância ao código por poder adicionar meta-dados (atributos) no código Java, isso é feito a partir de *tags* especiais JavaDoc (XDOCLET, 2005).

[3]http://xdoclet.sourceforge.net/xdoclet/index.html

Na Figura 2.13, pode ser visto um exemplo de template Xdoclet, o qual possui a característica que suas *tags* sempre iniciarem com **XDt**. A *tag* **XDtClass:className** denota o nome da classe em notação de arquivo Java, *Annotations*.

Outra ferramenta que se destaca é o Apache Velocity Engine[4] da (APACHE, 2011) que é um sistema livre e código aberto para geração de *templates*. Ele permite a utilização de uma linguagem de *templates*, o VTL (*Velocity Template Language*), para referenciar objetos definidos em código Java.

```
public class <XDtClass:classOf><XDtClass:className/>Impl</XDtClass:classOf>
            extends <XDtClass:classOf><XDtClass:className/></XDtClass:classOf> {
  public static String[ ][ ] argumentNames = new String[ ][ ] {
  <XDtField:forAllFields>
    <XDtField:ifHasFieldTag tagName="clarguments.argname">
      {
        "<XDtField:fieldName/>",
        "<XDtField:fieldTagValue tagName="clarguments.argname" paramName="longname"/>",
        "<XDtField:fieldTagValue tagName="clarguments.argname" paramName="shortname"/>",
        "<XDtField:fieldTagValue tagName="clarguments.argname" paramName="shortdesc"/>"
      },
    </XDtField:ifHasFieldTag>
  </XDtField:forAllFields>
  };
  ... ...
```

Figura 2.13: Template Xdoclet

No Velocity as *tags* são definidas através de variáveis, que necessitam iniciar com o caractere especial **$** (cifrão), por exemplo a variável **$class.Name**, denota o nome que a classe irá receber quando iniciar o processo de geração (Figura 2.14).

```
1  ## class.vm
2  package modelo;
3  import java.util.*;
4  import java.io.Serializable;
5
6   /**@author Willamys Araújo
7   **Generate for Implementor**/
8
9  public class $utility.firstToUpperCase($class.Name)VO implements Serializable{
10
11 #foreach($atributoCorrente in $class.Attributes)
12   public $atributoCorrente.Type $atributoCorrente.Name;
13 #end
14 }
```

Figura 2.14: Template Velocity

O *Epsilon Generation Language* (EGL)[5] é uma linguagem *Model-to-Text Transformation* (M2T) baseada em TBG para gerar código, documentação e outros artefatos a partir de modelos. Este faz parte da plataforma do Eclipse Epsilon, a qual detém várias linguagens e ferramentas

[4] http://velocity.apache.org/
[5] http://www.eclipse.org/epsilon/doc/egl/

para geração de código. O EGL pode ser usado para transformar modelos em vários tipos de artefatos textuais, incluindo código executável (por exemplo, Java), relatórios (por exemplo, em HTML), imagens (por exemplo, usando DOT), especificações formais (por exemplo, notação Z) (KOLOVOS et al., 2016).

```
1  package modelo;
2  import java.util.*;
3  import java.io.Serializable;
4
5   /**@author Willamys Araujo
6   **Generate for Jacroid**/
7
8  public class [%=classes.name.firstToUpperCase()%]VO implements Serializable{
9
10 [%for (attributes in classes.attributes) { %]
11     public [%=attributes.typeData%] [%=attributes.name%];
12 [%}%]
13 [%for (projeto in projeto){
14     for( relationClasses in projeto.relationClasses){
15         if(relationClasses.source.name.equals(classes.name)){%]
16     public String key[%=relationClasses.target.name%];
17 [%}}}%]
```

Figura 2.15: Template Epsilon Generation Language

A sintaxe no EGL é similar ao PHP. Todo o código dinâmico, ou seja, que deve ser interpretado, rigorosamente deve estar entre as *tags* **[% %]** (Figura 2.15).

As três linguagens apresentadas fazem referência a objetos Java em seu *templates*, porém a escolha do EGL, foi devido a implementação da ferramenta está atrelada a utilização do Eclipse Epsilon.

EuGENia

A plataforma Epsilon[6] compõe uma família de linguagens e ferramentas para geração de código, transformação de modelo para modelo, validação de modelos, comparação, migração e refatoração que funcionam com o *Eclipse Modeling Framework* (EMF)[7] e outros tipos de modelos. As linguagens da plataforma são: 1) Epsilon Object Language (EOL), 2) Epsilon Transformation Language (ETL), 3) Epsilon Validation Language (EVL), 4)Epsilon Generation Language (EGL), 5) Epsilon Wizard Language (EWL) e 6) Epsilon Comparison Language (ECL). No presente trabalho é utilizado o EGL que já foi comentado no seção anterior.

O EMF é um padrão comum para modelos de dados, e que muitas tecnologias e *frameworks* se baseiam. O EMF (Core) é formado por três partes fundamentais (ECLIPSE, 2006):

- o **EMF**, que representa o núcleo do EMF, o qual é composto pelo metamodelo ecore (modelo usado para representar modelos em EMF);

[6]http://www.eclipse.org/epsilon/
[7]http://www.eclipse.org/modeling/emf/

- o **EMF.Edit** que inclui classes genéricas reutilizáveis para editores de construção para modelos EMF;
- **EMF.Codegen** que se utiliza do EMF.Edit e do EMF(ecore) definido para gerar um editor de trabalho completo para o modelo desejado.

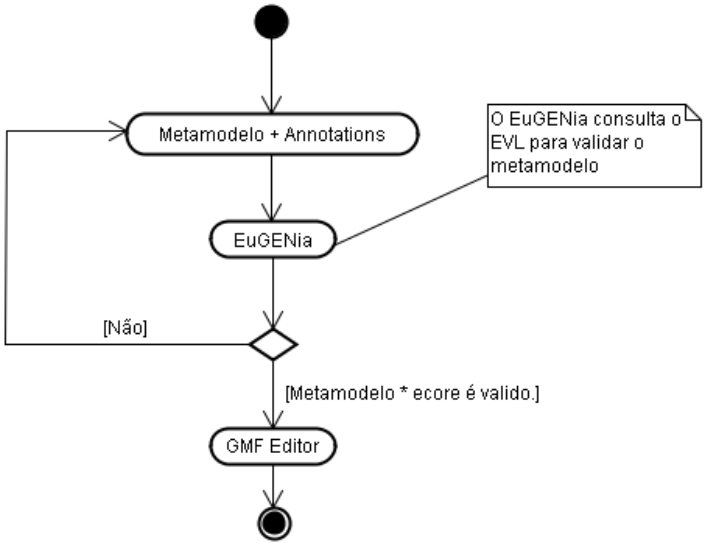


Figura 2.16: Gerar GMF Editor

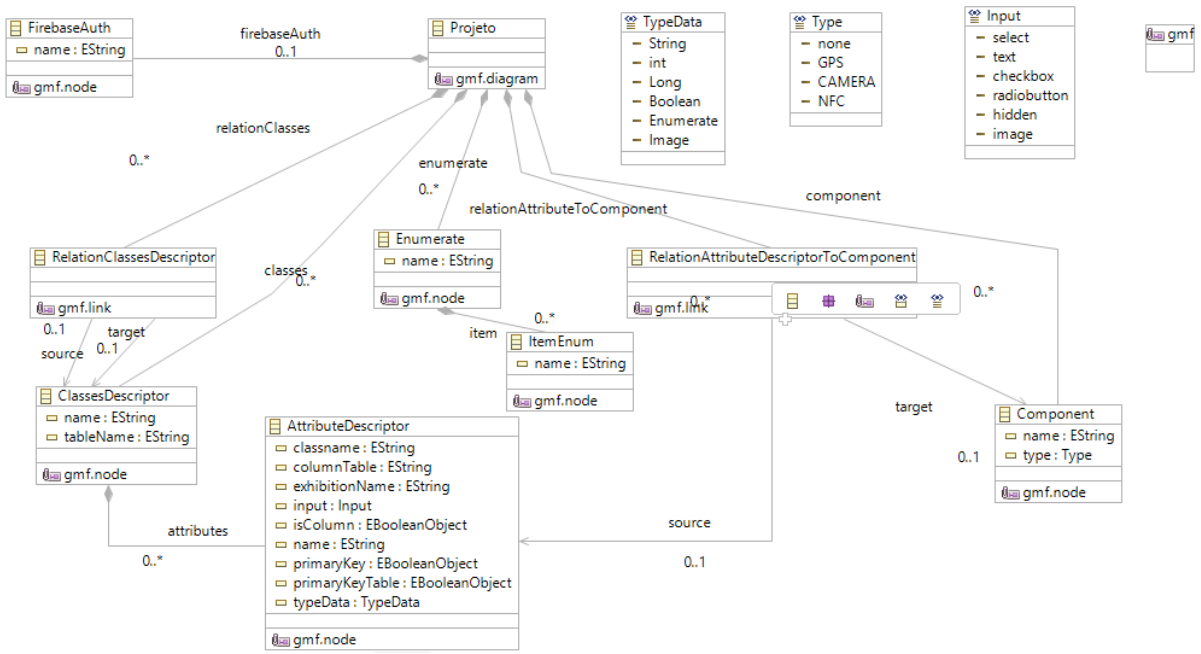


Figura 2.17: Metamodelo Ecore

Entre as diversas ferramentas disponíveis na plataforma Epsilon, a que foi definida e usada no presente trabalho foi a EuGENia[8]. Esta ferramenta tem a capacidade de gerar

[8]http://www.eclipse.org/epsilon/doc/eugenia/

automaticamente os modelos necessários para implementar um editor GMF a partir de um único metamodelo ecore (Figura 2.17). O EuGENia fornece anotações de alto nível que abstraem a complexidade do GMF e reduz a barreira de entrada para a criação do editor GMF. Para a criação de um editor GMF os passos realizados estão descritos na Figura 2.16).

Uma vez o editor GMF criado, é possível ser realizada a diagramação dos componentes de um modelo válido para o metamodelo usado como referência pelo editor. O editor será apresentado no próximo capitulo, em que será abordado o desenvolvimento da ferramenta.

2.4.3 Components

A abordagem de componentes é uma outra alternativa para a obtenção de soluções reutilizáveis para o processo de desenvolvimento de software, buscando uma maior produtividade com menor custo (BLOIS, 2006).

Segundo (SZYPERSKI; GRUNTZ; MURER, 1998) define um componente de software como uma unidade de composição com interfaces contratualmente especificadas e com únicas dependências explícitas de contexto. Cita também que um componente de software pode ser instalado independentemente e estar sujeito à composição por outros (terceiros) componentes.

Em (GIMENES; HUZITA, 2005) são identificados alguns benefícios relacionados à reutilização de componentes, como: redução de custo e tempo de desenvolvimento; gerenciamento da complexidade; desenvolvimento paralelo de componentes que possuem serviços independentes e bem definidos; aumento da qualidade, uma vez que espera-se que estes componentes sejam cuidadosamente testados para promover a sua reutilização, e facilidade de manutenção e atualização, em função da identificação clara dos serviços prestados pelos mesmos.

Os componentes dentro de uma arquitetura, geralmente são agrupados em diferentes camadas, de acordo com o nível de abstração, e prestando serviços da camada inferior para a superior (SZYPERSKI; GRUNTZ; MURER, 1998).

Conforme (BROWN, 2000) a primeira camada compreende os componentes de negócio, que implementam um conceito ou processo de negócio autônomo. A camada intermediária possui os componentes utilitários, que prestam serviços genéricos necessários ao desenvolvimento das aplicações.A última camada envolve os componentes de infra-estrutura. Dentre os serviços providos por componentes de infra-estrutura, estão: comunicação entre componentes distribuídos, persistência, tratamento de exceções e portabilidade, Veja Figura 2.18.

Como exemplo, podemos destacar o *Enterprise Java Bean* (EJB) em que os componentes de negócio correspondem aos *Entity Beans* e os componentes de processo os *Session Beans*. Na camada de infra-estrutura, mais precisamente para persistência o EJB possui o *Java Persistence API* (JPA).

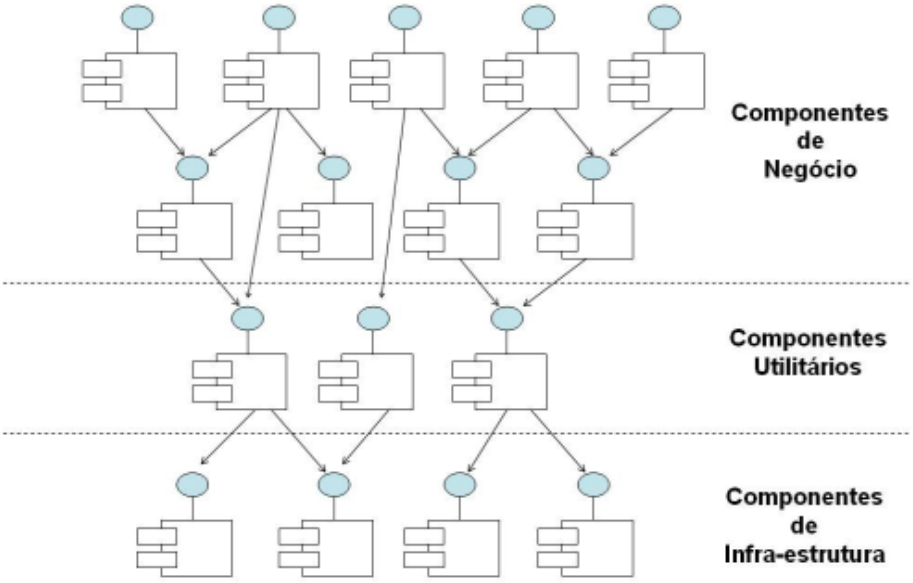


Figura 2.18: Componentes de uma Arquitetura em Camadas

2.5 Trabalhos Relacionados

Nesta subseção serão abordados os trabalhos relacionados a essa dissertação. Ao final serão apresentadas uma análise das características que a ferramenta pretende atacar em relação aos trabalhos analisados.

[1] IoTLink: An Internet of Things Prototyping Toolkit (PRAMUDIANTO et al., 2014)

Permite inexperientes desenvolvedores comporem aplicações *mashup* através de uma linguagem específica de domínio gráfica que pode ser facilmente configurados e ligados entre si para criar um aplicativo Internet das coisas. A aplicação desenvolvida é baseada na abordagem do MDD e funciona em forma de *plugin* do eclipse, implementado usando o EMF/GMF para sua construção.

Como pontos fortes podemos destacar: permitir a conexão de componentes Arduino, SOAPInput, RESTInput, OPCClient, MTQTTInput; permitir a construção de aplicações *mashup* utilizando-se do Eclipse para o seu desenvolvimento. Como pontos fracos temos: não suportar a plataforma Android, muito menos a utilização dos sensores de *smartphones*; a solução encontrada é para disponibilizar aplicações para uso somente em Arduino e Raspberry; a ferramenta ainda não conta com suporte a serviços de armazenamento em nuvem implementado, como o firebase.

[2] Integration In The Physical World In Iot Using Android Mobile Application (THIYAGARAJAN; RAVEENDRA, 2015)

O artigo apresenta um serviço IoT que utiliza sensores para coletar informações e analisar os dados para serem enviados via internet. Os dados recebidos pelo aplicativo Android e as ações correspondentes são apresentadas e o usuário pode tomar ações baseadas no evento. Uma aplicação disso é a utilização de *Smart Home Automation*, em que na sua casa tem sensores avaliando cenários que geram informações que necessitam de uma tomada de decisão. Essas informações chegam ao *smartphone* android para que o usuário decida o que deve ser feito.

Um ponto forte apresentado é a utilização de aplicações android em conjunto com serviços IoT. Porém, como pontos fracos temos: o *smartphone* é utilizado apenas como "controle remoto", ou seja, não utiliza-se dos sensores dos dispositivos móveis; as informações obtidas não são armazenadas em nenhuma base da dados, o que pode ocasionar em perda de informação; o desenvolvimento da aplicação não utiliza nenhum ambiente gráfico para o desenvolvedor implementar seu aplicativo através de modelos, por exemplo.

[3] OpenIoT - An Open Service Framework for the Internet of Things (KIM; LEE, 2014)

O OpenIoT tem como objetivo prover um serviço ou um ecossistema para aplicações e dispositivos IoT. Ele permite que desenvolvedores disponibilizem suas APIs na plataforma, a fim de auxiliar na criação de novas aplicações e uso de dispositivos. Essa plataforma é composta por cenários como Device Plataform (gerenciar o dispositivo), Planet Platform (comunidade de usuários do serviço) e Mashup Platform. O OpenIoT permite a gerencia das “coisas” através de aplicativos instalados em *smartphones*. Por exemplo, o OpenIoT disponibiliza uma API, chamada de iBike. Essa API prover os insumos necessários para o desenvolvimento de aplicações que irão usar o serviço público de aluguel de bicicletas.

Como pontos fortes do OpenIoT é possível destacar: ser um *framework* que auxilia no desenvolvimento de projetos que fazem uso de *smartphones* android para o controle dos sensores; apresenta uma ferramenta mashup. Como pontos fracos observados, segue: não se beneficia dos sensores presentes nos dispositivos móveis; foi necessário criar toda uma plataforma pra o funcionamento, uma vez que poderia usufruir de algumas já disponíveis como o firebase, parse.com; a ferramenta mashup é apenas para mostrar as informações referentes aos dispositivos conectados a plataforma; Não possui uma editor gráfico para a construção de aplicações baseada em modelos.

[4] Mobile digcovery: discovering and interacting with the world through the Internet of things (JARA et al., 2014)

Este artigo apresenta mecanismos para a descoberta de recursos globais, acesso de dispositivos para objetos inteligentes em diferentes cenários e sensores e dispositivos de usuários finais (detecção participativa). A arquitetura Digcovery oferece uma estrutura para permitir que os usuários registrem e/o incluam seus próprios sensores em uma infraestrutura e acessem e/ou descubram os recursos disponíveis através da mobile digcovery. A mobile digcovery explora a Tecnologias de reconhecimento de contexto, geolocalização e identificação disponíveis em plataformas móveis, como *smartphones* para descobrir, interagir e acessar os recursos por meio de Motor ElasticSearch.

A estrutura que o mobile digcovery proporciona a integração com o mundo real, composto de diversas espécies de objetos que podem ser localizados ou identificados com a utilização Zigbee, 6LoWPAN. E ainda proporciona a interação em que os usuários com o uso de sensores dos *smartphones* e plataformas mobile podem obter geolocalização (GPS), ler informações de códigos de barra/QR code (Camera), interação mobile-to-mobile (NFC).

Pontos fortes a serem destacados:a ferramenta apresentada possui um plataforma própria para suportar os dispositivos que controla; uso dos sensores presentes nos dispositivos móveis. Pontos fraco: não apresenta nenhuma ferramenta gráfica ou editor gráfico que auxilie o desenvolvimento de aplicações.

[5] A High-Level Modeling Language for the Efficient Design, Implementation, and Testing of Android Applications (JABER et al., 2016)

Este artigo apresenta o MoDroid uma linguagem de modelagem de alto nível para facilitar o desenvolvimento de aplicações android. A ferramenta prover a desenvolvedores android as seguintes vantagens: modelos construídos usam primitivas de alto nível que abstraem vários detalhes de implementação; Permite a definição de interfaces entre modelos para compô-los automaticamente; Uma aplicação nativa do Android pode ser gerada automaticamente juntamente com a definição de permissões necessárias; Suporta execução de testes eficiente que sobre os modelos.

A ferramenta presentada no artigo possui os seguintes pontos fortes: baseado em modelos; plataforma Android; a linguagem de alto nível utilizada faz uso de composição de arquivos de configuração abstraindo a implementação para o desenvolvedor. Os Pontos fracos: não possui serviço de armazenamento de dados na nuvem; não faz uso dos sensores dos dispositivos móveis para capturar informações; não há uma ferramenta gráfica para a composição dos modelos.

2.5.1 Análise dos trabalhos

Na Tabela 2.2 são apresentadas alguns pontos fortes e pontos fracos identificados nas abordagens analisadas e comparadas com a em desenvolvimento. A primeira coluna que tem a informação "Abordagem", utilizou a numeração atribuída aos artigos da seção 2.5, e a abordagem desenvolvida foi nomeada de "Proposta". As demais colunas são características que aplicações do domínio de pessoas como serviço necessitam.

Tabela 2.2: Características identificadas nos trabalhos relacionados

Abordagens	Editor Gráfico	Plataforma Android	Sensores Smartphone	Firebase (Storage e Realtime Database)	IoT
1	SIM	NÃO	NÃO	NÃO	SIM
2	NÃO	SIM	NÃO	NÃO	SIM
3	NÃO	SIM	NÃO	NÃO*	SIM
4	NÃO	SIM	SIM	NÃO*	SIM
5	NÃO	SIM	NÃO	NÃO	NÃO
Proposta	SIM	SIM	SIM	SIM	SIM

*Ferramenta utiliza plataforma própria

É possível verificar que nas ferramentas analisadas nem todas possuem editor gráfico para modelar a aplicação, e também não utilizam serviços de armazenamento em nuvem, a não ser, as ferramentas 3 e 4 que utilizam sua própria plataforma. Em relação a utilizar sensores dos *smartphones*, apenas a ferramenta 4 atentou-se em apresentar essa funcionalidade.

Diante dos fatos pode-se constatar que existem oportunidades a serem exploradas, e possivelmente, fornecer uma solução para atender algumas das fragilidades elencadas nos trabalhos relacionados, principalmente, em relação a abordagem contribuir com um ambiente propício para a criação de aplicações para o paradigma de pessoas como sensores, abstraindo a complexidade da programação através de uma ferramenta gráfica. Além desses pontos, foram observadas também as tendencias para o desenvolvimento para IoT elencadas por (SUNDMAEKER et al., 2010), as quais foram de grande valia para definição das *features* a serem implementadas na ferramenta de modelagem para a abordagem proposta.

2.6 Considerações Finais

Neste capítulo foi apresentada toda a fundamentação teórica que guiou este estudo. Foram elencadas as tendências, cuja a área de aplicações para IoT estão sendo guiadas, como: aplicações orientadas a usuários, colaboração *machine to human*, *easy deploy* (SUNDMAEKER et al., 2010).

Técnicas de programação generativa também foram abordadas a fim de selecionar a que mais tinha relação com a abordagem proposta neste estudo. Nesse sentido, a geração de código baseado em *templates* foi a escolhida, pois era a que mais se adequava ao paradigma de desenvolvimento orientado a modelos.

Nos trabalhos relacionados a dissertação foram extraídos os pontos fortes, que devem ser explorados e os pontos fracos que devem ser minimizados na abordagem proposta. No próximo capítulo, a abordagem proposta será apresentado, agregando as teorias e técnicas estudadas.

3

Abordagem Proposta

Este capítulo tem por finalidade apresentar a abordagem desenvolvida. Os objetivos, visão geral da mesma e a descrição das etapas serão apresentados, contendo os detalhes dos seus conjuntos de atividades.

3.1 Técnicas da Abordagem Proposta

A abordagem proposta combina algumas técnicas e procedimentos para a geração de aplicações dentro do domínio de pessoas como sensores. Dessa forma, estas serão apresentadas nas seções seguintes.

3.1.1 Desenvolvimento Orientado a Modelos

O desenvolvimento orientado a modelos, ou *Model-Driven Development* (MDD) é um paradigma de desenvolvimento que usa modelos como o artefato primário do processo de desenvolvimento a partir dos quais são gerados o código e outros artefatos de acordo com as boas práticas (YUSUF; CHESSEL; GARDNER, 2006).

Um modelo é uma descrição de um sistema a partir de uma determinada perspectiva, omitindo detalhes irrelevantes a fim de que as características do domínio alvo sejam evidenciadas. Dessa forma, o desenvolvimento incita a preocupações inerentes ao processo de modelagem, cuja intenção permeia pela elicitação dos requisitos funcionais, ao invés dos requisitos não funcionais, os quais serão abstraídos do processo.

Na abordagem proposta, o MDD é utilizado como opção por dois fatores: 1) permite atacar problemas de uma classe específica, composta por abstrações e notações apropriadas; 2) possui potencial de melhorar grandemente as práticas correntes de desenvolvimento de software, das quais destacam-se: a maior produtividade, capacidade de manutenção, reutilização de artefatos, adaptabilidade, dentre outros.

3.1.2 Transformações de Software

A Abordagem proposta utiliza um sistema transformacional para auxiliar e automatizar a geração do código da aplicação baseado em *templates*, conhecido como *Template Based Generation* (TBG).

Dentre as ferramentas que utilizam o TBG, destacam-se: o XDoclet (XDOCLET, 2005), Apache Velocity (APACHE, 2011) e EGL (KOLOVOS et al., 2016). A preferência pela EGL deve-se ao fato de que: 1) pode ser usado para transformar modelos em vários tipos de artefatos textuais, incluindo código executável (por exemplo, Java), relatórios (por exemplo, em HTML), imagens (por exemplo, usando DOT), especificações formais (por exemplo, notação Z); 2) a ferramenta desenvolvida para a abordagem associa-se a plataforma do Eclipse Epsilon, em especial o Eugenia; e 3) sua sintaxe é similar ao PHP, o que tornou mais amigável a implementação.

3.1.3 Ferramenta de Modelagem

Com as ideias do desenvolvimento orientado a modelos, novas tecnologias surgiram para facilitar esse trabalho. São os chamados *frameworks* de linguagens, que implementam diversas funções necessárias para a modelagem visual, como a representação, criação e edição de diagramas, e o processamento automático de suas informações internas. Exemplos incluem o GME (Generic Modeling Environment) e o GMF (Graphical Modeling Framework) (LUCRÉDIO, 2009).

A ferramenta de modelagem da abordagem foi guiada pela Engenharia de Domínio, a qual é composta por três fases: análise de domínio, projeto e implementação. A primeira fase, Análise de Domínio, tem a finalidade de selecionar e definir o foco do domínio e coletar informações relevantes do domínio e integrá-las em um modelo de domínio coerente. Para isso, foi utilizado o modelo de *feature*, *Feature-Oriented Domain Analysis* (FODA). Na fase de Projeto de Domínio é estabelecido uma arquitetura comum para os sistemas no domínio. Nesta fase são definidos os padrões/táticas de projetos a serem utilizados, componentes, artefatos MDD (transformações e gerações de código). Para a última fase, Implementação de Domínio, é realizado efetivamente a implementação o domínio, ou seja, implementar componentes, DSLs, transformações e geradores de código, seguindo o projeto definido na fase anterior.

Engenharia de Domínio

A atividade de desenvolvimento de software requer a utilização de métodos, processos ou uma orientação, a fim de permitir alcançar os melhores resultados no que diz respeito a qualidade e atendimento aos requisitos especificados para cada aplicação. Essas especificidades que cada software possui, muitas vezes podem ser encontradas em um conjunto de aplicações com características comuns (mesmo domínio), os quais incitam a promover a reutilização de conceitos e funcionalidades comuns. Esta forma de desenvolvimento de software para domínios

de aplicação é conhecida como Engenharia de Domínio.

De acordo com CZARNECKI (1998), a Engenharia de Domínio é a atividade de coletar, organizar e armazenar experiências passadas na construção de sistemas ou partes de sistemas em um domínio particular sob a forma de ativos reutilizáveis, bem como fornecer meios adequados para reutilizar esses ativos na construção de novos sistemas.

Em KANG et al. (1990), a Engenharia de Domínio (ED) pode ser vista como um processo abrangente que inclui a análise de domínio e a subsequente construção de componentes, métodos e ferramentas que abordam os problemas de desenvolvimento de sistemas / subsistemas através da aplicação da análise de domínio de produtos. Para isso é necessário realizar uma análise sobre o domínio a fim de extrair o escopo a ser atacado, descrever o problema abordado dentro do domínio, apresentando os objetos e seus relacionamentos, e por fim, estabelecer uma estrutura para implementação do software no domínio.

A seguir são apresentados as fases da ED.

Análise de Domínio

A fase de Análise de Domínio tem por finalidade fechar o escopo em que a aplicação irá abordar. De acordo com LUCRÉDIO (2009), a análise de domínio envolve a identificação dos principais conceitos e elementos de um domínio e a determinação de seu escopo, além de identificar as comunalidades e variabilidades.

Nesse sentido, uma das abordagens mais utilizadas para esse fim, é a **modelagem de *features***. Segundo BLOIS (2006) o modelo de características, é um elemento chave para relacionar conceitos de mais alto nível aos demais artefatos de análise e projeto, e é considerado o melhor caminho para efetuar os pontos de corte para reutilização de parte de um domínio.

O modelo de *feature* permite a definição de um modelo conceitual que na fase de projeto poderá ser refinado para o modelo de domínio. Esse processo de refinamento começa com: a identificação e descrição dos entidades; definir associações entre as entidades; determinar ordem de apresentação (mesma ordem dos atributos nas classes); Identificar os tipos de dados dos atributos; em alguns casos, identificação de informações sobre restrições de valores possíveis dos atributos.

Projeto de Domínio

O projeto do domínio é uma fase essencial da engenharia de domínio (BOSCH, 2000). Esta fase tem a incumbência de converter as *features* capturadas na análise de domínio em artefatos reutilizáveis, e que estes venham a compor uma arquitetura de software referencia para o domínio em questão.

Para atingir esse objetivo é necessário mapear o modelo conceitual já refinado para o modelo de projeto. Esse modelo de projeto consiste em definir de forma manual quais os padrões de projeto, componentes, plataformas, eventuais restrições e tecnologias a serem utilizados na

fase de implementação.

A Escolha das táticas e padrões arquiteturais seguiu a abordagem definida por (LUCRÉDIO, 2009). Em sua abordagem LUCRÉDIO (2009), apresenta duas formas de padronizar a implementação de variabilidade, uma baseada em *feature* e uma baseada em DSL. Para a implementação de variabilidade baseada em features, podem ser utilizados dois padrões baseados em práticas bem conhecidas para a escrita de geradores de código (CZARNECKI et al., 2002). Na abordagem *visitante*, cada elemento é visitada. Dessa forma, para cada elemento, um *template* correspondente é chamado para gerar a classe. Na abordagem *template*, o *template* principal analisa os elementos do modelo e decide quais *templates* serão chamados, quando serão chamados, a ordem. Estes podem ser interligados a outros para formar as classes.

Observe na Figura 3.1, que na abordagem visitante, todas as *features* e *sub-features* são visitadas e o *template* associado é selecionado para gerar a classe. Na abordagem de *template*, o *template* principal avalia as *features* e *sub-features* e seleciona os *templates* associados para criar a classe e os métodos.

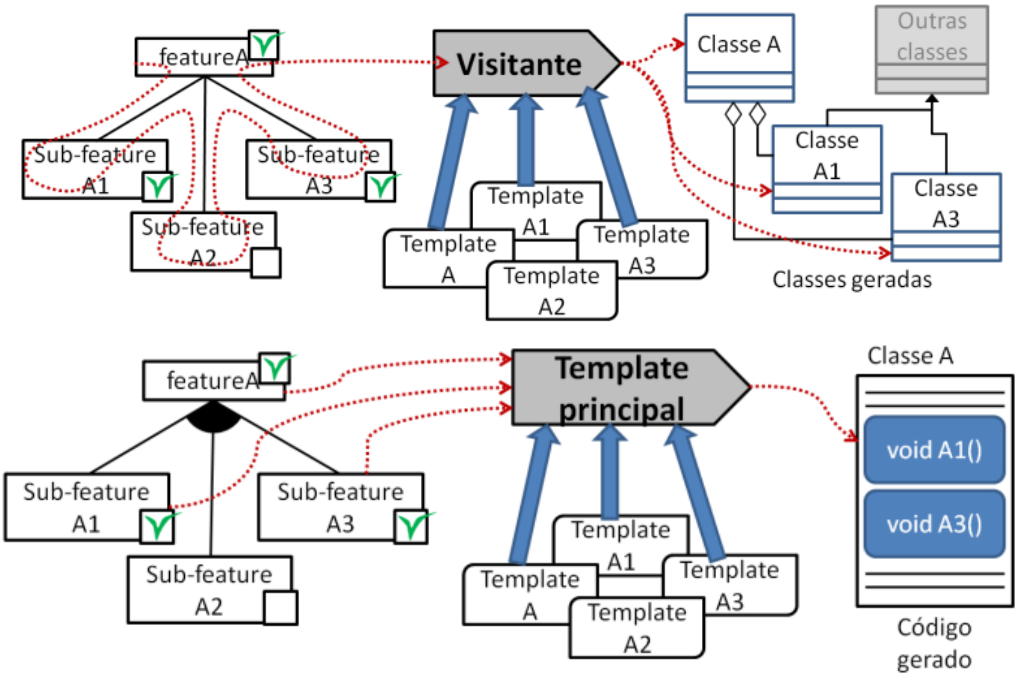


Figura 3.1: Abordagem Visitante e Abordagem Template (LUCRÉDIO, 2009)

A implementação de padrões arquiteturais para variabilidade baseada em DSLs é focada em como as ferramentas baseadas em DSL e geradores de código podem ser integrados à arquitetura e aos geradores de código baseados em *features*. O desenvolvedor especifica um programa/modelo que segue a sintaxe de uma DSL, e um gerador produz código-fonte automaticamente LUCRÉDIO (2009).

Um dos padrões propostos denomina-se *camada fina de dados*, que facilita a integração entre o gerador e a ferramenta de modelagem DSL. Os geradores de código obtém informações

por meio da ferramenta de DSL, que costuma ser desenvolvida manualmente ou através de algum *framework* de construção de linguagens como o GMF (Graphical Modeling Framework), ver seção 2.4.4. Esse padrão defende uma camada de dados separada e independente da tecnologia que implementou a ferramenta DSL, requerendo apenas informações essenciais para o gerador. Observe na Figura 3.2 que a camada fina de dados divide a carga de trabalho entre os desenvolvedores do gerador de código e o da ferramenta de modelagem, ao mesmo tempo que a mesma funciona como ponte de ligação entre ambas. A camada permite interligação fácil e desenvolvimento do trabalho em paralelo, ou seja, ganho de produtividade e baixo acoplamento.

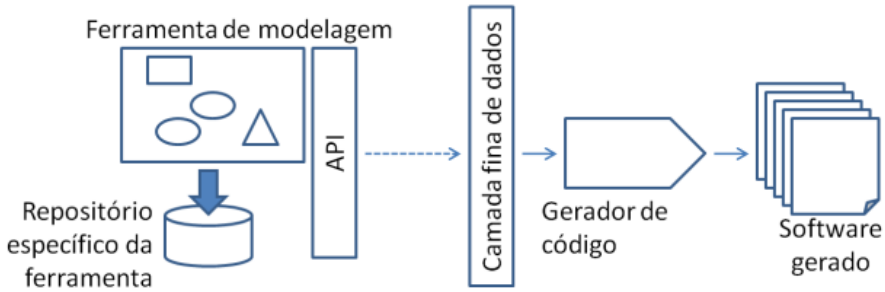


Figura 3.2: Padrão Camada Fina de Dados
(LUCRÉDIO, 2009)

Implementação de Domínio

Nesta fase é realizada a implementação manual com base nas especificações contidas no modelo de domínio resultante da fase de projeto.

Conforme LUCRÉDIO (2009), projetar e implementar um domínio descrito em termos de suas *features* e mapeá-las para a arquitetura e o código, considerando todos os relacionamentos e restrições, é considerado um grande desafio. Para tal, o MDD surge como alternativa, pois permite que o desenvolvedor da aplicação não precise se atentar as restrições, pois estas estão encapsuladas pelas transformações MDD e linguagens específicas de domínio.

3.1.4 Eclipse e Android Studio

Ambos os programas são considerados Ambientes de Desenvolvimento Integrado, ou do inglês, *Integrated Development Environment* (IDE), os quais reúnem características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. Essas ferramentas integram a abordagem na última etapa, que consiste no *deploy* ou implantação da aplicação no *smartphone*.

A ferramenta de modelagem, tratada anteriormente, já é integrada ao Eclipse Epsilon necessitando apenas que o *plugin* de Desenvolvimento da Plataforma Android esteja instalado. Já para uso do Android Studio é necessário que o código fonte gerado pela ferramenta de modelagem seja importado.

A adoção dessas duas IDEs foi impulsionada pela maior popularidade das mesmas no auxílio a implementação de sistemas na plataforma Android.

3.2 A Abordagem

3.2.1 Objetivos da Abordagem

A finalidade da abordagem é possibilitar uma melhoria de produtividade, qualidade e reúso na construção de aplicações através da transformação de modelos, com o enfoque em adquirir informações baseadas nos sensores mais comuns e usuais encontrados em *smartphones*.

A literatura apresenta algumas ferramentas que por hora, usam os *smartphones* como ferramenta de tomada de decisão e não como figura de destaque, veja discussão na seção 2.5. Outras, utilizam os *smartphones* somente para gerir as informações. Grande maioria não apresenta um ambiente gráfico no processo de desenvolvimento das aplicações.

Dessa forma, esse trabalho procura trazer uma alternativa por meio da utilização de um método de desenvolvimento de aplicativos na plataforma android com os preceitos da IoT. A ferramenta desenvolvida baseia-se na orquestração de componentes de softwares reutilizáveis de forma gráfica e geração do código automatizada por meio da transformação de modelos. Os aplicativos terão a possibilidade de utilizarem os sensores mais comuns de *smartphones*. As ideias aplicadas permitirão que os refinamentos inerentes à abordagem proposta de desenvolvimento, sejam apoiados por modelos e que os componentes criados possam ser reutilizáveis. Consequentemente, a reutilização dos componentes poderá trazer diversos ganhos em produtividade e padronização. Vale ressaltar que as aplicações já estarão aptas a serem integradas ao firebase (veja seção 2.2.3), obtendo banco de dados atualizado em tempo real e armazenamento para imagens capturadas por intermédio da câmera.

3.2.2 Desenvolvimento da ferramenta de modelagem

A abordagem proposta necessitou de uma ferramenta de modelagem que auxiliasse a sua implantação. Para tal e como foi falado na seção 3.1.3, o desenvolvimento da ferramenta foi guiada pela Engenharia de requisitos. A seguir são mostradas as etapas do desenvolvimento.

Análise de Domínio

A fase de análise de domínio utilizou o modelo de *Features* conhecido como *Feature-Oriented Domain Analysis* (FODA). Na Figura 3.3 pode ser observado os componentes necessários para modelagem e criação de um projeto no domínio de pessoas como sensores. A definição das *features* presentes nesse modelos foram obtidas pelos estudos realizados na literatura, os quais apontam para utilização de sensores presentes nos *smartphones* como Câmera e GPS e

ainda as tendências para quais a IoT tem sido guiada, conforme pode ser visto em SUNDMAEKER et al. (2010). Outro ponto levado em consideração, é que o domínio das aplicações que irão funcionar em dispositivos móveis utilizam a plataforma Android, a qual é implementada em JAVA, Linguagem de programação orientada a objetos, por isso, a alusão a utilização de classes e atributos para a sua definição. A *feature* que relaciona-se ao Firebase surgiu da necessidade de permitir que as aplicações tenham segurança, armazenamento e sincronização em tempo real e alta disponibilidade.

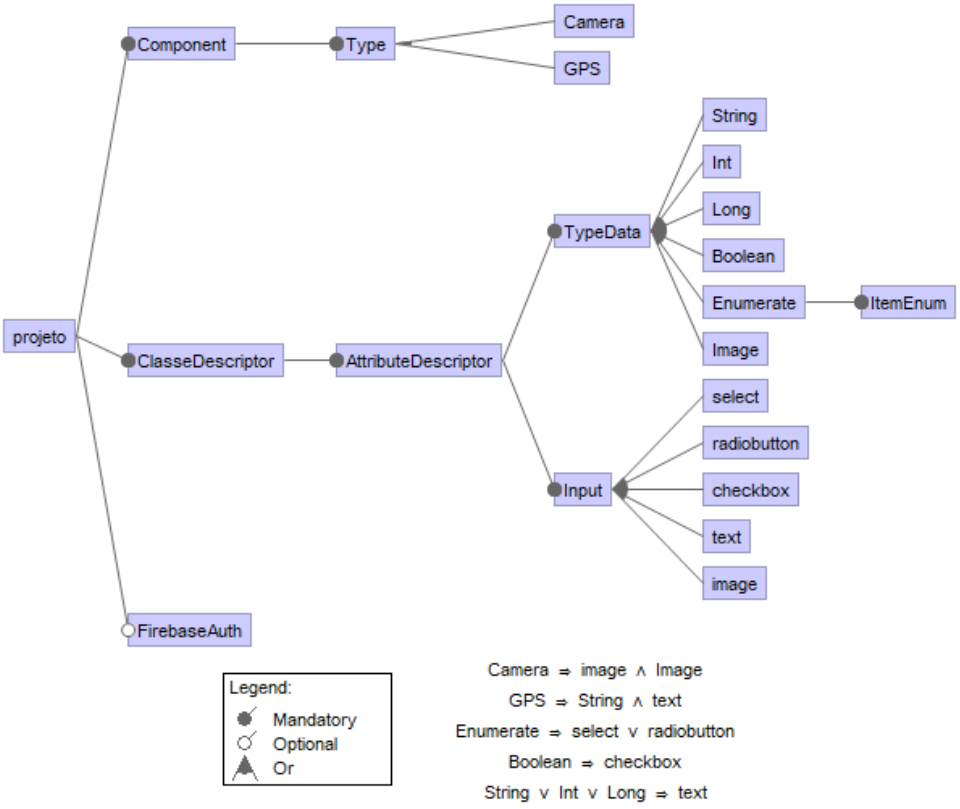


Figura 3.3: Modelo de features desenvolvido no projeto da ferramenta, segundo a notação da ferramenta FeatureIDE (THÜM et al., 2014)

Nos tópicos a seguir serão apresentadas as descrições de cada *feature* definido no modelo FODA:

- **Projeto**: Representa uma aplicação do domínio, possui necessariamente Componentes (Component) e Classes (ClassDescriptor).
- **FirebaseAuth**: Representa um componente para realizar a autenticação. A não utilização dessa *feature* implica em utilizar os serviços de armazenamento e sincronização do firebase anonimamente.
- **Component**: São os componentes que uma aplicação possa vir a ter. O Componente necessariamente deve ter um tipo (Type).
- **Type**: Define o tipo de componente (Component), que pode ser Câmera ou GPS.

- **ClassDescriptor**: São as classes da aplicação. As classes devem possuir atributos (AttributeDescritor).
- **AttributeDescriptor**: Definem os atributos da Classe (ClassDescriptor). Os atributos devem possuir dentre outras informações (nome, classe associada) o tipo de dado (TypeData) e o tipo de entrada (Input) que irá ser associada na GUI.
- **Input**: Define o tipo de entrada do atributo (AttributeDescriptor) que será associada na GUI. A entrada pode variar entre: select, checkbox, radiobutton, text, Image.
- **TypeData**: Define o tipo de dado do atributo (AttributeDescriptor). O tipo pode variar entre: int, Long, String, Boolean, Enumerate, image.
- **Restrições**: Algumas restrições (constraints) lógicas foram estabelecidas. São elas:
  - Para utilizar o Componente do tipo Camera é necessário que o tipo de dado (TypeData) seja *Image* e a entrada (Input) seja *image*.
  - Para utilizar o Componente do tipo GPS é necessário que o tipo de dado (TypeData) seja *String* e a entrada (Input) seja *text*.
  - Para utilizar o tipo de dado (TypeData) *Enumerate* a entrada (Input) deve ser um *select* ou um *radiobutton*. E para cada *Enumerate* é necessário no mínimo um *ItemEnum*.
  - Para utilizar o tipo de dados (TypeData) *Boolean* a entrada (Input) deve ser um *checkbox*.
  - Para utilizar os tipos de dado (TypeData): *String*, *int* e *Long*, a entrada (Input) deve ser um *text*.

Projeto de Domínio

Nesta fase, foi realizada a conversão das *features* capturadas na fase anterior em artefatos reutilizáveis. Para isso, técnicas e padrões arquiteturais foram selecionados.

Uma das técnicas definidas por (LUCRÉDIO, 2009) foi a abordagem de *templates*, esta, por sua vez, foi selecionada e incorporada ao motor de geração da ferramenta. Cada caraterística mapeada no modelo de *feature* corresponde a um ou a um conjunto de *templates* para montar suas classes. Além de que o projeto de aplicação gerado com a ferramenta utiliza-se do artificio de um *template* principal que se comunica com aos demais. Na Figura 3.4 temos o *template* principal que ao analisar o modelo de *feature* decide pela utilização do *template* Component e o *template* Type (GPS), os quais são combinados para gerar a classe Maps.

Um padrão que também foi utilizado nessa etapa foi o de *camada fina de dados* o qual facilita a integração entre o gerador e a ferramenta de modelagem DSL, permitindo que ambas possam fluir o seu desenvolvimento independentemente. Nesse sentido, a ferramenta teve como

framework de construção o EuGEnia, o qual fornece anotações de alto nível que abstraem a complexidade do GMF e reduz a barreira de entrada para a criação do editor GMF (Seção 2.4.4).

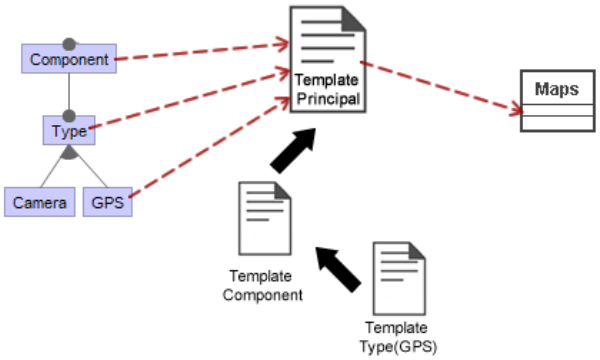


Figura 3.4: Exemplo de uso das abordagens para a característica Component

Implementação de Domínio

A primeira atividade desenvolvida na etapa de implementação foi obter o meta-modelo que permitiu a criação do Perfil UML. O metamodelo é uma das peças chaves da atividade de **Criar o Editor GMF**. O editor GMF será a ferramenta de modelagem.

Como foi utilizado a ferramenta EuGENia, o mesmo abstrai algumas etapas que poderiam requerer mais tempo para o desenvolvimento, como: criar o arquivo *.genmodel (responsável por gerar classes java de acordo com o meta-modelo; criar o arquivo *.gmfgraph (responsável por gerar os componentes gráficos); criar o arquivo *.gmftool (que permite gerar a paleta com as *features*); criar o arquivo *.gmfmap (que realiza o mapeamento do diagrama, e também um conjunto de referências. Ele referencia o meta-modelo ecore, o arquivo gmfgraph e gmftool); para que, finalmente o editor GMF seja criado. Observe na Figura 3.5 o fluxo para gerar o editor GMF.

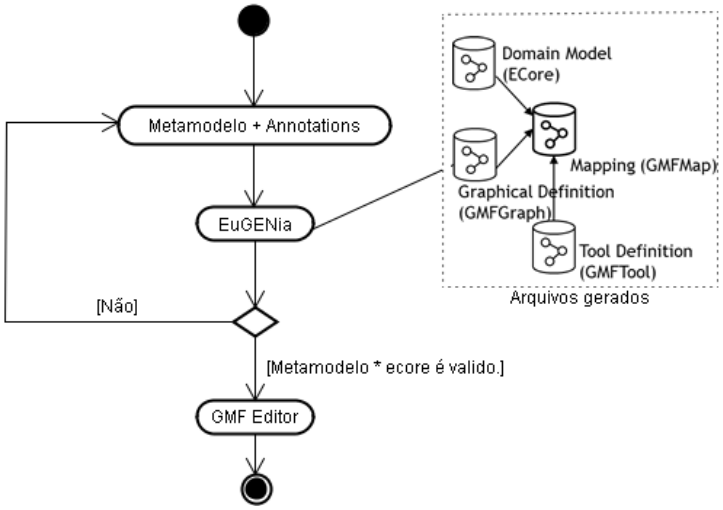


Figura 3.5: Fluxo Gerar Editor GMF

Dessa forma, a preocupação principal para a criação do editor, é projetar o metamodelo de domínio EMF. Para isso, surge a opção de utilizar a linguagem EMFatic, cuja finalidade é representar os modelos EMF Ecore numa forma textual. Na Figura 3.6, marcado com o número um (1), é ilustrado a representação da classe Enumerate que é composta por um ou mais ItemEnum.

Observe que no EMFatic, marcado com o número dois (2) na Figura 3.6, existem algumas notações (*Annotations*), como *@gmf.node, @gmf.compartment*, as quais estão realizando mapeamentos entre elementos de metamodelo e GMF. Outras vantagens trazidas pela linguagem, além dos mapeamentos são: sua sintaxe é semelhante ao Java e permite especificar restrições ao editor gráfico. A etapa final, gerar o código do editor, é realizada através de ações na IDE Eclipse a qual o Eugenia está associado. Para isso deve ser selecionado o modelo EMFfatic, e na opção Eugenia selecionar *Generate GMF Editor*.

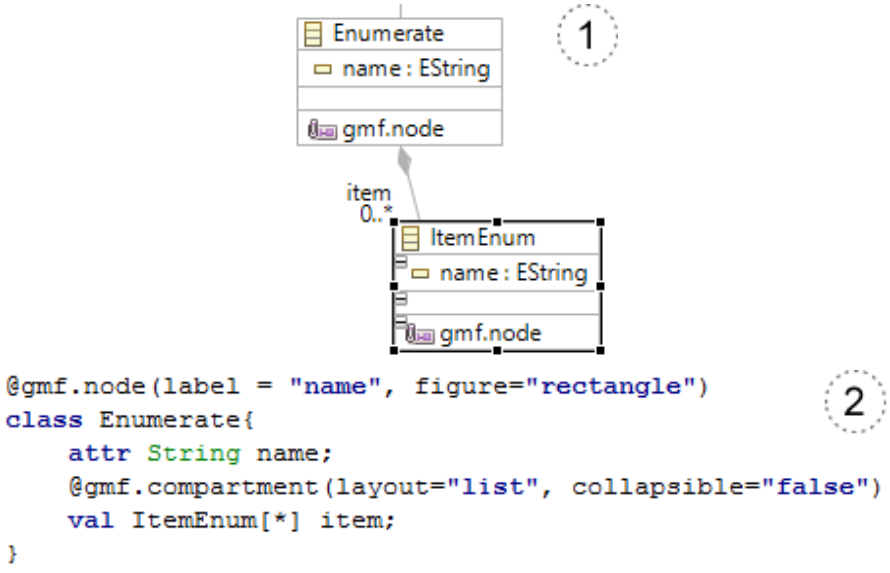


Figura 3.6: EMF Ecore e EMFatic

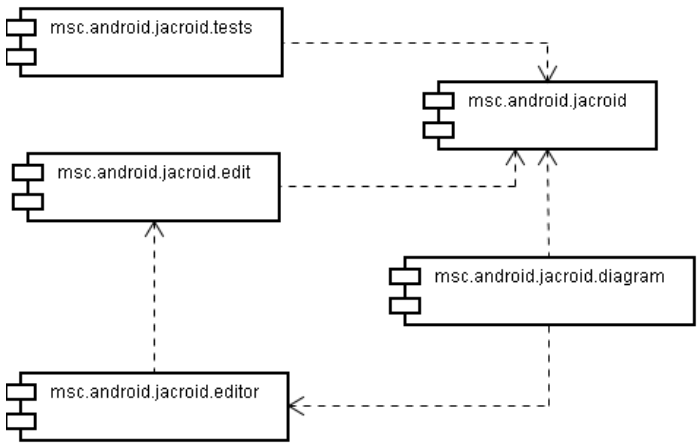


Figura 3.7: Diagrama de Componentes da ferramenta

A estrutura gerada pelo Eugenia pode ser visualizado através de um diagrama de componentes na Figura 3.7. A mesma é composta por em 5 partes fundamentais, são elas: *msc.android.jacroid*, composto pelo modelo ecore e os templates *.egl; *msc.android.jacroid.diagram*, responsável por gerenciar o modelo que o desenvolvedor, ao usar ferramenta irá desenvolve-lo, e realizar a transformação modelo para texto através do *merge* entre os templates *.egl e o modelo. A classe responsável por essa ação é a M2Action, iniciada pelo método run; *msc.android.jacroid.editor* e *msc.android.jacroid.edit*, responsáveis pelo editor gráfico da ferramenta; e *msc.android.jacroid.tests* realiza os testes para os modelos especificados na ferramenta.

Na Figura 3.8 é ilustrado o Editor GMF. Com o propósito de permitir um melhor entendimento, foram enumeradas as partes que o compõem.

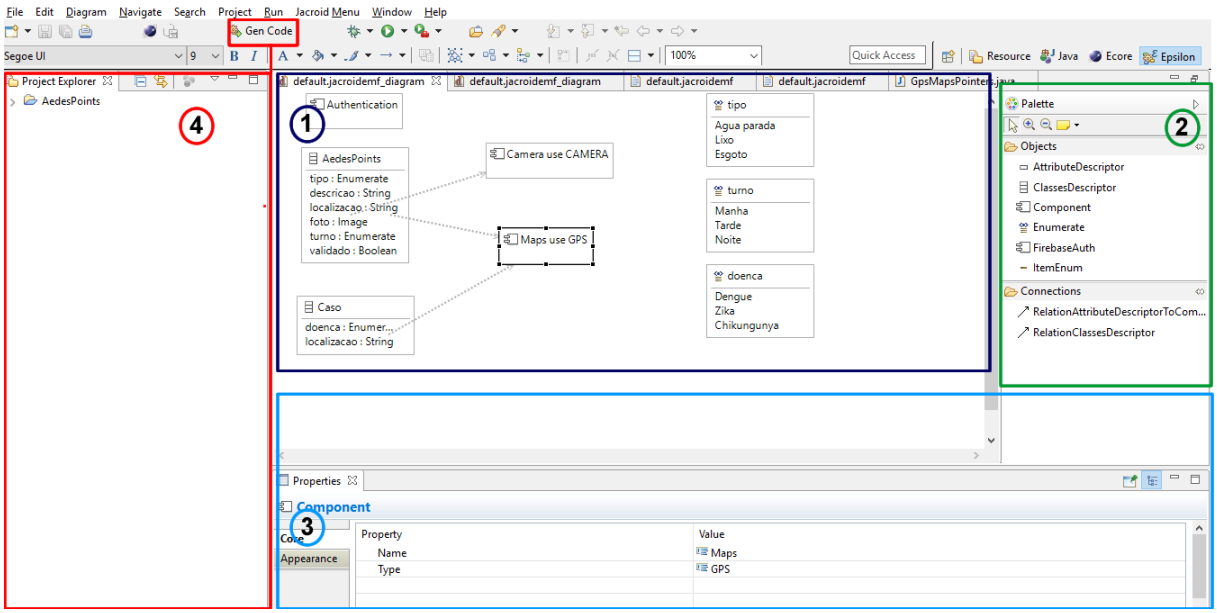


Figura 3.8: Editor GMF

1. **Área de desenho** - destinada a montagem de diagramas. Nesse espaço, componentes selecionados na paleta de construtores são dispostos de maneira a montar todo o contexto diagramático.

2. **Construtores** - Paleta criada obedecendo a conceitos da UML Profiles, de modo a facilitar e instruir a usuários utilizarem a ferramenta. Dentro de uma lógica de associação mnemônica, o usuário é levado a compor seu modelo, utilizando componentes num esquema de cima para baixo nas seções estabelecidas, assim em qualquer criação de regra parte-se da utilização dessa sequência de seções, para utilização dos construtores. A paleta de construtores está ilustrada na Figura 3.9 e possui as seguintes ações:

    - Objects - estabelece os componentes principais da área de desenho. *ClassesDescriptor* é um Object que define uma classe do metamodelo de

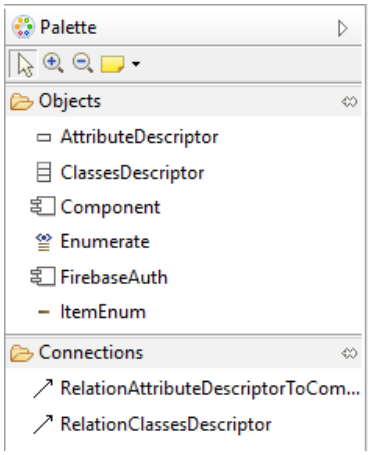


Figura 3.9: Paleta de construtores

entrada. O *AttributeDescriptor* é um Object que define atributos que devem ser associados as ClassesDescriptor. O *Component* é um Object que define o Component/Sensor que o usuário deseja utilizar. O *Enumerate* é um Object para representar um AttributeDescriptor que possua uma lista de opções, essas opções são definidas por cada *ItemEnum*. O *FirebaseAuth* é Object que tem a função de permitir a autenticação na base de dados do firebase. Observe as propriedades de ClassesDescritor, AttributeDescriptor, Component, Enumerate e ItemEnum na Tabela 3.1.

- Connections - estabelece a ligação entre os Objects. Para isso, em suas propriedades ele armazena a origem (source) e o destino (target). O *RelationClasseDescriptor* permite a ligação entre as ClassesDescriptor e o *RelationAttributeDescriptorToComponent* entre uma AttributeDescriptor e um Component.

3. **Propriedades** - Na aba propriedades é possível estabelecer valores de entrada para as propriedades dos componentes inseridos na área de desenho do diagrama.

4. **Geração de Código e o Botão Gen Code** - nessa área, em que está localizado o projeto criado inicialmente, irá abrigar uma pasta(*gen*), com o código fonte da aplicação que foi modelada e de acordo com a IDE selecionada (por exemplo, gen/eclipse). Para isso, é preciso estar com o diagrama aberto e clicar no botão "Gen Code". Ao clicar no botão, a Classe *M2TAction* dentro de *msc.android.jacroid.diagram* irá recuperar a informação do metamodelo construído no editor e efetuará a transformação junto aos templates.

Tabela 3.1: Propriedades ClassesDescriptor, Component e AttributeDescriptor

Component	Propriedade	Tipo de Dado
ClassesDescriptor	Name	String
ClassesDescriptor	TableName	String
Component	Name	String
Component	Type	none / GPS / CAMERA / NFC
AttributeDescriptor	ClassName	String
AttributeDescriptor	Name	String
AttributeDescriptor	Column Table	String
AttributeDescriptor	Exhibition Name	String
AttributeDescriptor	Input	select / text / checkbox / radiobutton / hidden
AttributeDescriptor	Is Column	Boolean
AttributeDescriptor	Primary Key	Boolean
AttributeDescriptor	Primary Key Table	Boolean
AttributeDescriptor	Type Data	String / int / Long / Boolean
Enumerate	Name	String
ItemEnum	Name	String
FirebaseAuth	Name	String

3.2.3 Visão Geral

A abordagem proposta é composta por três passos: a primeira é definida pela **Identificação dos objetos do domínio**, **Realização das transformações** e **Implantação da aplicação**, conforme a Figura 3.10.

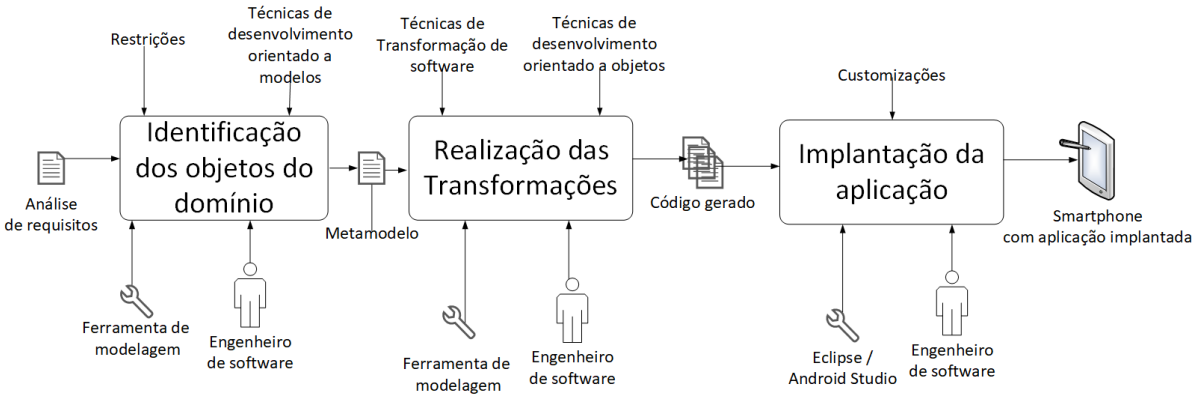


Figura 3.10: Etapas da Abordagem

No primeiro passo, inicia-se a identificação dos objetos do domínio para a aplicação

que se deseja implementar, com o auxílio da ferramenta de modelagem desenvolvida. No passo seguinte, em que se deve ter os objetos identificados, a ferramenta analisa as informações coletadas e realiza as transformações de software necessárias. Ao final dessa etapa o código fonte da aplicação já foi gerado. No último passo, é necessário realizar as modificações necessárias no código para adequar-se aos dados do desenvolvedor/engenheiro de software e efetuar a implantação da aplicação gerada no dispositivo com sistema Android escolhido.

Segue-se uma apresentação detalhada de cada passo da abordagem, além das técnicas e procedimentos utilizados. Para a definição da abordagem foi realizado um estudo de caso dentro do domínios da abordagem. Nas seções que se seguem, esse sistema é utilizado como exemplo, para facilitar o entendimento das entradas, saídas e controles de cada passo.

3.2.4 Etapas da Abordagem

Nesta seção, são descritas as etapas da abordagem, visando facilitar o desenvolvimento de aplicativos no domínio de pessoas como sensores utilizando as técnicas do desenvolvimento orientado a modelos.

Com intuito de demonstrar a aplicabilidade da abordagem, foi desenvolvida uma aplicação chamada de "Lembre-me!". A aplicação visa permitir que usuários ao terem ideias, registrem-a no aplicativo com a informação de localização e fotos do local. Essas ideias podem ser compartilhadas para aqueles que possuem o aplicativo instalado.

Identificação dos objetos do domínio

Inicialmente, o Engenheiro de Software deve realizar a análise de requisitos procurando identificar os possíveis objetos do domínio (requisitos funcionais). Essas informações irão servir como entrada para dar-se inicio a modelagem das *features* na ferramenta de modelagem. A especificação dos requisitos, não necessariamente, deve seguir uma regra em específico, com todos os artefatos que a análise requer, esta servirá mais como documentação da aplicação em desenvolvimento.

Como forma de ilustrar a análise, um dos artefatos mais comuns é a especificação de casos de uso. Em FOWLER (2014) temos que os casos de uso servem para descrever as interações típicas entre os usuários de um sistema e o próprio sistema, fornecendo uma narrativa sobre como o sistema é utilizado. Foram capturados cinco (5) casos de uso, são eles: cadastrar, visualizar e listar Lembrete, capturar foto e localizar GPS. Estão especificados nas Tabelas 3.2, 3.3, 3.4, 3.5 e 3.6, descrevendo seus fluxos normais e alternativos. O UC01 ou *USE CASE 01*, descrito na Tabela 3.2, aborda o processo de cadastro de um lembrete. Para essa ação, o usuário deve acessar a tela de listar, iniciar a tela de cadastro pelo botão (+), preencher os dados e salvar. Se algo estiver errado, ele deve corrigir os dados, de acordo com o fluxo alternativo 1. Caso o usuário deseje inserir uma foto, e a localização GPS, este deverá utilizar os fluxos alternativos 2 e 3, respectivamente. O Fluxo alternativo 2 necessita do UC04 (Tabela 3.5) e o fluxo alternativo

3 o UC05 (Tabela 3.6).

Tabela 3.2: Exemplo de Caso de Uso da aplicação "Lembre-me"

Nome: **UC01 - Cadastrar Lembretes**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar Lembretes 2. O usuário clica no botão Cadastrar Lembretes 3. O usuário preenche os dados e clica em salvar 4. Aplicação valida os dados e emite mensagem de sucesso. 5. Aplicação retorna a tela Listar Lembretes para o usuário
Fluxo Alternativo 1: 4. Os dados informados são inválidos 4.1 Aplicação emite mensagem para corrigir os dados. Fluxo Alternativo 2: 3. O usuário preenche os dados, mas deve informar uma foto 3.1 UC04 Fluxo Alternativo 3: 3. O usuário preenche os dados, mas deve informar a localização 3.1 UC05

Tabela 3.3: Exemplo de Caso de Uso da aplicação "Lembre-me"

Nome: **UCC02 - Listar Lembretes**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar Lembretes 2. Aplicação retorna os Lembretes cadastrados
Fluxo Alternativo: Não há.

Tabela 3.4: Exemplo de Caso de Uso da aplicação "Lembre-me"

Nome: **UCC03 - Visualizar Lembretes**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar Lembretes 2. A Aplicação retorna os Lembretes cadastrados 3. O usuário deve selecione um dos lembretes listados 4. A Aplicação retorna o lembrete selecionado
Fluxo Alternativo: Nenhum lembrete cadastrado.

Tomando como base as informações das especificações dos casos de uso, o Engenheiro de Software pode iniciar a orquestração dos componentes que jugue necessário a fim de atender aos requisitos especificados. Mas antes deverá criar o projeto da aplicação utilizando a ferramenta. Uma vez que criado o projeto, precisa-se criar um novo diagrama. Na Figura 3.11 temos o projeto criado, após seguir os passos anteriores. Dessa forma, já poderá ser iniciada a modelagem da aplicação.

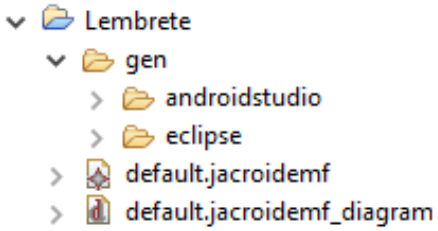


Figura 3.11: Projeto

Para realizar os casos de uso, *cadastrar lembretes*, *listar lembretes* e *visualizar lembretes*, o Engenheiro de Software necessita apenas criar um *Class Descriptor*, para essa aplicação, com o nome Lembrete, e seus *Attributes Descriptor*. Os *Attributes Descriptor* informados para essa aplicação foram: assunto do tipo String, localização do tipo String, descrição do tipo String e foto do tipo image.

Tabela 3.5: Exemplo de Caso de Uso da aplicação "Lembre-me"

Nome: **UC04 - Capturar Foto**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário clica no campo foto 2. A aplicação fornece a opção de usar foto da galeria 2.1. Selecione foto da galeria.
Fluxo Alternativo: 2. A aplicação fornece a opção de capturar nova foto 2.1. Capture a nova foto

Mais outros dois casos de uso necessitaram ser realizados, o *capturar foto* e *localizar GPS*. A realização desses está relacionada a inserção de dois *Components*, um cujo o tipo é GPS e outro Camera. Esses *Components* só funcionarão mediante o estabelecimento da relação entre um Attribute Descriptor e um Component. Essa relação pode ser feita utilizando a *Connection RelationAttributeDescriptorToComponent*. Ao final da modelagem, temos o modelo de domínio criado através da ferramenta, na Figura 3.12. Observe que foram criados 3 objetos, Lembrete, Maps e Camera e não 5, conforme os casos de uso capturados inicialmente. Na Figura é possível perceber as ligações entre os objetos, como a estabelecida entre o atributo localização e o componente Maps, e entre o atributo foto e o componente Camera.

Tabela 3.6: Exemplo de Caso de Uso da aplicação "Lembre-me"

Nome: **UC05 - Localizar GPS**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário clica no campo localização 2. A aplicação abre a tela de GPS 3. O usuário obtém a localização.
Fluxo Alternativo: 2. A aplicação abre a tela de GPS 2.1 O GPS não está ativado. 2.2 O usuário habilita o GPS 3. O usuário obtém a localização.

A modelagem realizada com a ferramenta, além de gerar o modelo de domínio, também apresenta o mesmo modelo descrito em forma textual, por meio de um *script* XML. Esse *script* é composto por *tags* que identificam qual *component* foi selecionado no modelo. Por exemplo a *tag* que identifica uma *Class Descriptor* é representa pelo nome *classes*. Esse arquivo tem suma importância na etapa de realização de transformações, em que as *tags* serão interpretadas e convertidas no código fonte correspondente. Mais detalhes sobre o script XML será abordado na próxima etapa.

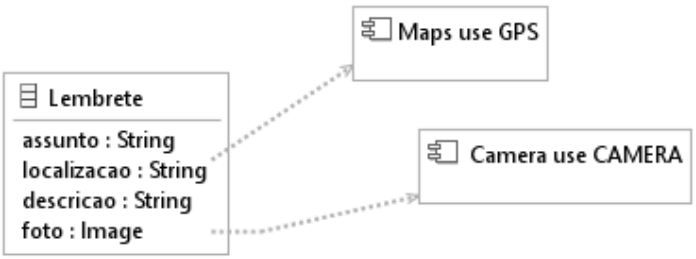


Figura 3.12: Modelo de Domínio do Aplicativo

A ferramenta impõe algumas restrições que seguiram as determinações feitas na *modelagem de features*, a qual já foi apresentada na seção 3.2.2. Uma das restrições é que um atributo da classe só pode se ligar a um único componente, porém se outra classe necessite do mesmo componente, este poderá ser reutilizado.

Realização das transformações

Após identificar os objetos, tem inicio a etapa em que são realizadas as transformações M2T. Essa etapa é responsável por definir como os modelos criados com o editor refletirão na aplicação, além de permitir o reúso dos mesmos componentes em outras aplicações do mesmo domínio. Essas transformações são realizadas de forma automatizada, necessitando apenas que o Engenheiro de Software acione o botão *Gen Code*.

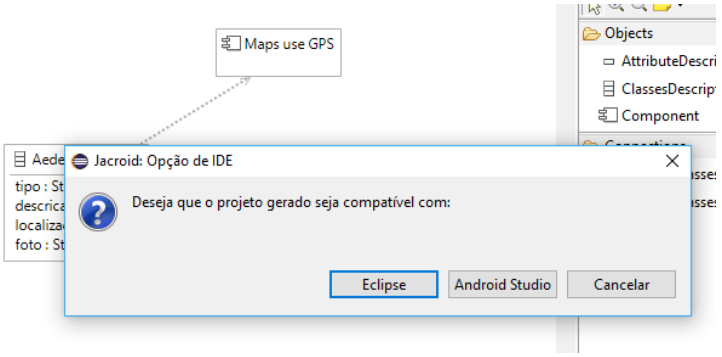


Figura 3.13: Seleção da IDE

A ferramenta desenvolvida permite que o código fonte seja compatível tanto com a IDE Eclipse quanto com a Android Studio. Nesse sentido, no momento em que o usuário inicia a ação de gerar o código, a ferramenta emite uma mensagem, solicitando a confirmação de qual IDE é mais satisfatória para ele (Figura 3.13).

```
<?xml version="1.0" encoding="UTF-8"?>
<jacroidEMF:Projeto xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:jacroidEMF="jacroidEMF">

  <classes name="Lembrete" tableName="lembrete">
    <attributes classname="Lembrete" columnTable="assunto"
        exhibitionName="Assunto" input="text" isColumn="true" name="assunto"/>
    <attributes classname="Lembrete" columnTable="localizacao"
        exhibitionName="Localizacao" input="text" isColumn="true" name="localizacao"/>
    <attributes classname="Lembrete" columnTable="descricao"
        exhibitionName="Descricao" input="text" isColumn="true" name="descricao"/>
  </classes>

  <component name="Maps" type="GPS"/>

  <relationAttributeToComponent
    source="//@classes.0/@attributes.1"
    target="//@component.0"/>
</jacroidEMF:Projeto>
```

Figura 3.14: Arquivo XML do Modelo de Domínio

Discutiu-se na etapa anterior da abordagem que além do modelo de domínio em que apresenta as ligações entre as classes, atributos e componentes, existe um arquivo no formato XML com todas essas informações de forma detalhada. Essas informações auxiliam no processo de transformação do modelo para a aplicação final propriamente dita. Para o exemplo apresentado na Figura 3.12, temos o arquivo ilustrado na Figura 3.14.

Neste arquivo estão as propriedades das *tags*: classes, seus attributos, componentes e relações. Tendo como exemplo o atributo *localizacao*, temos: informações de qual Classe pertence, neste caso *Lembrete*; qual o tipo de entrada (input) a ser utilizada, cuja escolha do *text* repercutirá no código Android em um TextView e EditView; dentre outras informações.

Observando a *tag* `relationAttributeToComponent`, a mesma apresenta duas propriedades: `source` (informação de origem) e `target` (o alvo). Dessa forma, seguindo o exemplo apresentado temos que o `attribute.1`(localizacao) da `classes.0` (Lembrete) está ligado ao `component.0` (Maps). Esses dados informam ao processo de transformação

M2T que o método de chamada para o Componente Maps deve estar associado ao atributo localização.

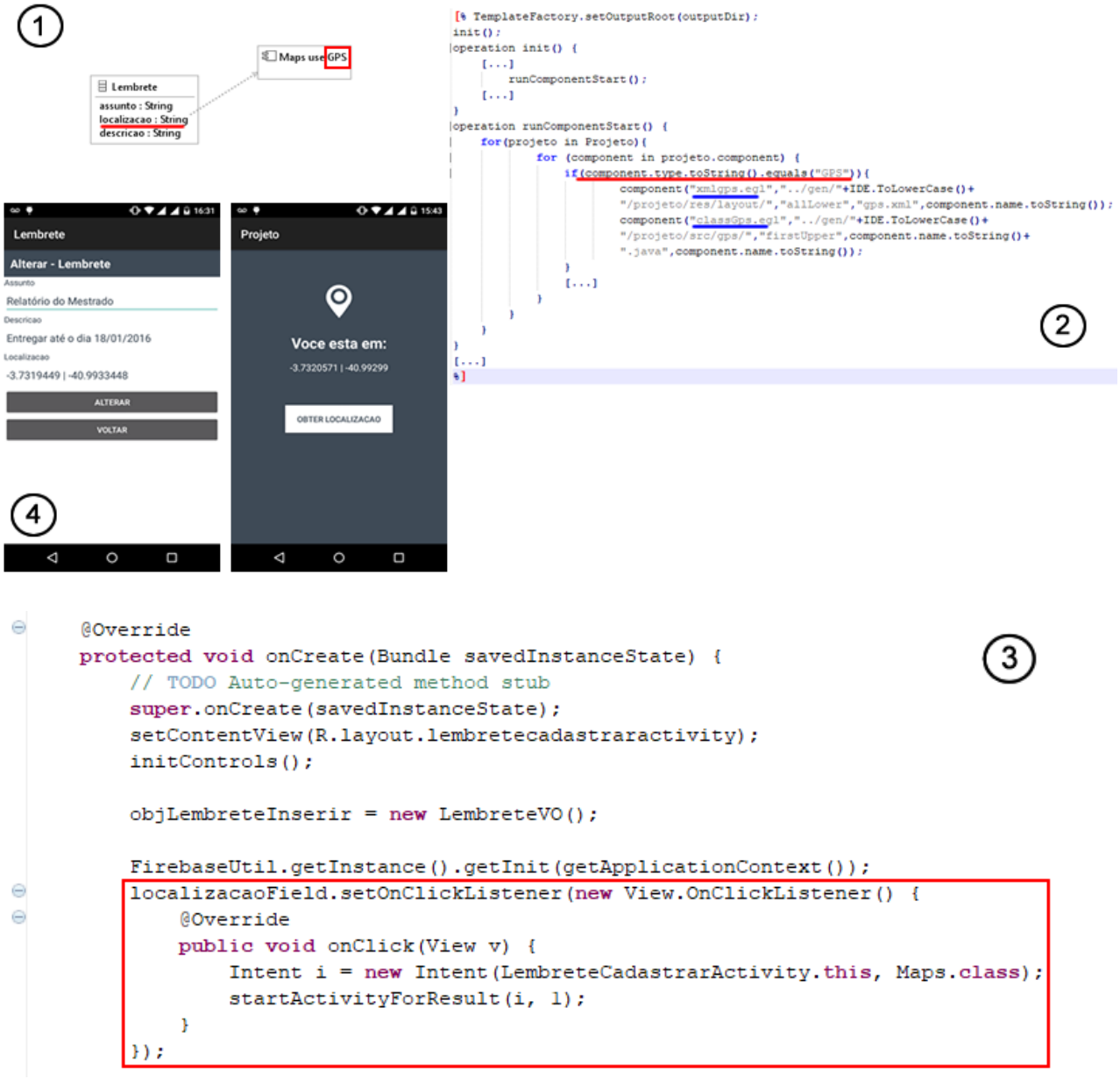


Figura 3.15: Modelo de Domínio x Trecho Template x Tela de Aplicativo Gerado X Código Fonte Gerado

A Figura 3.15 apresenta um exemplo de transformação, a imagem representada pelo item **(1)** ilustra o modelo de domínio para a classe *Lembrete* com seus atributos (assunto e localização) e o Component *Maps* do tipo GPS. Observe em destaque a relação entre o atributo localização e o componente Maps. O arquivo XML correspondente ao modelo, é ilustrado na Figura 3.14; a imagem representada pelo item **(2)**, é o trecho do *template* principal escrito em EGL. O trecho destacado em vermelho, ilustra o código do *template* principal em que realiza a decisão de escolha do componente do tipo GPS. Destacado em azul, são os outros *templates* necessários para geração da classe GPS, o *xmlgps.egl*, responsável pelo layout xml da tela de GPS e o

classGPS.egl, que tem a função de criar o código da classe que irá interagir com o layout; o item **(3)** Representa o código fonte resultante para o atributo localização, após as transformações. No trecho em destaque temos a chamada de método para a tela de GPS, componente a qual o atributo localização está relacionado no modelo de domínio; o item **(4)** Representa os layouts das telas de cadastro de lembretes e tela do GPS, ambas geradas automaticamente.

Todo o processo de transformação é realizado pelo método `run`, na classe M2TAction, localizada no componente `msc.android.jacroid.diagram`. No método `run`, é realizado o acesso ao editor e extração do metamodelo. Após isso, o módulo EGL, responsável por gerenciar todas as interações entre o metamodelo e os *templates*, é iniciado. Nesse momento, as transformações vão sendo realizadas e o código fonte gerado é armazenado. Dependendo da IDE escolhida pelo usuário, a ferramenta armazena o código fonte no diretório correspondente, `gen/nome_da_ide`, dentro do projeto criado na ferramenta.

Algumas definições para a geração de código são realizadas em tempo de execução, como: quais *templates*, quais sensores, quais as permissões de acesso ao dispositivo celular serão necessárias. Para isso a ferramenta realiza uma checagem no metamodelo por meio das *tags* setadas e realiza a transformação. Essas transformações são detalhadas na Tabela 3.7.

Na etapa final é realizada a cópia de arquivos padrão, como imagens, diretórios e outros arquivos de configuração que compõem um projeto de aplicativo Android.

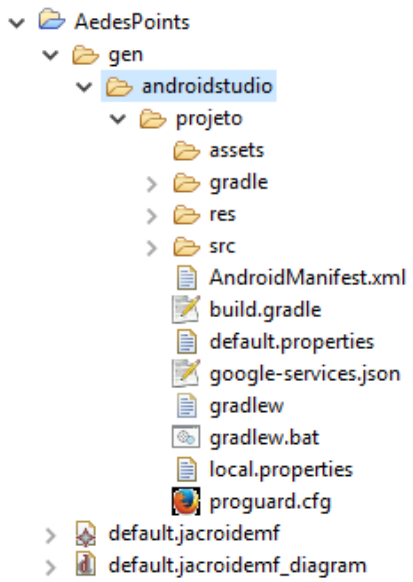


Figura 3.16: Estrutura do Código Fonte

Na Figura 3.16 é ilustrada a estrutura de código gerada, a qual é padrão de aplicações para a plataforma Android. Uma particularidade, é a utilização do *Gradle* (MUSCHKO, 2014), como ferramenta de compilação baseada em JVM. Ele tem a capacidade de combinar as melhores

features de outras ferramentas de compilação, como maven [1], ant [2], Ivy [3], Gant [4]. Outro fato que deve ser mencionado é que devido a adoção da Google ao Android Studio, fez com que o ADT do Eclipse viesse a entrar em recessão. O Gradle surge como forma de alavancar mais uma vez o Eclipse.

No diretório */assets* é onde podem ser colocados arquivos que queiram a ser usados quando o projeto estiver em execução, imagens e outros arquivos.Outra pasta que compõe o projeto é a */res* que é composta por */drawablehdpi, /drawable-ldpi, /drawable-mdpi*, as quais têm a função de armazenar as imagens do projeto, mais especificamente as que serão usadas como ícone que representará a aplicação quando a mesma for instalada no dispositivo com Android. O diretório */res* ainda possui as pastas */layout*, */values* e *menu*.O diretório */layout* possui os xmls das telas da aplicação, como as de cadastro, alteração, camera, gps, dentre outras. Já */values* é composta por alguns arquivos como o *arrays.xml* e o *strings.xml*. No arquivo *arrays.xml* são descritos os arrays (vetores) que possam a ser usadas pela aplicação. No arquivo strings são definidos valores padrões para strings que forem a ser usadas nos XML do layout. Em *menu* temos o menus necessários na tela de listar e na barra de ações.

O */src* é composto por 5 pacotes, para o exemplo em questão. A estrutura de organização do código segue o padrão de projetos MVC (*Model, View and Controler*). Na camada de Modelo, representada pelo diretório de mesmo nome, encontram-se as classes de modelos, as quais descrevem as Entidades. O diretório */util*, encontra-se a classe FirebaseUtil, a qual possui em sua implementação, dois padrões de projeto, o *Singleton*, para garantir que a classe tenha somente uma instância e fornecer um ponto global de acesso a ele, e a *Facade*, a fim de fornecer uma interface unificada para acessar os métodos.

Os pacotes *camera e gps* possuem as classes necessárias para acessarem ao sensores que lhes dizem respeito. Nas classes que fazem referencias a esses pacotes é apresentado um código exemplo para possíveis customizações.

Por fim , temos as Classes de Visão e Controle que estão presentes no diretório, Telas. Para cada entidade é criada uma tela de listagem, de cadastro, de alteração e exclusão.

Como já foi mencionado anteriormente, a Tabela 3.7 apresenta as *tags* e suas respectivas regras de transformação. As *tags* que estão sublinhadas, representam cada uma um nó, os quais englobam as *tags* sucessivas na tabela como suas propriedades. Por exemplo, a *tag* classes, possui as propriedades name e tableName, e assim por diante.

[1] https://maven.apache.org/
[2] http://ant.apache.org/
[3] http://ant.apache.org/ivy/
[4] https://gant.github.io/

Tabela 3.7: Regras de Transformação

Tag	Regras de Transformação
classes	Essa *Tag* é responsável por identificar entidades dentro do modelo de domínio especificado. Cada entidade criada sera processada pelo gerador que irá transformá-la em classes atendendo ao padrão de projeto MVC. Será criada a classe de modelo e persistência de dados (firebase), classes de layouts de tela em Android (View) e classes controladoras com validações.
name	Nesta *tagged Value* o Engenheiro de Sistema informa o nome da Classe, assim como o nome apresentado no layout da tela.
tableName	Nesta *tagged Value* o Engenheiro de Sistema informa o nome da tabela/-raiz que irá representar os dados armazenados no Realtime Database no Firebase.
attributes	Essa *Tag* é responsável por identificar os atributos de uma classe no modelo de domínio especificado.
classname	Nesta *tagged Value* o Engenheiro de Sistema informa o nome da classe a qual o atributo pertence.
name	Nesta *tagged Value* o Engenheiro de Sistema informa o nome do atributo.
exhibition name	Nesta *tagged Value* o Engenheiro de Sistema informa a descrição do nome que será transformado pelo gerador em um TextView da tela em Android. Esse nome do atributo aparecerá do acima do campo.
column table	Nesta *tagged Value* o Engenheiro de Sistema informa o nome da coluna
is column	Nesta *tagged Value* o Engenheiro de Sistema informa true or false para identificar se o atributo será ou não armazenado como coluna.
input	Nesta *tagged Value* o Engenheiro de Sistema informa o tipo de apresentação (select, text, checkbox, radiobutton) do atributo. Com isso, o gerador transforma em código correspondente na linguagem Android.
type Data	Nesta *tagged Value* o Engenheiro de Sistema informa o tipo de apresentação (String, int, Long, Boolean) do atributo. Com isso, o gerador transforma em código correspondente na linguagem Android.
primary key	Nesta *tagged Value* o Engenheiro de Sistema informa true or false para identificar se o atributo será ou não chave primária.
primary key table	Nesta *tagged Value* o Engenheiro de Sistema informa true or false para identificar se o atributo será ou não chave primária da tabela.

Tabela 3.7: Continuação

Tag	Regras de Transformação
enumerate	Essa *Tag* é responsável por identificar um atributo que terá itens para escolha. É necessário que o atributo possua input (select ou radiobutton) e mesmo nome. Cada enumerate é composto por um ou mais itemEnum. Com isso, o gerador transforma em código correspondente na linguagem Android.
itemEnum	Essa *Tag* é responsável por identificar um item de um enumerate.
component	Essa *Tag* é responsável por identificar um novo component de acordo com o tipo selecionado. Com isso, o gerador transforma em layout de tela correspondente na linguagem Android.
name	Nesta *tagged Value* o Engenheiro de Sistema informa o nome do componente.
type	Nesta *tagged Value* o Engenheiro de Sistema informa o tipo de componente (GPS, Camera). Com isso, o gerador transforma em código correspondente na linguagem Android. Ou seja, uma tela que realize as ações de GPS ou de Camera. A ferramenta ainda análise quais as permissões necessárias a esses componentes na aplicação. Para Camera são necessárias: *<uses-permission android:name="android.permission.CAMERA"/>, <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/> e <uses-feature android:name="android.hardware.camera"/>*. Para o GPS são as seguintes: *<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/> e <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>*.
Relation Attribute To Component	Nesta *tagged Value* o Engenheiro de Sistema informa qual atributo está relacionado com o componente. Para isso é necessário informar o alvo (target) e a origem (source). O atributo associado ao componente é identificado pelo gerador, o qual realiza as transformações necessárias a fim de permitir que o campo (EditText do Android) inicie a tela do Componente.
Relation Classes-Descriptor	Nesta *tagged Value* o Engenheiro de Sistema informa a relação entre as classes. Para isso é necessário informar o alvo (target) e a origem (source). A associação entre classes faz com que o gerador crie uma chave (key) estrangeira. A chave estrangeira fica armazenada na classe source.

Tabela 3.7: Continuação

Tag	Regras de Transformação
FirebaseAuth	Nesta *tagged Value* o Engenheiro de Sistema informa a necessidade de autenticação para uso da aplicação. A ferramenta utiliza-se do Firebase, o qual necessita algumas permissões para funcionar efetivamente, são elas: *<uses-permission android:name="android.permission.INTERNET"/> e <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>* . A ferramenta gera um tela de login/signin, a qual requere um email e senha.

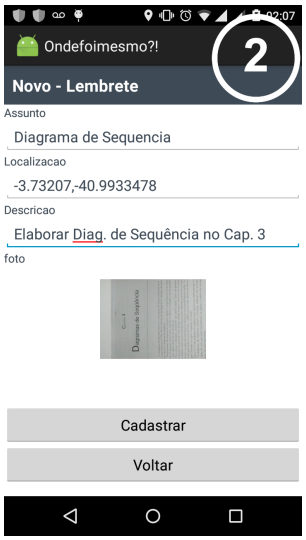
Recapitulando, a fase de transformação, tem como resultado o código fonte dos seguintes artefatos do aplicativo: 1) das telas do aplicativo (Activitys); das funções responsáveis por restringir o conteúdo dos atributos das entidades; 3) das funções responsáveis pela manipulação dos dados (persistência). Na próxima etapa, implantação da aplicação, é demonstrado o processo de *deploy* da aplicação e o funcionamento da mesma.

Implantação da aplicação

Na fase de implantação da ferramenta é realizado o *deploy* do código fonte no dispositivo com o sistema operacional Android. Nesta fase também são realizados os testes de sistema, cujo aplicativo é executado com o intuito de verificar se os aspectos gerais do sistema estão sendo atendidos. Os requisitos do sistema são testados através da execução do aplicativo pelo engenheiro de software.

Com o projeto pronto o usuário deverá adicioná-lo a IDE Eclipse ou Android Studio e customizar da forma que achar conveniente. Para ambos é necessário a instalação do JDK 7 ou superior. Para o uso do Eclipse, é preciso o *Android Developer Tools* (ADT), um plugin para o Eclipse que fornece acesso baseado em GUI a muitas das ferramentas do *Software Development Kit* (SDK) do Android e o Gradle, método suportado de criação de aplicativos para Android e Gradle. No Android Studio o mesmo já possui por padrão o Gradle e outras ferramentas para auxiliar o desenvolvimento e *deploy* das aplicações.

Como as aplicações usam o Firebase, é preciso criar sua conta/aplicação e substituir o arquivo *google-services.json*. Uma vez o projeto adicionado a IDE escolhida, o usuário precisa apenas colocar o projeto pra rodar. Se houver um dispositivo conectado ao computador, o *deploy* poderá ser realizado nele, caso contrário, a IDE iniciará um Emulador, que possui todas as características necessárias para ver a aplicação em funcionamento. Essa ação pode demorar alguns instantes.




```java
package telas;

import modelo.LembreteVO;

 /**@author Willamys Araujo
 **Generate for Jacroid**/

public class LembreteCadastrarActivity extends Activity{

    //Firebase
    private FirebaseDatabase database;
    private DatabaseReference reference;
    private String LEMBRETES_CHILD = "lembretes";

    private static final String tag = "LembreteCadAct";
    private EditText assuntoField;
    private EditText localizacaoField;
    private EditText descricaoField;
    private EditText fotoAuxField;
    private byte[] imgData;
    private int PICK_IMAGE_REQUEST = 1;
    private ImageView fotoField;
    private String message;
    private long idB;
    private LembreteVO objLembrete;
    //objeto criado com o intuito de obter os dados alterados
    private LembreteVO objLembreteInserir;

    public LembreteCadastrarActivity() {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lembretecadastraractivity);
        initControls();

        objLembreteInserir = new LembreteVO();

        FirebaseUtil.getInstance().getInit(getApplicationContext());
        localizacaoField.setOnClickListener(new View.OnClickListener() {
        fotoAuxField.setVisibility(android.view.View.INVISIBLE);
        fotoField.setOnClickListener(new OnClickListener() {

        Button botao = (Button) findViewById(R.id.ButtonSendFeedback);
        botao.setOnClickListener(new View.OnClickListener(){

        Button botaoVoltar = (Button) findViewById(R.id.ButtonBack);
        botaoVoltar.setOnClickListener(new View.OnClickListener(){
    }

    public void insert(){

    protected boolean validation(LembreteVO objLembreteInserir) {
```

Figura 3.17: Tela de Cadastro Gerada e código fonte gerados automaticamente

Na Figura 3.17 é ilustrado o resultado das transformações e como uma parte de aplicação é apresentada após a implantação. No item **(1)** temos um trecho do código fonte da Classe `LembreteCadastrarActivity`, responsável pelo funcionamento da tela de cadastro; no item **(2)** é ilustrado a interface da tela de cadastro. Ambas geradas automaticamente pela ferramenta e sem nenhuma implementação adicional.

3.3 Considerações Finais

Este capítulo apresentou todas as etapas, de forma detalhada, da proposta de abordagem para o desenvolvimento de aplicações para dispositivos móveis no domínio de pessoas como sensores. Foram demonstrados as técnicas, procedimentos e ferramentas que auxiliaram na implementação da mesma, bem como a sua aplicação através de um exemplo simples.

Foi apresentado todo o processo de desenvolvimento da ferramenta de modelagem

baseada na abordagem proposta. A construção desta foi guiada pela Engenharia de Domínio (ED), passando por todas as suas etapas. Dentro das etapas da ED foram selecionadas técnicas, como modelo de *features* conhecido como *Feature-Oriented Domain Analysis* (FODA) e a abordagem de *templates*. Como padrão arquitetural destacou-se a utilização da *camada fina de dados*, a qual facilita a integração entre o gerador e a ferramenta de modelagem DSL, estas representadas pelo *framework* Eugenia.

A abordagem pôde mostrar que apenas com a orquestração de componentes na ferramenta, é possível obter uma aplicação completa e funcional no âmbito de pessoas como sensores (*crowdsensing*). O processo de transformação do modelo construído até o código fonte também foi contemplado, dando ênfase e detalhando cada arquivo e diretório presente no contexto da aplicação desenvolvida.

O exemplo apresentado foi a aplicação *Lembre-me*, a qual apresentava 5 casos de uso que foram mapeados através da ferramenta de modelagem que foi desenvolvida com o intuito de aplicar a abordagem na criação de novos apps. Esse App foi selecionado por apresentar poucos requisitos e mesmo assim, conseguir apresentar as principais *features* que uma aplicação baseada em pessoas como sensores utilizaria, como o GPS, Câmera e Mapa.

O App *Lembre-me* está disponível na plataforma *github.com*[5] e pode ser obtida a partir do link *https://github.com/willamys/lembre-me*.

No próximo capítulo, será apresentado um estudo de caso utilizando-se a abordagem, além da sua avaliação.

[5]É uma plataforma de hospedagem de código-fonte com controle de versão. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos Open Source de qualquer lugar do mundo.

4

Estudo de caso e avaliação da abordagem

Neste capítulo é apresentado um cenário para ilustrar a aplicação da abordagem. Em seguida, é realizada uma avaliação apoiada pela abordagem *Goal Question Metric* (GQM), o qual irá analisar o reúso para as aplicações avaliadas.

4.1 *Aedes Points*

O *Aedes Points* é um aplicação desenvolvida com base na experiência da dissertação de mestrado de (OLIVEIRA, 2015), a qual tinha como objetivo analisar e demostrar a distribuição espacial dos locais acometidos com o desequilíbrio ambiental que possam influenciar na prevalência dos casos de dengue. A pesquisa dela levou em consideração quatro bairros do município de Juazeiro/BA. A metodologia realizada consistia em georreferenciar o desequilíbrio ambiental (lixo, esgoto à céu aberto e água parada) nos quatro bairros mais acometidos pela dengue, no município de Juazeiro-BA, com um total de 181 casos de dengue confirmados nos quatro bairros.

Todos os pontos georreferenciados foram realizados pela pesquisadora sem ter havido colaboração de nenhuma outra pessoa. A obtenção dos pontos foi realizada com equipamento GPS cedido pela UNIVASF [1] e exportados para uma planilha. Posteriormente, os dados foram digitados, um a um, no Google Earth [2].

Como forma de tornar a pesquisa colaborativa, no âmbito de pessoas como sensores, foi pensando na aplicação em que pessoas comuns, com o uso do *smartphone*, pudessem realizar a marcação do pontos vistos como de desequilíbrio ambiental e casos de pessoas (anônimas) acometidas pelas enfermidades transmitidas pelo *Aedes aegypti*, como a Dengue, Zica e Chikungunya. Dessa forma, as autoridades locais poderiam tomar providências para impedir a proliferação do mosquito contribuindo na diminuição de novos casos.

Na Figura 4.1 é ilustrado o mapa dos casos de dengue notificados em Juazeiro/BA. No bairro Santo Antônio: as notificações georreferenciadas estão representadas pelos pontos em azul; no Centro: em amarelo, no Dom José Rodrigues: em verde, e Antônio Guilhermino, em

[1] Universidade Federal do Vale do São Francisco
[2] https://www.google.com.br/intl/pt-PT/earth/

vermelho.

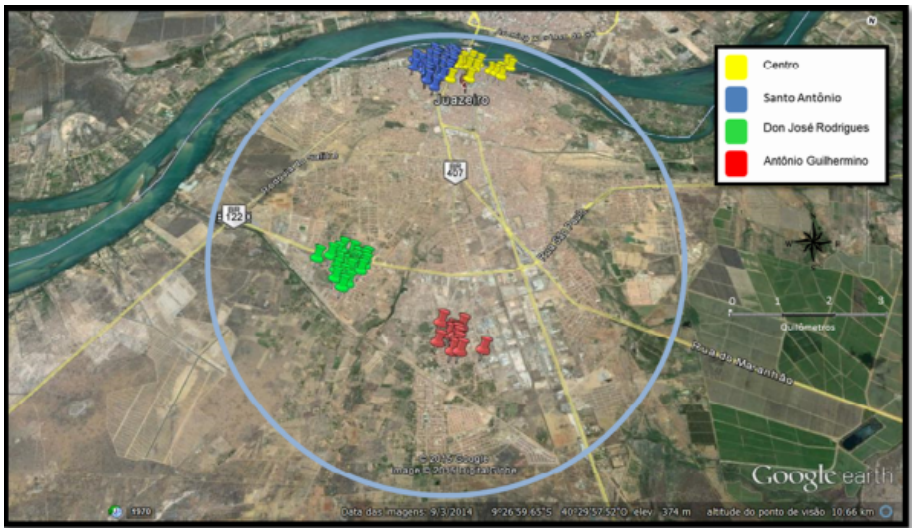


Figura 4.1: Mapa dos casos de dengue notificados em Juazeiro/BA (OLIVEIRA, 2015)

4.1.1 Identificação dos objetos do domínio

Primeiramente, foi realizado um análise de requisitos sobre o tema abordado na dissertação. Nessa análise foi realizada a especificação de requisitos, descrevendo interações típicas entre os usuários de um sistema e o próprio sistema. Para o domínio da aplicação *AedesPoints* foram capturados doze (12) casos de uso, são eles: Cadastrar AedesPoints (UC01), Atualizar AedesPoints (UC02) e Listar AedesPoints (UC03), Cadastrar Casos (UC04), Atualizar Casos (UC05) e Listar Casos (UC06), Camera (UC07), GPS (UC08), Mapear Casos e AedesPoints (UC09), Efetuar Login (UC10), Efetuar Logout (UC11), Realizar Sign in (UC12). Dois (2) casos de uso são especificados nas Tabelas 4.1 e 4.2, descrevendo seus fluxos normais e alternativos, os demais encontram-se no Apêndice B.

O UC04 ou *USE CASE 04*, descrito na Tabela 4.1, aborda o processo de cadastro de um Caso. Para essa ação, o usuário deve acessar a tela de listar, iniciar a tela de cadastro pelo botão (+), preencher os dados e salvar. Se algo estiver errado, ele deve corrigir os dados, de acordo com o fluxo alternativo 1. Caso o usuário deseje inserir a localização GPS, este deverá utilizar o fluxo alternativo 2, o qual necessita do UC08 (Tabela 4.2).

O processo de modelagem, através da ferramenta desenvolvida, foi iniciado após a etapa de análise. Para os casos de uso *Efetuar Login* (UC10), *Efetuar Logout* (UC11) e *Realizar Sign in* (UC12) foi inserido o objeto *Authentication*. Para os casos de uso Cadastrar, Atualizar e Listar Casos, apenas uma Classe precisa ser mapeada, a Casos. Da mesma forma para Aedes Points. Para GPS e CAMERA é necessário adicionar um componente para cada um. A definição de um componente implica que este deve ser ligado a um atributo da classe que realizará a sua invocação. As outras informações mapeadas são relacionadas aos atributos que cada classe necessita. O diagrama do modelo da aplicação é ilustrado na Figura 4.2.

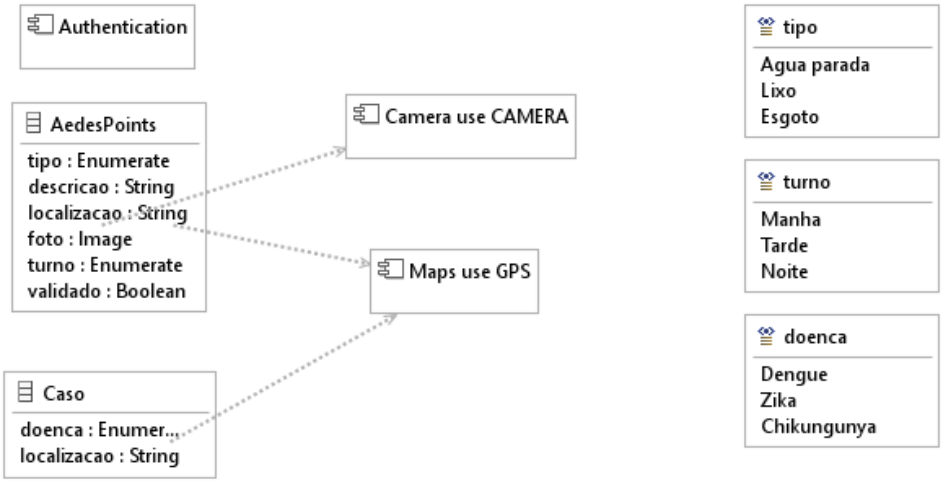


Figura 4.2: Diagrama do Modelo da aplicação Aedes Points

Tabela 4.1: Exemplo de Caso de Uso da aplicação "AedesPoints"

Nome: **UC04 - Cadastrar Casos**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar Casos 2. O usuário clica no botão Cadastrar Casos. 3. O usuário preenche os dados e clica em salvar. 4. Aplicação valida os dados e emite mensagem de sucesso. 5. Aplicação retorna a tela Listar Casos para o usuário.
Fluxo Alternativo 1: 4. Os dados informados são inválidos. 4.1 Aplicação emite mensagem para corrigir os dados. Fluxo Alternativo 2: 3. O usuário preenche os dados, mas deve informar a localização. 3.1 UC08

Tabela 4.2: Exemplo de Caso de Uso da aplicação "AedesPoints"

Nome: **UC08 - GPS**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário clica no campo localização. 2. A aplicação abre a tela de GPS. 3. O usuário obtém a localização.
Fluxo Alternativo: 2. A aplicação abre a tela de GPS, mas o mesmo não está ativado no dispositivo. 2.1 O usuário habilita o GPS. 3. O usuário obtém a localização.

4.1.2 Realização das Transformações

Após a identificação dos objetos do domínio, na fase anterior, o processo de realização das transformações é iniciado. O engenheiro de software necessita apenas acionar o botão GenCode que a ferramenta irá realizar todo o trabalho. Como foi abordado no capítulo 4, as transformações seguem algumas regras. Por exemplo, será detalhado a classe Caso, a qual possui dois atributos, `doença` do tipo `Enumerate` e `localização` do tipo `String` e que está relacionada ao componente Maps. Veja Figura 4.2.

De acordo com as regras, a Classe Caso irá repercutir na geração de 4 Classes, `Caso AlterarActivity`, `CasoCadastrarActivity`, `CasoListarActivity` e `Caso Adapter`. Está última, auxilia a classe `CasoListarActivity` na exibição dos casos cadastrados. As outras três realizam a ação a qual o nome propriamente diz: alterar, cadastrar e listar os casos. Essas três classes, são telas do aplicativo.

Pode ser visto no diagrama que o atributo `localização` está ligado ao componente `Maps`. Ele por se tratar de um String, será na tela da aplicação um `EditText` (campo para preenchimento) e por possui essa ligação a um componente, terá uma ação de invocar a tela de GPS. A invocação a tela de GPS, é realizada no momento que o cursor da aplicação selecionar o atributo localização. Uma vez a tela aberta, o usuário terá a opção de obter as coordenadas que serão setadas no campo de localização.

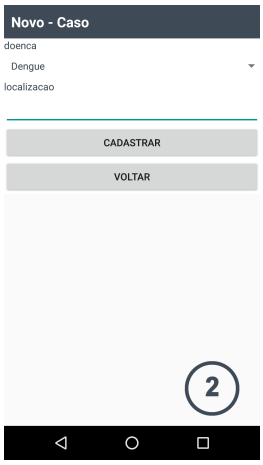
```
1   package telas;
2
3⊕  import modelo.CasoVO;
56
57⊖  /**@author Willamys Araujo
58   **Generate for Jacroid**/
59
60  public class CasoCadastrarActivity extends Activity{
61
62      private FirebaseDatabase database;
63      private DatabaseReference reference;
64      private String CASOS_CHILD = "casos";
65
66      private static final String tag = "CasoCadAct";
67      private Spinner doencaField;
68      private EditText localizacaoField;
69      private String message;
70      private long idB;
71      private CasoVO objCaso;
72      //objeto criado com o intuito de obter os dados alterados
73      private CasoVO objCasoInserir;
74
75⊖     private void initControls(){
76          doencaField =  (Spinner) findViewById(R.id.Spinnerdoenca);
77          localizacaoField = (EditText) findViewById(R.id.EditTextlocalizacao);
78      }
79
80⊕     public CasoCadastrarActivity() {
83
85⊕     protected void onCreate(Bundle savedInstanceState) {
146
147⊕    public void insert(){
177
178⊕    protected boolean validation(CasoVO objCasoInserir) {
204⊕    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
217⊕    public void onBackPressed(){
223⊕    public String getAppName(){
234 }
```

Figura 4.3: Código fonte gerado da Classe Caso da aplicação Aedes Points

O atributo `doença`, do tipo `enumerate`, possui três opções de seleção: `Dengue`, `Zika` e `Chikungunya`. Este atributo por se tratar de um `enumerate` na tela da aplicação, o mesmo será um `Spinner` (campo de seleção de itens). Na Figura 4.3 é ilustrado o código fonte da classe de `CasoCadastrarActivity`.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ScrollView01" android:layout_width="fill_parent"
    android:layout_height="wrap_content"  android:scrollbars="vertical">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:background="@color/ViewBgDefault"
    android:gravity="center_horizontal" android:orientation="vertical" >
<TextView
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:background="#3e4a56" android:padding="10dp" android:text="Novo - Caso"
    android:textColor="@color/white" android:textSize="20sp" android:textStyle="bold" />
<TextView
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:text="doenca" android:textColor="#3e4a56"/>
<Spinner
    android:id="@+id/Spinnerdoenca" android:layout_height="wrap_content"
    android:layout_width="fill_parent" android:textColor="#3e4a56"/>
<TextView
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:text="localizacao"  android:textColor="#3e4a56"/>
<EditText
    android:id="@+id/EditTextlocalizacao" android:layout_height="wrap_content"
    android:inputType="textLongMessage"  android:layout_width="fill_parent"
    android:textColor="#3e4a56"/>
<Button
    android:id="@+id/ButtonSendFeedback" android:layout_height="wrap_content"
    android:text="Cadastrar" android:layout_width="fill_parent"/>
<Button
    android:id="@+id/ButtonBack" android:layout_height="wrap_content"
    android:text="Voltar" android:layout_width="fill_parent"/>
</LinearLayout>
</ScrollView>
```

1



2

Figura 4.4: Telas do aplicativo gerado pela ferramenta

Na plataforma Android, para cada classe que representa uma `Activity` (uma Tela), a mesma necessita de um layout desta, mas escrita no formato XML. A Figura 4.4 ilustra o layout para a tela da Classe `CasoCadastrarActivity`, representada pelo item **(1)**. Observe que são usadas *tags* com as informações referentes aos componentes de tela que estarão presentes. O `EditText` do atributo localização, em destaque na figura, apresenta características da cor do texto, capacidade para textos longos no campo, dentre outras. A tela de cadastro de caso, em funcionamento em um dispositivo, é mostrado no item **(2)**.

Após apresentar as transformações da Classe `CadastrarCasoActivity`, na Figura 4.5 é ilustrado a estrutura do código fonte da aplicação que foi gerado automaticamente para o padrão de aplicações para a plataforma Android.

No diretório */assets* é onde podem ser colocados arquivos que queiram ser usados quando o projeto estiver em execução, imagens e outros arquivos. Na pasta */res* são armazenadas as imagens do projeto, como o ícone que representará a aplicação no dispositivo. Neste mesmo diretório temos ainda as pastas */layout* (arquivos de layout no formato xml), */values* e *menu*.

O */src* é composto por 5 pacotes, para o exemplo em questão. A estrutura de organização do código segue o padrão de projetos MVC (*Model, View and Controler*). Na camada de Modelo,

representada pelo diretório de mesmo nome, encontram-se as classes de modelos, as quais descrevem as Entidades. O diretório */util*, encontra-se a classe FirebaseUtil, a qual possui em sua implementação, dois padrões de projeto, o *Singleton*, para garantir que a classe tenha somente uma instância e fornecer um ponto global de acesso a ele, e a *Facade*, a fim de fornecer uma interface unificada para acessar os métodos.

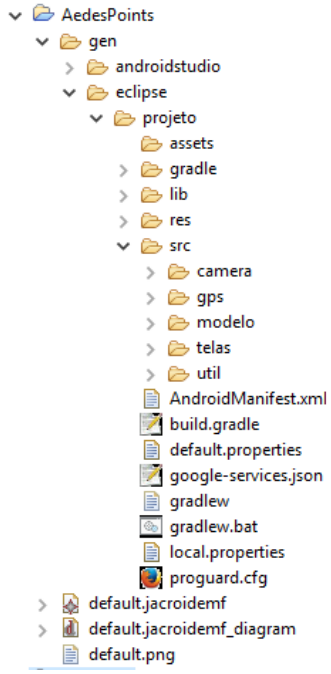


Figura 4.5: Estrutura do código gerado

Os pacotes *camera e gps* possuem as classes necessárias para acessarem ao sensores que lhes dizem respeito.

Por fim , temos as Classes de Visão e Controle que estão presentes no diretório, Telas. Para cada entidade é criada uma tela de listagem, de cadastro, de alteração e exclusão.

Na Figura 4.6 é ilustrado o diagrama de classes da aplicação gerada. No diagrama é possível visualizar como as classes estão interagindo. Observe que todas a classes que representam telas (cadastrar, alterar, listar) dependem da Classe `FirebaseUtil`. Essa classe é indispensável para a aplicação, pois concentra o processo de persistência dos dados (em conjunto com as classes de cadastro e alteração), autenticação de usuário (classe `authentication`), upload e download de imagens capturadas (realizadas pela classe `Camera`). Todos esses serviços são oferecidos pela plataforma Firebase.

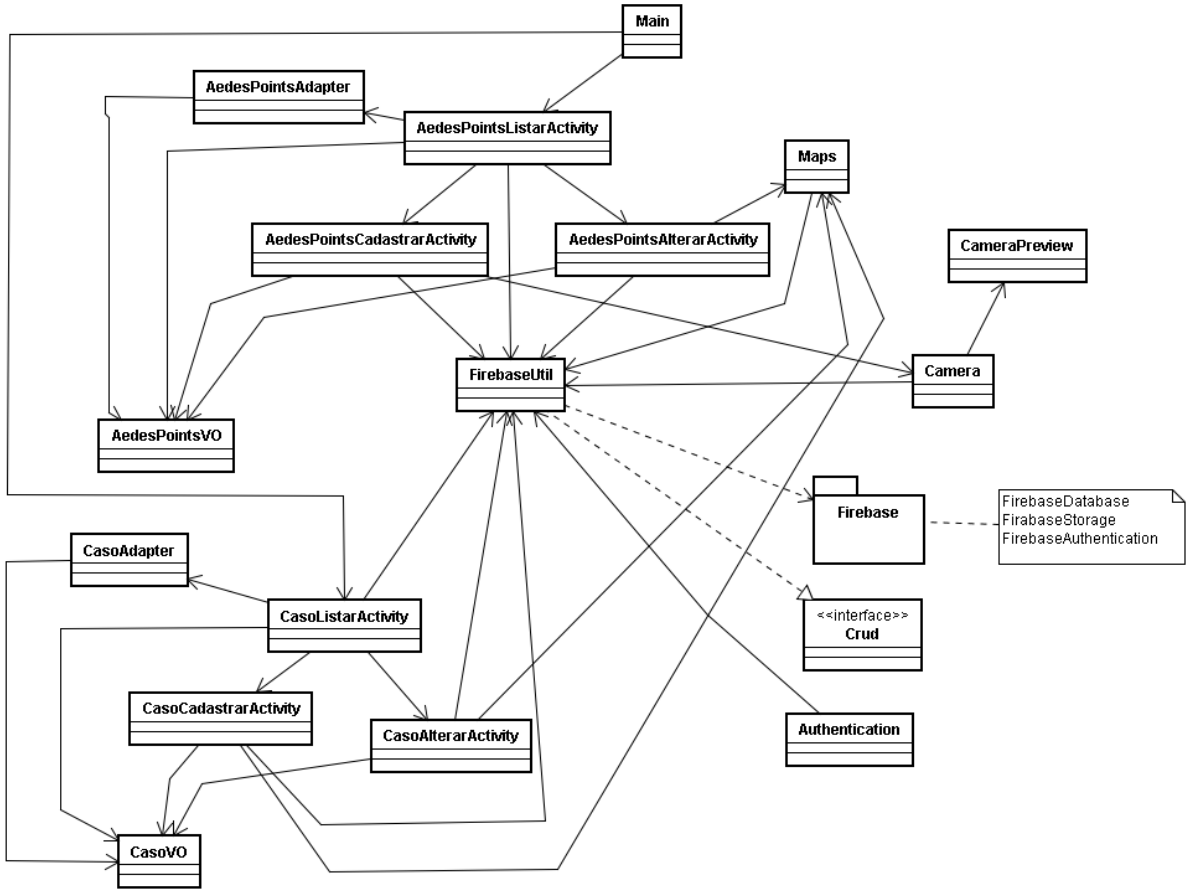


Figura 4.6: Diagrama de Classes do Aedes Points

4.1.3 Implantação da aplicação

Na fase de implantação da ferramenta é realizado o *deploy* do código fonte no dispositivo com o sistema operacional Android. Nesta fase também são realizados os testes de sistema, cujo aplicativo é executado com o intuito de verificar se os aspectos gerais do sistema estão sendo atendidos. Os requisitos do sistema são testados através da execução do aplicativo pelo engenheiro de software.

Com o projeto pronto o usuário deverá adicioná-lo a IDE Eclipse ou Android Studio e customizar da forma que achar conveniente. O *AedesPoints* foi adicionado ao Eclipse. Para importar o projeto no eclipse, o usuário deve ir no menu File->Import. Uma janela irá abrir, neste momento selecione a pasta Android e a opção "Existing Android Code in Workspace". Verifique se o projeto apareceu na listagem e aperte em "Finish". Aguarde enquanto o projeto é carregado.

Após o projeto importado, o arquivo `build.gradle` deve ser ser executado para que seja realizado o download das bibliotecas necessárias para o funcionamento da aplicação. As bibliotecas estarão disponíveis em "*/build/intermediates/exploded-aar*", importe todas no `Build Path`.

Dois arquivos devem ser alterados e são de inteira responsabilidade do usuário. O primeiro, é o arquivo `FirebaseUtil.java` dentro do pacote `util` que deve ser pre-

enchido com os dados da conta do usuário no Firebase. A conta pode ser criada no link https://console.firebase.google.com/. Abaixo é ilustrado o código fonte para alteração:

```
FirebaseOptions firebaseOptions = new FirebaseOptions.Builder()
.setDatabaseUrl("[YOUR_DATABASE]")
.setApiKey("[YOUR_APP_KEY]")
.setStorageBucket("[YOUR_STORAGE]")
.setApplicationId("[YOUR_APPLICATION_ID]").build();
```

O segundo é o arquivo `google_maps_api.xml`, localizado no diretório *res/values*. Nesse arquivo deve ser informado a sua KEY da API do Google Maps, a mesma pode ser criada no link https://console.developers.google.com/apis/credentials. Informe a sua KEY no espaço reservado no arquivo, conforme código abaixo:

```
<string name="google_maps_key"
templateMergeStrategy="preserve"
translatable="false» KEY </string>
```

Uma vez o projeto adicionado a IDE Eclipse e realizada as mudanças comentadas anteriormente, o usuário precisa apenas colocar o projeto pra rodar. Se houver um dispositivo conectado ao computador, o *deploy* será realizado nele, caso contrário, a IDE iniciará um Emulador, que possui todas as características necessárias para ver a aplicação em funcionamento. Essa ação pode demorar alguns instantes.

Na Figura 4.7 é possível observar as telas que foram geradas pelo aplicativo e já alimentado com algumas informações. A primeira tela da esquerda para a direita, é a tela boas vindas, já com usuário autenticado, representado pela classe `Authentication`. A segunda imagem apresenta Tela Inicial, gerada através da classe `Main`. A terceira temos a lista de pontos marcados (classe `AedesPointsListarActivity`). A quarta mostra o processo de inserção dos dados relativos a um caso de dengue notificado (classe `CasoCadastrarActivity`). A quinta demonstra a obtenção das coordenadas do ponto através do GPS (classe `Maps`). Na sexta é apresentado os pontos plotados no mapa, diferenciados por azul os pontos de possível proliferação de mosquito e vermelho, casos de notificados (classe `MapsPoints`). As telas apresentadas não tiveram nenhuma alteração no código fonte, é apresentado da maneira que a ferramenta gerou.

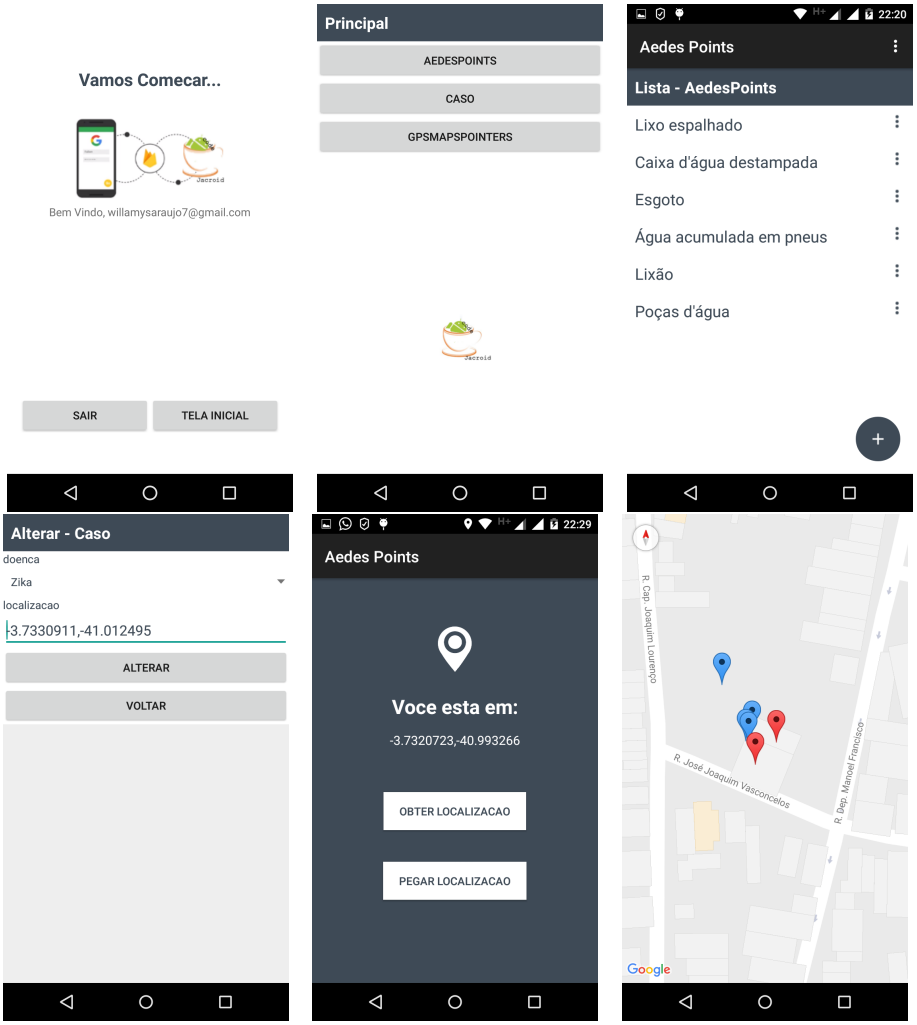


Figura 4.7: Telas do aplicativo gerado pela ferramenta

4.2 Discussão

Neste seção será realizada a descrição da avaliação da proposta, na busca de analisar o reúso de software entre aplicações desenvolvidas com o auxílio da ferramenta. A abordagem dessa avaliação tem inicio com a descrição da metodologia utilizada, objetivos e componentes necessários para reproduzir os experimentos. Em seguida, são descritos os resultados obtidos. E por fim, são apresentadas as conclusões acerca dos resultados.

4.2.1 Metodologia

Para a avaliação da proposta, a metodologia utilizada baseou-se na abordagem *Goal Question Metric* (GQM), cujo é um mecanismo para definir e avaliar um conjunto de metas operacionais, usando medição. Representa uma abordagem sistemática para adaptar e integrar objetivos com modelos de processos de software, produtos e perspectivas de qualidade de interesse, com base nas necessidades específicas do projeto e da organização (BASILI, 1992).

De acordo com a Figura 4.8, a abordagem selecionada consiste em quatro fases: Planejamento, Definição, Coleta de dados e Interpretação.

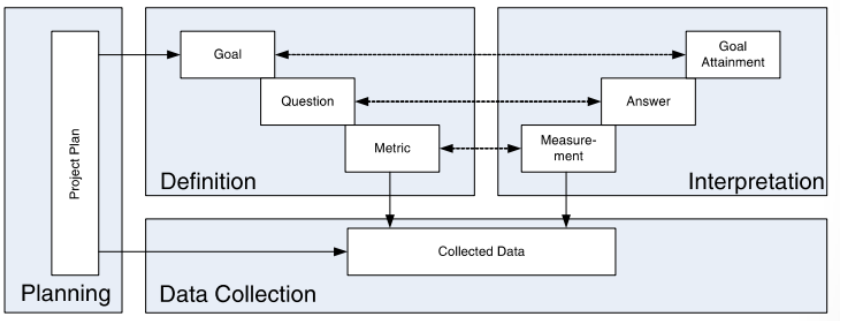


Figura 4.8: Fases do GQM (KOZIOLEK, 2008)

4.2.2 Planejamento

A fase inicial do planejamento é estabelecer a equipe GQM e identificar a área de melhoria desejada (por exemplo, confiabilidade, desempenho, reúso, etc). Posteriormente, é preciso caracterizar o produto ou processo a ser estudado.

Nesse sentido, a área escolhida para ser averiguada foi a de reúso de software no escopo da abordagem proposta e apresentada no capítulo 3.

4.2.3 Definição

Durante a fase de definição, são estabelecidas as metas de medição. Com base nos objetivos, são feitas perguntas, pedindo atributos de qualidade específicos e tornando certos aspectos das metas mais concretas. Para cada questão deve ser definida uma hipótese com uma resposta esperada. Posteriormente, as métricas são definidas para cada pergunta e verificadas em consistência e completude. Os resultados dessa fase são um plano GQM, um plano de medição e um plano de análise.

O objetivo definido é:

Avaliar o reúso nas aplicações geradas, no ponto de vista do engenheiro de software no contexto da abordagem proposta.

As métricas utilizadas são baseadas nas Métricas de Reúso Orientada a Estrutura de Software que estão divididas em duas categorias principais: métricas de reúso e uma avaliação de métricas de reutilização. A primeira avalia o reúso de ativos existentes, enquanto a segunda tem por objetivo avaliar, baseado em um conjunto de atributos de qualidade, o quanto um ativo é reusável (ALMEIDA et al., 2007).

Na Tabela 4.3 é apresentado o plano GQM em que são desenvolvidas as questões e métricas utilizadas.

Tabela 4.3: Plano GQM

Objetivo	Objeto de Estudo: Objetivo: Enfoque de Qualidade: Ponto de Vista: Contexto:	Aplicações geradas Avaliação Reúso Engenheiro de Software Abordagem
Questão Métrica	Q1 M1.1 M1.2	Qual a distribuição do reúso de forma geral no código gerado da aplicação? Percentual de Reúso (PR) (POULIN; CARUSO, 1993) Taxa de Reúso (TR) (DEVANBU et al., 1996)
Questão Métrica	Q2 M2.1 M2.2 M2.3	Para cada classe: Qual a distribuição do reúso entre as classes da aplicação gerada? Percentual de Reúso (PR) (POULIN; CARUSO, 1993) Taxa de Reúso (TR) (DEVANBU et al., 1996) Tamanho e Frequência de Reúso (TFR) (DEVANBU et al., 1996)

4.2.4 Coleta de Dados

A coleta dos dados foi auxiliada pela ferramenta *Eclipse Metrics*[3], versão 1.3.6. O *Metrics* fornece o cálculo de métricas e um complemento ao analisador de dependências para a plataforma Eclipse. Este permite calcular desvio padrão, ciclos nas dependências de pacotes e tipos e plotar gráficos.

A etapa de verificação de reúso demandou maior tempo, uma vez que a análise foi realizada arquivo por arquivo de forma manual. Isso proporcionou uma pesquisa mais aprofundada, porém bastante onerosa. A verificação manual só possível tendo em vista que os projetos avaliados eram pequenos, para projetos maiores ferramentas mais completas são indispensáveis.

Para auxiliar a consulta ao código fonte e realizar as comparações, outra ferramenta foi utilizada, o *Notepad++* [4], versão 5.6. Essa ferramenta possui um *plugin*, chamado de *compare*, que permite a visualização simultânea de arquivos, mostrando conteúdos que se repetem, linhas a mais e menos.

Os dados coletados alimentaram uma tabela que permitiu realizar os cálculos conforme as métricas estabelecidas no Plano GQM. Na Tabela 4.4 é descrita as fórmulas utilizadas para a realização dos cálculos.

[3]https:// sourceforge.net/projects/metrics/
[4]https://notepad-plus-plus.org/download/v7.3.3.html

Tabela 4.4: Métricas de reúso

Métrica	Definição
Percentual de Reúso (PR) (POULIN; CARUSO, 1993)	Proporção entre o número de linhas de código reusadas (LOC_REUS) e o número total de linhas de código (LOC_TOTAL) Fórmula: PR = LOC_REUS / LOC_TOTAL
Tamanho e Frequência de Reúso (TFR) (DEVANBU et al., 1996)	Semelhante a Frequência de Reúso, mas também considera o tamanho dos itens no número de linhas de código. TAM é a soma das quantidades de linhas novas de código geradas (LOC(new parts)); TAM_EXP considera a soma das quantidades de linhas das partes reusadas (LOC (Part)) vezes a quantidade de vezes que esta foi reusada (Refs(Part)). Fórmula: TAM = SUM (LOC (new parts)) TAM_EXP = SUM(LOC (part) * Refs(part)) TFR = (TAM_EXP – TAM) / TAM_EXP
Taxa de Reúso (TR) (DEVANBU et al., 1996)	Semelhante ao Percentual de Reúso, mas também considera itens que foram parcialmente alterados como reusados. LOC_REUS_TOTAL é número de linhas reusadas e parcialmente reusadas. LOC_TOTAL é a quantidade total de linhas. Fórmula: TR = LOC_REUS_TOTAL / LOC_TOTAL

4.2.5 Interpretação

Esta seção envolve a análise dos dados coletados através da avaliação de duas aplicações desenvolvidas com o uso da abordagem proposta. A primeira, aborda a aplicação que foi descrita no estudo de caso, a *AedesPoints*. A segunda foi a aplicação, *Lembre-me* que foi utilizada no capítulo anterior para explicar abordagem. Na Questão 1 e 2, as quais trazem as métricas de Percentual de Reúso e Taxa de Reúso(Q1) e Percentual de Reuso, Taxa de Reuso e Tamanho e Frequência de Reúso(Q2), ambas irão avaliar o reúso do código fonte de forma geral e por classe, através da relação *template->código gerado*. O *template* consiste em um código de linguagem de objeto com marcadores de posição, *as tags*, as quais contem expressões para obter os dados do modelo. A ferramenta realiza o *merge* entre o modelo e *template*, e temos o código gerado. No Apêndice C é possível observar o quão é reutilizado o código do *template* do componente GPS, para o seu código fonte. A Tabela 4.5 apresenta algumas informações básicas das aplicações avaliadas, como a quantidade de classes, a quantidade de linhas de código da aplicação geradas automaticamente.

A seguir são apresentadas as respostas das perguntas definidas no plano GQM.

Tabela 4.5: Aplicações avaliadas

Aplicação	Pacotes	Classes	LOC* Automatizada	LOC Manual
AedesPoints	5	18	2356	0
Lembre-me	5	13	1546	0

*Lines Of Code

Questão 1: Qual a distribuição do reúso de forma geral no código gerado da aplicação?

O gráfico da Figura 4.9 apresenta o Percentual de Reúso (PR) e Taxa de Reúso (TR) para as aplicações Aedes Points e Lembre-me. Analisando as duas aplicações em relação ao PR, verificamos um alto índice de reúso em que temos 81,79% (AedesPoints) e 91,91% (Lembre-me). Seguindo a análise, mas observando os valores de TR, o índice de reúso continua alto, chegando a 0,8561 (AedesPoints) e 0,9554. Os dados utilizados para a geração do gráfico da Figura 4.9 podem ser vistos na Tabela 4.6, os quais foram submetidos as fórmulas definidas na Tabela 4.4. Por exemplo, a classe *AedesPointsCadastrarActivity* possui 303 linhas de código, das quais 218 puderam ser reutilizadas pela utilização do *template classCadastrarActivity.egl*.

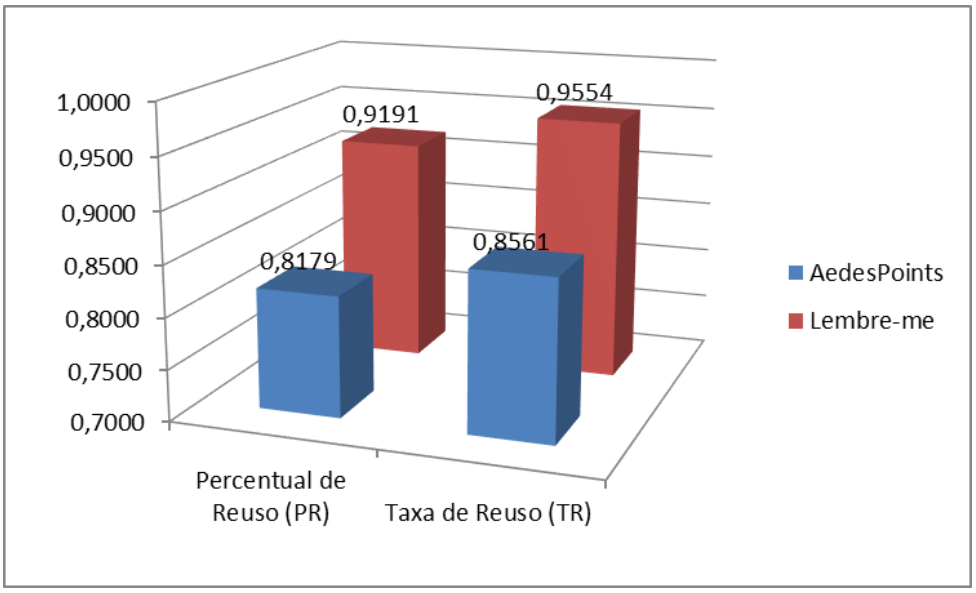


Figura 4.9: Análise do Reúso por aplicação

Tabela 4.6: Tabela PR e TR para aplicações analisadas

Aplicação	**Pacotes**	**Classes**	**LOC* TOTAL**	**LOC REUS**	**LOC REUS TOTAL**
AedesPoints	telas	AedesPointsCadastrarActivity	303	218	234
AedesPoints	telas	CasoCadastrarActivity	200	165	165
AedesPoints	telas	AedesPointsAlterarActivity	345	237	237
AedesPoints	telas	CasoAlterarActivity	225	183	183
AedesPoints	telas	AedesPointsListarActivity	150	131	150
AedesPoints	telas	CasoListarActivity	150	131	150
AedesPoints	telas	AedesPointsAdapter	50	35	50
AedesPoints	telas	CasoAdapter	50	35	50
AedesPoints	telas	Main	74	61	61
AedesPoints	telas	Authentication	154	154	154
AedesPoints	util	FirebaseUtil	161	161	161
AedesPoints	modelo	AedesPointsVO	57	18	18
AedesPoints	modelo	CasoVO	25	18	18
AedesPoints	modelo	ConnectionException	15	15	15
AedesPoints	gps	Maps	108	108	108
AedesPoints	gps	GpsMapsPointers	96	64	70
AedesPoints	camera	CameraActivity	149	149	149
AedesPoints	camera	CameraPreview	44	44	44
AedesPoints (Total)			2356	1927	2017
Lembre-me	telas	LembreteCadastrarActivity	252	223	239
Lembre-me	telas	LembreteAlterarActivity	283	244	244
Lembre-me	telas	LembreteListarActivity	150	131	150
Lembre-me	telas	LembreteAdapter	50	35	50
Lembre-me	telas	Main	66	62	62
Lembre-me	telas	Authentication	154	154	154
Lembre-me	util	FirebaseUtil	161	161	161
Lembre-me	modelo	LembreteVO	41	28	28
Lembre-me	modelo	ConnectionException	15	15	15
Lembre-me	gps	Maps	108	108	108
Lembre-me	gps	GpsMapsPointers	73	67	73
Lembre-me	camera	CameraActivity	149	149	149
Lembre-me	camera	CameraPreview	44	44	44
Lembre-me (Total)			1546	1421	1477

*Lines Of Code

Questão 2: Qual a distribuição do reúso entre as classes da aplicação gerada?

O gráfico da Figura 4.10 apresenta o Percentual de Reúso por Classe (PRC), Taxa de reúso por Classe (TRC) e Tamanho e Frequência de Reúso (TFR) das aplicações Aedes Points e Lembre-me. Analisando as duas aplicações em relação ao PRC, verificamos um alto índice de reúso em que temos 81,77% (AedesPoints) e 91,24% (Lembre-me). Seguindo a análise, mas observando os valores de TRC, o índice de reúso continua alto, chegando a 0,8715 (AedesPoints) e 0,9564. O Tamanho e Frequência de Reúso (TFR) para as classes da aplicação apresentaram uma ótima proporção de reúso, chegando a 0,8951 (Aedes Points) e 0,9457 (Lembre-me). Na Tabela 4.7 podemos ver os valores de PRC e TRC por classe, o valor apresentado no gráfico da Figura 4.10 é a média desses valores para aplicação completa. Já na Tabela 4.8 os valores apresentados serão submetidos ao cálculo do TFR.

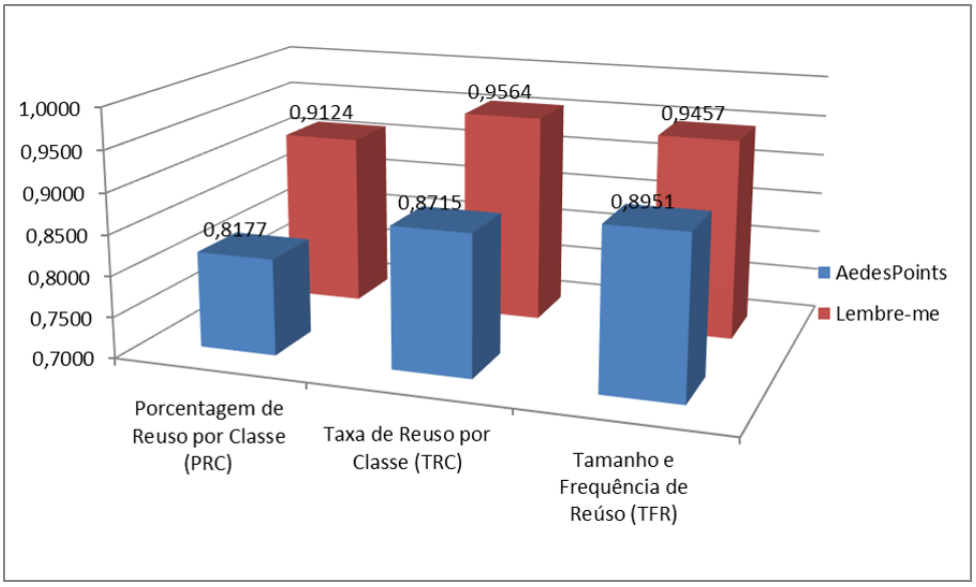


Figura 4.10: Análise do Reúso por classe

Tabela 4.7: Tabela PRC e TRC para aplicações analisadas

Aplicação	**Classes**	**LOC* TOTAL**	**LOC REUS**	**LOC REUS TOTAL**	**PRC**	**TRC**
AedesPoints	AedesPointsCadastrarActivity	303	218	234	0,7194	0,7722
AedesPoints	CasoCadastrarActivity	200	165	165	0,825	0,825
AedesPoints	AedesPointsAlterarActivity	345	237	237	0,6869	0,6869
AedesPoints	CasoAlterarActivity	225	183	183	0,8133	0,8133
AedesPoints	AedesPointsListarActivity	150	131	150	0,8733	1
AedesPoints	CasoListarActivity	150	131	150	0,87333	1
AedesPoints	AedesPointsAdapter	50	35	50	0,7	1
AedesPoints	CasoAdapter	50	35	50	0,7	1
AedesPoints	Main	74	61	61	0,8243	0,8243
AedesPoints	Authentication	154	154	154	1	1
AedesPoints	FirebaseUtil	161	161	161	1	1
AedesPoints	AedesPointsVO	57	18	18	0,3157	0,3157
AedesPoints	CasoVO	25	18	18	0,72	0,72
AedesPoints	ConnectionException	15	15	15	1	1
AedesPoints	Maps	108	108	108	1	1
AedesPoints	GpsMapsPointers	96	64	70	0,6666	0,7291
AedesPoints	CameraActivity	149	149	149	1	1
AedesPoints	CameraPreview	44	44	44	1	1
AedesPoints					**0,8177**	0,**8715**
Lembre-me	LembreteCadastrarActivity	252	223	239	0,8849	0,9484
Lembre-me	LembreteAlterarActivity	283	244	244	0,8621	0,8621
Lembre-me	LembreteListarActivity	150	131	150	0,8733	1
Lembre-me	LembreteAdapter	50	35	50	0,7	1
Lembre-me	Main	66	62	62	0,9393	0,9393
Lembre-me	Authentication	154	154	154	1	1
Lembre-me	FirebaseUtil	161	161	161	1	1
Lembre-me	LembreteVO	41	28	28	0,6829	0,6829
Lembre-me	ConnectionException	15	15	15	1	1
Lembre-me	Maps	108	108	108	1	1
Lembre-me	GpsMapsPointers	73	67	73	0,9178	1
Lembre-me	CameraActivity	149	149	149	1	1
Lembre-me	CameraPreview	44	44	44	1	1
Lembre-me					**0,9124**	**0,9564**

*Lines Of Code

Tabela 4.8: Tabela TFR para aplicações analisadas

Aplicação	Classes	Refs	LOC x Refs	LOC New Parts
AedesPoints	AedesPointsCadastrarActivity	1	303	85
AedesPoints	CasoCadastrarActivity	1	200	35
AedesPoints	AedesPointsAlterarActivity	1	345	108
AedesPoints	CasoAlterarActivity	1	225	42
AedesPoints	AedesPointsListarActivity	1	150	19
AedesPoints	CasoListarActivity	1	150	19
AedesPoints	AedesPointsAdapter	1	50	15
AedesPoints	CasoAdapter	1	50	15
AedesPoints	Main	0	0	13
AedesPoints	Authentication	0	0	0
AedesPoints	FirebaseUtil	10	1610	0
AedesPoints	AedesPointsVO	4	228	39
AedesPoints	CasoVO	4	100	7
AedesPoints	ConnectionException	0	0	0
AedesPoints	Maps	4	432	0
AedesPoints	GpsMapsPointers	1	96	32
AedesPoints	CameraActivity	1	149	0
AedesPoints	CameraPreview	0	0	0
AedesPoints	(TAM_EXP e TAM)		4088	429
Lembre-me	LembreteCadastrarActivity	1	252	29
Lembre-me	LembreteAlterarActivity	1	283	39
Lembre-me	LembreteListarActivity	1	150	19
Lembre-me	LembreteAdapter	1	50	15
Lembre-me	Main	0	0	4
Lembre-me	Authentication	0	0	0
Lembre-me	FirebaseUtil	6	966	0
Lembre-me	LembreteVO	4	164	13
Lembre-me	ConnectionException	0	0	0
Lembre-me	Maps	2	216	0
Lembre-me	GpsMapsPointers	1	73	6
Lembre-me	CameraActivity	1	149	0
Lembre-me	CameraPreview	0	0	0
Lembre-me	(TAM_EXP e TAM)		2303	125

*Lines Of Code

4.2.6 Análise dos Resultados

Esses valores apresentados como resposta das questões levantadas anteriormente, são reflexo do escopo/domínio bem definido, restrições impostas pelo desenvolvimento orientado a modelos com a utilização da ferramenta de modelagem e componentes bem elaborados. É certo que abordagem possui seus bônus e ônus, os quais são apresentados a seguir.

Como benefícios pode-se destacar:

i. Automatização: O uso de um sistema transformacional para a geração do código da aplicação baseado em *templates*, automatiza um estágio importante no desenvolvimento de software: a implementação do projeto do sistema. Outro ponto que a automatização agregou ao projeto do sistema foi no ganho de produtividade.

ii. Modelagem de requisitos: As atividades de especificação de requisitos e modelagem do domínio, presentes na etapa de identificação dos objetos do domínio, são aceleradas devido utilização de uma ferramenta de modelagem desenvolvida.

iii. reúso: os objetos identificados e extraídos dos domínio no qual deseja iniciar um desenvolvimento podem ser implementados de forma a serem reutilizados posteriormente. E de acordo com a análise tivemos níveis acima de 90% de reúso tanto na aplicação como um todo, quanto para as classes. Com modificações apenas nos *templates*, a ferramenta pode ser reutilizada e adaptada para outras linguagens de programação para dispositivos móveis como o iOS.

Além disso, algumas desvantagens puderam ser observadas:

i. Construção da ferramenta de modelagem: Visto que a abordagem proposta usa um sistema para modelagem de domínio e transformação de modelos para auxiliar o Engenheiro de Software, admite-se que devem existir ferramentas para identificar os objetos do domínio e realizar as transformações seguindo padrões e restrições. Entretanto, primeiramente estas ferramentas devem ser construídas, o que requer um esforço e um grande conhecimento sobre as linguagens envolvidas, visto que além das linguagens do próprio sistema transformacional para especificar as gramáticas e os transformadores, também é necessário ter um domínio sobre as linguagens de programação e de modelagem (Java, XML, por exemplo). O tempo gasto na construção dos transformadores também é um fator crítico.

ii. Código fonte gerado: O código fonte gerado pela abordagem talvez precise de uma refatoração para melhorar a sua eficiência, mesmo com o padrões de projeto que foram utilizados.

Ameaças também foram elencadas:

i. Má especificação: Podem levar a sistemas instáveis, e diminuir significativamente o reúso dos componentes da aplicação, bem como erros inesperados.

ii. Usar abordagem para outro domínio: Devido a abordagem possuir um domínio específico, a adequação desta para outros, pode desencadear problemas na implementação e funcionamento das aplicações.

4.3 Considerações Finais

Este capítulo apresentou um estudo de caso de uma aplicação desenvolvida a partir da abordagem proposta. Foram realizadas as análises e interpretação dos dados que afirmaram o nível de reúso satisfatório, bem como ganhos em automatização/produtividade, modelagem de requisitos. Uma desvantagem observada se refere ao tempo para o desenvolvimento da

ferramenta que auxiliou a abordagem, bem como o código gerado pode ser refatorado para melhorar a sua eficiência. Ameaças também foram levantas, como a má especificação pelo engenheiro de software e utilizar a abordagem para outros domínios, as quais podem trazer problemas a implementação e funcionamento das aplicações. As conclusões serão apresentadas no próximo capítulo.

5

Conclusão

Esta dissertação apresenta uma abordagem para o desenvolvimento de aplicações no domínio pessoas como sensores para *smartphones*, a criação de uma ferramenta que aplica a abordagem definida e um estudo sobre a reutilização de software auxiliada pelo desenvolvimento orientado a modelos. Diversas tecnologias foram estudadas e aplicadas, como o Desenvolvimento orientado a modelos que contribuiu para a construção da ferramenta de modelagem, Transformações de Software tendo como base *templates*, e o Desenvolvimento Orientado a Objetos aplicados na implementação das aplicações.

5.1 Contribuições

A principal contribuição deste trabalho é a definição de uma abordagem, que permite o desenvolvimento de aplicações no domínio de pessoas como sensores para dispositivos móveis. A abordagem é baseada no desenvolvimento orientado a modelos, e é utilizada na obtenção de um novo sistema orientado a objetos para funcionamento na plataforma android, priorizando o reúso, a produtividade e automatização.

No contexto da identificação de objetos do domínio, o desenvolvimento de uma ferramenta para modelagem dentro do domínio de pessoas como sensores pode ser considerada como contribuição, permitindo que o engenheiro de software ao especificar os objetos que necessitava para a aplicação, também estivesse implementando-a, abstraindo todo o processo de codificação.

Esta pesquisa também contribuiu com a especificação de regras de transformações baseadas em *templates* para a geração do código fonte das aplicações, as quais podem ser utilizadas pelo Engenheiro de Software na construção de ferramentas ou adaptação desta em outros domínios.

Outra contribuição desta pesquisa vem da integração de diferentes técnicas e mecanismos para apoiar a abordagem, automatizando parte das tarefas do Engenheiro de Software.

A pesquisa também demonstrou o potencial do desenvolvimento orientado a modelos, que aliadas ao reúso, apresentaram bons índices de reutilização de software e automatização da implementação das aplicações.

As aplicações implementadas pela abordagem apresentam estrutura completa com integração a banco de dados em tempo real e armazenamento na nuvem, com poucas interações com código fonte.

Por fim também pode ser considerada como contribuição desta pesquisa o uso combinado de uma ferramenta de modelagem com a IDE Eclipse, para apoiar a implementação de aplicações que vão desde a especificação/modelagem dos requisitos, até a aplicação implantada no dispositivo na plataforma android.

5.2 Trabalhos Futuros

Outra forma de contribuição importante em um trabalho de pesquisa é a geração de trabalhos futuros. Com relação à presente dissertação, foram elencadas algumas possibilidades.

Em relação a abordagem, a mesma utiliza-se da infraestrutura da IDE Eclipse para o seu funcionamento e analisando as tendências das ferramentas de desenvolvimento de aplicações na plataforma Android sendo guiada para o uso do Android Studio, abre a oportunidade de integrar esse abordagem como forma de plugin ao Android Studio. Existe também a necessidade de uma nova etapa, em que uma metodologia de testes possa ser integrada, a fim de garantir que o resultado do processo de desenvolvimento das aplicações esteja de acordo com o esperado. E ainda é possível que a ferramenta desenvolvida para a abordagem possa ser adaptada para outros domínios de aplicação, como também para outras plataformas como a iOS [1], ou até mesmo, o Arduíno [2].

Com o advento das cidades inteligentes, ferramentas ou abordagens que propiciem o desenvolvimento de aplicações para esse domínio são de suma importância, uma vez que esta está totalmente ligada a *crowd sensing* ou pessoas como sensores. Nesse sentido, a abordagem poderia se adaptada para permitir criar aplicações que possam medir a inteligência de uma cidade permitindo construir uma estrutura de índices para revelar o quão melhorou a qualidade urbana. Ou permitir aplicações que atuem no gerenciamento de desastres e/ou políticas públicas, em que a atuação coletiva seria um excelente forma de contribuir e, ao mesmo tempo, receber informações úteis de acordo com o cenário atual.

[1] https://www.apple.com/br/ios/ios-10/
[2] https://www.arduino.cc/

Referências

AAZAM, M. et al. Cloud of Things: integrating internet of things and cloud computing and the issues involved. In: APPLIED SCIENCES AND TECHNOLOGY (IBCAST), 2014 11TH INTERNATIONAL BHURBAN CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.414–419.

ALLIANCE, Z. ZigBee 3.0: the foundation for the internet of things. , [S.l.], 2015.

ALMEIDA, E. S. et al. CRUISE: component reuse in software engineering. , [S.l.], 2007.

AMBLER, S. W. **Modelagem ágil**: práticas eficazes para a programação extrema e o processo unificado. [S.l.]: Bookman Editora, 2009.

ANDROID, D. Android, the world's most popular mobile platform. , [S.l.], 2016.

ANDROID, D. Platform Architecture. , [S.l.], 2016.

ANNIE, A. App Annie 2016 Retrospective. **App Annie**, [S.l.], 2017.

APACHE, S. F. Apache Velocity Project. , [S.l.], 2011.

ARNOLDUS, J.; BIJPOST, J.; BRAND, M. van den. Repleo: a syntax-safe template engine. In: INTERNATIONAL CONFERENCE ON GENERATIVE PROGRAMMING AND COMPONENT ENGINEERING, 6., New York, NY, USA. **Proceedings...** ACM, 2007. p.25–32. (GPCE '07).

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: a survey. **Computer Networks**, [S.l.], v.54, n.15, p.2787 – 2805, 2010.

BANDYOPADHYAY, S. et al. Role of middleware for internet of things: a study. **International Journal of Computer Science and Engineering Survey**, [S.l.], v.2, n.3, p.94–105, 2011.

BASILI, V. R. **Software modeling and measurement**: the goal/question/metric paradigm. [S.l.: s.n.], 1992.

BLOIS, A. P. T. B. **Uma Abordagem de projeto arquitetural baseado em componentes no contexto de Engenharia de Domínio**. 2006. Tese (Doutorado em Ciência da Computação) — UNIVERSIDADE FEDERAL DO RIO DE JANEIRO.

BOSCH, J. **Design and use of software architectures**: adopting and evolving a product-line approach. [S.l.]: Pearson Education, 2000.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-driven software engineering in practice. **Synthesis Lectures on Software Engineering**, [S.l.], v.1, n.1, p.1–182, 2012.

BROWN, A. **Large-scale, Component-based Development**. [S.l.]: Prentice Hall PTR, 2000. (Object and component technology series).

CAMPBELL, A. T. et al. People-centric urban sensing. In: WIRELESS INTERNET, 2. **Proceedings...** [S.l.: s.n.], 2006. p.18.

COULOURIS, G. et al. **Sistemas Distribuídos-**: conceitos e projeto. [S.l.]: Bookman Editora, 2013.

CZARNECKI, K. Generative programming: principles and techniques of software engineering based on automated configuration and fragment-based component models. , [S.l.], 1998.

CZARNECKI, K. et al. Generative programming. **Edited by G. Goos, J. Hartmanis, and J. van Leeuwen**, [S.l.], v.15, 2000.

CZARNECKI, K. et al. Generative programming for embedded software: an industrial experience report. In: INTERNATIONAL CONFERENCE ON GENERATIVE PROGRAMMING AND COMPONENT ENGINEERING. **Anais...** [S.l.: s.n.], 2002. p.156–172.

DEVANBU, P. et al. Analytical and empirical evaluation of software reuse metrics. In: SOFTWARE ENGINEERING, 18. **Proceedings...** [S.l.: s.n.], 1996. p.189–199.

ECLIPSE, E. Eclipse modeling framework. **URL http://www. eclipse. org/modeling/emf**, [S.l.], 2006.

FIREBASE. Firebase:app success made simple. , [S.l.], 2017.

FOWLER, M. **UML Essencial**: um breve guia para linguagem padrão. [S.l.]: Bookman Editora, 2014.

GIACHETTI, G.; MARÍN, B.; PASTOR, O. Using uml as a domain-specific modeling language: a proposal for automatic generation of uml profiles. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING. **Anais...** [S.l.: s.n.], 2009. p.110–124.

GIMENES, I. M. d. S.; HUZITA, E. H. M. Desenvolvimento baseado em componentes: conceitos e técnicas. **Rio de Janeiro: Ciência Moderna**, [S.l.], 2005.

GUBBI, J. et al. Internet of Things (IoT): a vision, architectural elements, and future directions. **Future generation computer systems**, [S.l.], v.29, n.7, p.1645–1660, 2013.

GUO, B. et al. From participatory sensing to mobile crowd sensing. In: PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS (PERCOM WORKSHOPS), 2014 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2014. p.593–598.

HERRINGTON, J. **Code generation in action**. [S.l.]: Manning Publications Co., 2003.

JABER, M. et al. A high-level modeling language for the efficient design, implementation, and testing of Android applications. **International Journal on Software Tools for Technology Transfer**, [S.l.], p.1–18, 2016.

JARA, A. J. et al. Mobile Digcovery: discovering and interacting with the world through the internet of things. **Personal Ubiquitous Comput.**, London, UK, UK, v.18, n.2, p.323–338, Feb. 2014.

KANG, K. C. et al. **Feature-oriented domain analysis (FODA) feasibility study**. [S.l.]: DTIC Document, 1990.

KIM, J.; LEE, J.-W. OpenIoT: an open service framework for the internet of things. In: INTERNET OF THINGS (WF-IOT), 2014 IEEE WORLD FORUM ON. **Anais...** [S.l.: s.n.], 2014. p.89–93.

KLEPPE, A. G.; WARMER, J. B.; BAST, W. **MDA explained**: the model driven architecture: practice and promise. [S.l.]: Addison-Wesley Professional, 2003.

KOLOVOS, D. et al. **The Epsilon Book**. [S.l.]: Eclipse Public Licence, 2016.

KOZIOLEK, H. Goal, question, metric. In: **Dependability metrics**. [S.l.]: Springer, 2008. p.39–42.

LARDINOIS, F. Google Acquires Firebase To Help Developers Build Better Real-Time Apps. , [S.l.], 2014.

LEE, I.; LEE, K. The Internet of Things (IoT): applications, investments, and challenges for enterprises. **Business Horizons**, [S.l.], v.58, n.4, p.431–440, 2015.

LUCRÉDIO, D. Uma abordagem orientada a modelos para reutilização de software. **INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO UNIVERSIDADE DE SÃO PAULO**, [S.l.], p.37, 2009.

MA, H.; ZHAO, D.; YUAN, P. Opportunities in mobile crowd sensing. **IEEE Communications Magazine**, [S.l.], v.52, n.8, p.29–35, 2014.

MEIRELLES, F. S. Uso da TI nas Empresas - Panorama e Indicadores. **27ª Pesquisa: Administração e Uso da TI nas Empresas**, [S.l.], 2016.

MUSCHKO, B. **Gradle in action**. [S.l.]: Manning, 2014.

OLIVEIRA, C. K. S. **Georreferenciamento do Desequilíbrio Ambiental e à Prevalência da Dengue no Município Juazeiro/BA**. 2015. Dissertação — UNIVASF.

PAULOS, E. Designing for Doubt Citizen Science and the Challenge of Change. In: ENGAGING DATA: FIRST INTERNATIONAL FORUM ON THE APPLICATION AND MANAGEMENT OF PERSONAL ELECTRONIC INFORMATION. 2009: MIT. **Anais...** [S.l.: s.n.], 2009.

POULIN, J. S.; CARUSO, J. M. A reuse metrics and return on investment model. In: SOFTWARE REUSABILITY, 1993. PROCEEDINGS ADVANCES IN SOFTWARE REUSE., SELECTED PAPERS FROM THE SECOND INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 1993. p.152–166.

PRAMUDIANTO, F. et al. IoT Link: an internet of things prototyping toolkit. In: UBIQUITOUS INTELLIGENCE AND COMPUTING, 2014 IEEE 11TH INTL CONF ON AND IEEE 11TH INTL CONF ON AND AUTONOMIC AND TRUSTED COMPUTING, AND IEEE 14TH INTL CONF ON SCALABLE COMPUTING AND COMMUNICATIONS AND ITS ASSOCIATED WORKSHOPS (UTC-ATC-SCALCOM). **Anais...** [S.l.: s.n.], 2014. p.1–9.

RATTI, C.; NABIAN, N. The city to come. **Innovation: Perspectives for the 21st Century, BBVA**, [S.l.], 2010.

SALIM, F.; HAQUE, U. Urban computing in the wild: a survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and internet of things. **International Journal of Human-Computer Studies**, [S.l.], v.81, p.31 – 48, 2015. Transdisciplinary Approaches to Urban Computing.

SILVA, W. M. d. Go! SIP: um framework de privacidade para cidades inteligentes baseado em pessoas como sensores. , [S.l.], 2014.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Pearson Education, 2011.

SUNDMAEKER, H. et al. Vision and challenges for realising the Internet of Things. **Cluster of European Research Projects on the Internet of Things, European Commision**, [S.l.], 2010.

SZYPERSKI, C.; GRUNTZ, D.; MURER, S. Component software: beyond object-oriented programming. **Addison-Wesley**, [S.l.], 1998.

THIYAGARAJAN, M.; RAVEENDRA, C. Integration in the physical world in IoT using android mobile application. In: GREEN COMPUTING AND INTERNET OF THINGS (ICGCIOT), 2015 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2015. p.790–795.

THÜM, T. et al. FeatureIDE: an extensible framework for feature-oriented software development. **Science of Computer Programming**, [S.l.], v.79, p.70–85, 2014.

VERMESAN, O.; FRIESS, P. **Internet of Things-From research and innovation to Market Deployment**. [S.l.]: River Publishers, 2014.

WANT, R. An introduction to RFID technology. **IEEE pervasive computing**, [S.l.], v.5, n.1, p.25–33, 2006.

WARMER, J. B.; KLEPPE, A. G. **The object constraint language**: getting your models ready for mda. [S.l.]: Addison-Wesley Professional, 2003.

XDOCLET, T. XDoclet, Attribute-Oriented Programming. , [S.l.], 2005.

YUSUF, L.; CHESSEL, M.; GARDNER, T. Implement model-driven development to increase the business value of your IT system. **Retrieved January**, [S.l.], v.29, p.2008, 2006.

Apêndice

A

Etapa 3 - Transformação do Metamodelo de Integração para perfil UML final

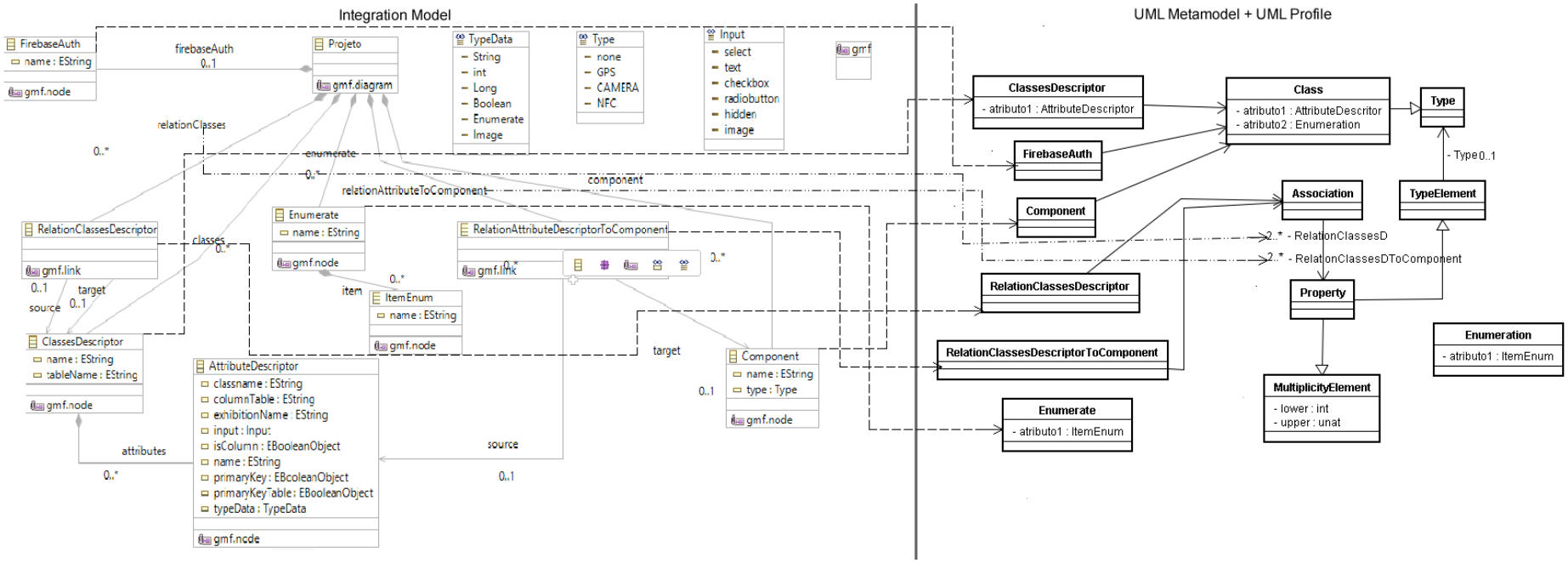


Figura A.1: Transformação do Metamodelo de Integração para o perfil UML final

B

Casos de Uso da aplicação AedesPoints

Tabela B.1: Caso de Uso da aplicação "AedesPoints"

Nome: **UC01 - Cadastrar AedesPoints**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar AedesPoints 2. O usuário clica no botão Cadastrar AedesPoints. 3. O usuário preenche os dados e clica em salvar. 4. Aplicação valida os dados e emite mensagem de sucesso. 5. Aplicação retorna a tela Listar AedesPoints para o usuário.
Fluxo Alternativo 1: 4. Os dados informados são inválidos. 4.1 Aplicação emite mensagem para corrigir os dados. Fluxo Alternativo 2: 3. O usuário preenche os dados, mas deve informar a localização. 3.1 UC08 Fluxo Alternativo 3: 3. O usuário preenche os dados, mas deve informar uma foto. 3.1 UC07

Tabela B.2: Caso de Uso da aplicação "AedesPoints"

Nome: **UC02 - Atualizar AedesPoints**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar AedesPoints 2. O usuário deve selecionar o AedesPoints desejado e seleciona a opção editar. 3. O usuário alterar os dados e clica em salvar. 4. Aplicação valida os dados e emite mensagem de sucesso. 5. Aplicação retorna a tela Listar AedesPoints para o usuário.
Fluxo Alternativo 1: 4. Os dados informados são inválidos. 4.1 Aplicação emite mensagem para corrigir os dados. Fluxo Alternativo 2: 3. O usuário preenche os dados, mas deve informar a localização. 3.1 UC08 Fluxo Alternativo 3: 3. O usuário preenche os dados, mas deve informar uma foto. 3.1 UC07

Tabela B.3: Caso de Uso da aplicação "AedesPoints"

Nome: **UC03 - Listar AedesPoints**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar AedesPoints 2. Aplicação retorna a Lista de itens cadastrados para o usuário.
Fluxo Alternativo: Não há.

Tabela B.4: Caso de Uso da aplicação "AedesPoints"

Nome: **UC05 - Atualizar Casos**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar Casos 2. O usuário deve selecionar o Caso desejado e seleciona a opção editar. 3. O usuário preenche os dados e clica em salvar. 4. Aplicação valida os dados e emite mensagem de sucesso. 5. Aplicação retorna a tela Listar Casos para o usuário.
Fluxo Alternativo 1: 4. Os dados informados são inválidos. 4.1 Aplicação emite mensagem para corrigir os dados. Fluxo Alternativo 2: 3. O usuário preenche os dados, mas deve informar a localização. 3.1 UC08

Tabela B.5: Caso de Uso da aplicação "AedesPoints"

Nome: **UC06 - Listar Casos**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela Listar Casos 2. Aplicação retorna a lista de itens cadastrados para o usuário.
Fluxo Alternativo: Não há.

Tabela B.6: Caso de Uso da aplicação "AedesPoints"

Nome: **UC07 - Camera**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário clica no campo foto 2. A aplicação fornece a opção de usar foto da galeria 2.1. Selecione foto da galeria.
Fluxo Alternativo: 2. A aplicação fornece a opção de capturar nova foto 2.1. Capture a nova foto

Tabela B.7: Caso de Uso da aplicação "AedesPoints"

Nome: **UC09 - Mapear Casos e AedesPoints**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário acessa a tela GPSMapsPointers. 2. Aplicação retorna um mapa com as marcações de Casos e AedesPoints, cada qual com uma cor associada.
Fluxo Alternativo: 3. Nenhuma marcação é mostrada.

Tabela B.8: Caso de Uso da aplicação "AedesPoints"

Nome: **UC10 - Efetuar Login**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário deve informar um email e senha. 2. O usuário clica no botão entrar. 3. Aplicação valida os dados e emite mensagem de sucesso. 4. Aplicação retorna a tela inicial para o usuário.
Fluxo Alternativo 1: 3. Os dados informados são inválidos. 3.1 Aplicação emite mensagem para corrigir os dados.

Tabela B.9: Caso de Uso da aplicação "AedesPoints"

Nome: **UC11 - Efetuar Logout**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário deve acessar o menu na tela inicial 2. O usuário clica na opção Sign Out. 3. Aplicação retornará o usuário para a tela de login.
Fluxo Alternativo: Não há.

Tabela B.10: Caso de Uso da aplicação "AedesPoints"

Nome: **UC12 - Realizar Sign in**
Ator: **Usuário Final**
Fluxo Normal: 1. O usuário deve informar um email e senha. 2. O usuário clica no botão nova conta. 3. Aplicação valida os dados e emite mensagem de sucesso. 4. Aplicação retorna a tela inicial para o usuário.
Fluxo Alternativo 1: 4. Os dados informados são inválidos. 4.1 Aplicação emite mensagem para corrigir os dados.

C

Componente GPS

```
1  package gps;
2
3  import com.google.android.gms.common.ConnectionResult;
4  import com.google.android.gms.common.api.GoogleApiClient;
5  import com.google.android.gms.location.LocationListener;
6  import com.google.android.gms.location.LocationRequest;
7  import com.google.android.gms.location.LocationServices;
8  import com.src.android.R;
9
10 import android.app.Activity;
11 import android.content.Context;
12 import android.content.Intent;
13 import android.location.Location;
14 import android.location.LocationManager;
15 import android.os.Bundle;
16 import android.provider.Settings;
17 import android.util.Log;
18 import android.view.View;
19 import android.widget.Button;
20 import android.widget.TextView;
21 import android.widget.Toast;
22
23  /**@author Willamys Araujo
24  **Generate for Jacroid**/
25
26 public class [%=component.toString().firstToUpperCase()%] extends
      Activity
27 implements GoogleApiClient.ConnectionCallbacks,
28 GoogleApiClient.OnConnectionFailedListener, LocationListener{
29
```

```
30      private TextView tvCoordinate;
31      private Location location;
32      private LocationRequest locationRequest;
33      private Button btnShowLocation;
34      private Button btnGetLocation;
35      private GoogleApiClient mGoogleApiClient;
36
37      @Override
38      protected void onCreate(Bundle savedInstanceState) {
39          super.onCreate(savedInstanceState);
40          setContentView(R.layout.gps);
41
42          tvCoordinate = (TextView) findViewById(R.id.TvCoordinate);
43          btnShowLocation = (Button) findViewById(R.id.
                ButtonShowLocation);
44          btnGetLocation = (Button) findViewById(R.id.ButtonGetLocation
                );
45
46          btnGetLocation.setOnClickListener(new View.OnClickListener()
                {
47              @Override
48              public void onClick(View v) {
49                  onBackPressed();
50              }
51          });
52
53          btnShowLocation.setOnClickListener(new View.OnClickListener()
                {
54        @Override
55        public void onClick(View v) {
56                  LocationManager manager = (LocationManager)
                        getSystemService( Context.LOCATION_SERVICE );
57                  boolean isOn = manager.isProviderEnabled(
                        LocationManager.GPS_PROVIDER);
58                  if(isOn) {
59                      callConnection();
60                  }else{
61                      Toast.makeText(Maps.this, "Ative a Localizacao!",
                            Toast.LENGTH_LONG).show();
62                      Intent intent = new Intent(Settings.
                            ACTION_LOCATION_SOURCE_SETTINGS);
```

```
                    startActivity(intent);
                }
        }
    });
        callConnection();
    }

    private synchronized void callConnection(){
        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .addOnConnectionFailedListener(this)
                .addConnectionCallbacks(this)
                .addApi(LocationServices.API)
                .build();
        mGoogleApiClient.connect();
    }

    private void initLocationRequest(){
        locationRequest = new LocationRequest();
        locationRequest.setInterval(5000);
        locationRequest.setFastestInterval(2000);
        locationRequest.setPriority(locationRequest.
           PRIORITY_HIGH_ACCURACY);
    }

    private void startLocationUpdate(){
        initLocationRequest();
        LocationServices.FusedLocationApi.requestLocationUpdates(
           mGoogleApiClient, locationRequest,Maps.this);
    }
    private void stopLocationUpdate(){
        LocationServices.FusedLocationApi.removeLocationUpdates(
           mGoogleApiClient,Maps.this);
    }

    @Override
    public void onConnected(Bundle bundle) {
        Log.i("LOG", "onConnected(" + bundle + ")");

       location = LocationServices.FusedLocationApi.getLastLocation(
          mGoogleApiClient);

```

```
100         if(location != null){
101             Log.i("LOG", "latitude: "+ location.getLatitude());
102             Log.i("LOG", "longitude: "+ location.getLongitude());
103             tvCoordinate.setText(location.getLatitude()+","+ location
                    .getLongitude());
104         }
105         startLocationUpdate();
106     }
107
108     @Override
109     public void onConnectionSuspended(int i) {
110         Log.i("LOG", "onConnectionSuspended(" + i + ")");
111     }
112     @Override
113     public void onConnectionFailed(ConnectionResult connectionResult)
            {
114         Log.i("LOG", "onConnectionFailed("+connectionResult+")");
115     }
116
117     @Override
118     public void onBackPressed(){
119         Intent data = new Intent();
120         data.putExtra("gps", tvCoordinate.getText());
121         setResult(2,data);
122         finish();
123         stopLocationUpdate();
124     }
125
126     @Override
127     public void onLocationChanged(Location location) {
128         tvCoordinate.setText(location.getLatitude()+","+ location.
                getLongitude());
129     }
130 }
```

Listing C.1: Código template *classGPS.egl*

```
1 package gps;
2
3 import com.google.android.gms.common.ConnectionResult;
4 import com.google.android.gms.common.api.GoogleApiClient;
5 import com.google.android.gms.location.LocationListener;
6 import com.google.android.gms.location.LocationRequest;
```

```
7  import com.google.android.gms.location.LocationServices;
8  import com.src.android.R;
9
10 import android.app.Activity;
11 import android.content.Context;
12 import android.content.Intent;
13 import android.location.Location;
14 import android.location.LocationManager;
15 import android.os.Bundle;
16 import android.provider.Settings;
17 import android.util.Log;
18 import android.view.View;
19 import android.widget.Button;
20 import android.widget.TextView;
21 import android.widget.Toast;
22
23  /**@author Willamys Araujo
24  **Generate for Jacroid**/
25
26 public class Maps extends Activity implements GoogleApiClient.
      ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener,
      LocationListener{
27
28     private TextView tvCoordinate;
29     private Location location;
30     private LocationRequest locationRequest;
31     private Button btnShowLocation;
32     private Button btnGetLocation;
33     private GoogleApiClient mGoogleApiClient;
34
35     @Override
36     protected void onCreate(Bundle savedInstanceState) {
37         super.onCreate(savedInstanceState);
38         setContentView(R.layout.gps);
39
40         tvCoordinate = (TextView) findViewById(R.id.TvCoordinate);
41         btnShowLocation = (Button) findViewById(R.id.
              ButtonShowLocation);
42         btnGetLocation = (Button) findViewById(R.id.ButtonGetLocation
              );
43
```

```
44          btnGetLocation.setOnClickListener(new View.OnClickListener()
                {
45              @Override
46              public void onClick(View v) {
47                  onBackPressed();
48              }
49          });
50
51          btnShowLocation.setOnClickListener(new View.OnClickListener()
                {
52        @Override
53        public void onClick(View v) {
54                  LocationManager manager = (LocationManager)
                       getSystemService( Context.LOCATION_SERVICE );
55                  boolean isOn = manager.isProviderEnabled(
                       LocationManager.GPS_PROVIDER);
56                  if(isOn) {
57                      callConnection();
58                  }else{
59                      Toast.makeText(Maps.this, "Ative a Localizacao!",
                           Toast.LENGTH_LONG).show();
60                      Intent intent = new Intent(Settings.
                          ACTION_LOCATION_SOURCE_SETTINGS);
61                      startActivity(intent);
62                  }
63        }
64    });
65          callConnection();
66    }
67
68    private synchronized void callConnection(){
69          mGoogleApiClient = new GoogleApiClient.Builder(this)
70                  .addOnConnectionFailedListener(this)
71                  .addConnectionCallbacks(this)
72                  .addApi(LocationServices.API)
73                  .build();
74          mGoogleApiClient.connect();
75    }
76
77    private void initLocationRequest(){
78          locationRequest = new LocationRequest();
```

```
79          locationRequest.setInterval(5000);
80          locationRequest.setFastestInterval(2000);
81          locationRequest.setPriority(locationRequest.
               PRIORITY_HIGH_ACCURACY);
82      }
83
84      private void startLocationUpdate(){
85          initLocationRequest();
86          LocationServices.FusedLocationApi.requestLocationUpdates(
               mGoogleApiClient, locationRequest,Maps.this);
87      }
88      private void stopLocationUpdate(){
89          LocationServices.FusedLocationApi.removeLocationUpdates(
               mGoogleApiClient,Maps.this);
90      }
91
92      @Override
93      public void onConnected(Bundle bundle) {
94          Log.i("LOG", "onConnected(" + bundle + ")");
95
96         location = LocationServices.FusedLocationApi.getLastLocation(
              mGoogleApiClient);
97
98          if(location != null){
99              Log.i("LOG", "latitude: "+ location.getLatitude());
100             Log.i("LOG", "longitude: "+ location.getLongitude());
101             tvCoordinate.setText(location.getLatitude()+","+ location
                    .getLongitude());
102         }
103         startLocationUpdate();
104     }
105
106     @Override
107     public void onConnectionSuspended(int i) {
108         Log.i("LOG", "onConnectionSuspended(" + i + ")");
109     }
110     @Override
111     public void onConnectionFailed(ConnectionResult connectionResult)
            {
112         Log.i("LOG", "onConnectionFailed("+connectionResult+")");
113     }
```

```
114
115     @Override
116     public void onBackPressed(){
117         Intent data = new Intent();
118         data.putExtra("gps", tvCoordinate.getText());
119         setResult(2,data);
120         finish();
121         stopLocationUpdate();
122     }
123
124     @Override
125     public void onLocationChanged(Location location) {
126         tvCoordinate.setText(location.getLatitude()+","+ location.
              getLongitude());
127     }
128 }
```

Listing C.2: Código fonte gerado a partir do *template* classGps.egl + modelo - *Maps.java*

```
1
2  package gps;
3  import android.app.Activity;
4  import android.app.Fragment;
5  import android.app.FragmentManager;
6  import android.app.FragmentTransaction;
7  import android.net.Uri;
8  import android.os.Bundle;
9  import android.support.v4.app.FragmentActivity;
10 import android.util.Log;
11
12 import com.google.android.gms.appindexing.Action;
13 import com.google.android.gms.appindexing.AppIndex;
14 import com.google.android.gms.appindexing.Thing;
15 import com.google.android.gms.common.api.GoogleApiClient;
16 import com.google.android.gms.location.LocationServices;
17 import com.google.android.gms.maps.CameraUpdateFactory;
18 import com.google.android.gms.maps.GoogleMap;
19 import com.google.android.gms.maps.MapFragment;
20 import com.google.android.gms.maps.MapView;
21 import com.google.android.gms.maps.OnMapReadyCallback;
22 import com.google.android.gms.maps.SupportMapFragment;
23 import com.google.android.gms.maps.model.LatLng;
24 import com.google.android.gms.maps.model.MarkerOptions;
```

```
25 import com.google.firebase.database.DataSnapshot;
26 import com.google.firebase.database.DatabaseError;
27 import com.google.firebase.database.DatabaseReference;
28 import com.google.firebase.database.ValueEventListener;
29 import com.src.android.R;
30
31 import java.util.ArrayList;
32 import java.util.List;
33 [%for (projeto in projeto){
34 for(relationAttributeToComponent in projeto.
      relationAttributeToComponent){
35   if(relationAttributeToComponent.target.type.toString().equals("GPS"
       )){%]
36 import modelo.[%=relationAttributeToComponent.source.classname.
      firstToUpperCase()%]VO;
37 [%}}}%]
38 import util.FirebaseUtil;
39
40  /**@author Willamys Araujo
41  **Generate for Jacroid**/
42
43 public class GpsMapsPointers extends FragmentActivity implements
      OnMapReadyCallback {
44     private GoogleMap mMap;
45     DatabaseReference reference;
46     /***CRIANDO A LISTA DE PONTOS **/
47 [%for (projeto in projeto){
48   for(relationAttributeToComponent in projeto.
       relationAttributeToComponent){
49     //if(relationAttributeToComponent.source.classname.equals(classes
         .name) and relationAttributeToComponent.target.type.toString()
         .equals("GPS")){
50     if(relationAttributeToComponent.target.type.toString().equals("
         GPS")){%]
51   ArrayList<[%=relationAttributeToComponent.source.classname.
       firstToUpperCase()%]VO> [%=relationAttributeToComponent.source.
       classname.ToLowerCase()%]List;
52 [%}}}%]
53     @Override
54     public void onCreate(Bundle savedInstanceState) {
55         super.onCreate(savedInstanceState);
```

```
56          setContentView(R.layout.map);
57      [%if(IDE.equals("AndroidStudio")){%]
58          SupportMapFragment mapFragment = (SupportMapFragment)
               getSupportFragmentManager()
59                    .findFragmentById(R.id.map);
60          mapFragment.getMapAsync(this);
61          [%}else{%]
62          MapFragment mapFragment = MapFragment.newInstance();
63      FragmentManager manager = getFragmentManager();
64        FragmentTransaction transaction = manager.beginTransaction();
65        transaction.add(R.id.map, mapFragment);
66        transaction.commit();
67        mapFragment.getMapAsync(this);
68        [%}%]
69      }
70
71      @Override
72      public void onMapReady(GoogleMap googleMap) {
73        mMap = googleMap;
74        FirebaseUtil.getInstance().getInit(getApplicationContext());
75      [%for (projeto in projeto){
76    for( relationAttributeToComponent in projeto.
          relationAttributeToComponent){
77      //if(relationAttributeToComponent.source.classname.equals(classes
           .name) and relationAttributeToComponent.target.type.toString()
           .equals("GPS")){
78      if(relationAttributeToComponent.target.type.toString().equals("
           GPS")){%]
79    [*** CRIANDO A LISTA DE PONTOS **]
80      [%=relationAttributeToComponent.source.classname.ToLowerCase()%]
           List = new ArrayList<[%=relationAttributeToComponent.source.
           classname.firstToUpperCase()%]VO>();
81          [%if(IDE.equals("AndroidStudio")){%]
82          reference = FirebaseUtil.getReference("[%=
              relationAttributeToComponent.source.classname.ToLowerCase
              ()%]s");
83          [%}else{%]
84      reference = FirebaseUtil.getInstance().getFirebaseDatabase().
           getReference("[%=relationAttributeToComponent.source.classname
           .ToLowerCase()%]s");
85          [%}%]
```

```
86          reference.addValueEventListener(new ValueEventListener() {
87              @Override
88              public void onDataChange(DataSnapshot dataSnapshot) {
89                  for (DataSnapshot msgSnapshot: dataSnapshot.
                        getChildren()) {
90                  [%if(projeto.firebaseAuth.isUndefined <> true){%]
91                    for(DataSnapshot msgSnapshot2: msgSnapshot.
                          getChildren()){
92                    [%=relationAttributeToComponent.source.classname.
                          ToLowerCase()%]List.add(msgSnapshot2.getValue
                          ([%=relationAttributeToComponent.source.
                          classname.firstToUpperCase()%]VO.class));
93                        String local = msgSnapshot2.getValue([%=
                            relationAttributeToComponent.source.
                            classname.firstToUpperCase()%]VO.class).get
                            [%=relationAttributeToComponent.source.name.
                            firstToUpperCase()%]();
94                        String latit =  local.substring(0,local.indexOf
                            (","));
95                        String longi = local.substring(local.indexOf(",
                            ")+1, local.length()-1);
96                        LatLng locate = new LatLng(Double.parseDouble(
                            latit),Double.parseDouble(longi));
97                        mMap.addMarker(new MarkerOptions().position(
                            locate).snippet(msgSnapshot2.getKey()).title
                            ("[%=relationAttributeToComponent.source.
                            classname.ToLowerCase()%]"));
98                        mMap.moveCamera(CameraUpdateFactory.
                            newLatLngZoom(locate,18));
99                    }
100                 }
101         [%}else{%]
102                   [%=relationAttributeToComponent.source.classname.
                          ToLowerCase()%]List.add(msgSnapshot.getValue
                          ([%=relationAttributeToComponent.source.
                          classname.firstToUpperCase()%]VO.class));
103                    String local = msgSnapshot.getValue([%=
                           relationAttributeToComponent.source.classname.
                           firstToUpperCase()%]VO.class).get[%=
                           relationAttributeToComponent.source.name.
                           firstToUpperCase()%]();
```

```
104                     String latit =  local.substring(0,local.indexOf("
                            ,"));
105                     String longi = local.substring(local.indexOf(",")
                            +1, local.length()-1);
106                     LatLng locate = new LatLng(Double.parseDouble(
                            latit),Double.parseDouble(longi));
107                     mMap.addMarker(new MarkerOptions().position(
                            locate).snippet(msgSnapshot2.getKey()).title("
                            [%=relationAttributeToComponent.source.
                            classname.ToLowerCase()%]"));
108                     mMap.moveCamera(CameraUpdateFactory.newLatLngZoom
                            (locate,18));
109                 }
110              [%}%]
111             }
112             @Override
113             public void onCancelled(DatabaseError databaseError) {
114             }
115         });
116
117   [%}}}%]
118     }
119 }
```

Listing C.3: Código *template GpsMapsPointers.egl*

```
1
2  package gps;
3  import android.app.Activity;
4  import android.app.Fragment;
5  import android.app.FragmentManager;
6  import android.app.FragmentTransaction;
7  import android.net.Uri;
8  import android.os.Bundle;
9  import android.support.v4.app.FragmentActivity;
10 import android.util.Log;
11
12 import com.google.android.gms.appindexing.Action;
13 import com.google.android.gms.appindexing.AppIndex;
14 import com.google.android.gms.appindexing.Thing;
15 import com.google.android.gms.common.api.GoogleApiClient;
16 import com.google.android.gms.location.LocationServices;
17 import com.google.android.gms.maps.CameraUpdateFactory;
```

```
18 import com.google.android.gms.maps.GoogleMap;
19 import com.google.android.gms.maps.MapFragment;
20 import com.google.android.gms.maps.MapView;
21 import com.google.android.gms.maps.OnMapReadyCallback;
22 import com.google.android.gms.maps.SupportMapFragment;
23 import com.google.android.gms.maps.model.LatLng;
24 import com.google.android.gms.maps.model.MarkerOptions;
25 import com.google.firebase.database.DataSnapshot;
26 import com.google.firebase.database.DatabaseError;
27 import com.google.firebase.database.DatabaseReference;
28 import com.google.firebase.database.ValueEventListener;
29 import com.src.android.R;
30
31 import java.util.ArrayList;
32 import java.util.List;
33 import modelo.LembreteVO;
34 import util.FirebaseUtil;
35
36  /**@author Willamys Araujo
37  **Generate for Jacroid**/
38
39 public class GpsMapsPointers extends FragmentActivity implements
      OnMapReadyCallback {
40     private GoogleMap mMap;
41     DatabaseReference reference;
42     /***CRIANDO A LISTA DE PONTOS **/
43   ArrayList<LembreteVO> lembreteList;
44     @Override
45     public void onCreate(Bundle savedInstanceState) {
46         super.onCreate(savedInstanceState);
47         setContentView(R.layout.map);
48         MapFragment mapFragment = MapFragment.newInstance();
49     FragmentManager manager = getFragmentManager();
50       FragmentTransaction transaction = manager.beginTransaction();
51       transaction.add(R.id.map, mapFragment);
52       transaction.commit();
53       mapFragment.getMapAsync(this);
54     }
55
56     @Override
57     public void onMapReady(GoogleMap googleMap) {
```

```
58        mMap = googleMap;
59        FirebaseUtil.getInstance().getInit(getApplicationContext());
60      lembreteList = new ArrayList<LembreteVO>();
61      reference = FirebaseUtil.getInstance().getFirebaseDatabase().
           getReference("lembretes");
62          reference.addValueEventListener(new ValueEventListener() {
63              @Override
64              public void onDataChange(DataSnapshot dataSnapshot) {
65                  for (DataSnapshot msgSnapshot: dataSnapshot.
                       getChildren()) {
66                    for(DataSnapshot msgSnapshot2: msgSnapshot.
                         getChildren()){
67                    lembreteList.add(msgSnapshot2.getValue(LembreteVO.
                         class));
68                        String local = msgSnapshot2.getValue(LembreteVO
                           .class).getLocalizacao();
69                        String latit =  local.substring(0,local.indexOf
                           (","));
70                        String longi = local.substring(local.indexOf(",
                           ")+1, local.length()-1);
71                        LatLng locate = new LatLng(Double.parseDouble(
                           latit),Double.parseDouble(longi));
72                        mMap.addMarker(new MarkerOptions().position(
                           locate).snippet(msgSnapshot2.getValue(
                           LembreteVO.class).getDescricao()).title("
                           lembrete"));
73                        mMap.moveCamera(CameraUpdateFactory.
                           newLatLngZoom(locate,18));
74                    }
75                }
76              }
77              @Override
78              public void onCancelled(DatabaseError databaseError) {
79              }
80          });
81
82      }
83  }
```

Listing C.4: Código fonte gerado a partir do *template* GpsMapsPointers.egl + modelo - *GpsMapsPointers.java*