UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO ACADÊMICO EM CIÊNCIA DA COMPUTAÇÃO

FABIANO JOSÉ GADELHA DE FREITAS

JUSTMODELING: UMA ABORDAGEM MDE PARA DESENVOLVIMENTO DE APLICAÇÕES NEGOCIAIS PARA ANDROID

FORTALEZA – CEARÁ

2016

FABIANO JOSÉ GADELHA DE FREITAS

JUSTMODELING: UMA ABORDAGEM MDE PARA DESENVOLVIMENTO DE APLICAÇÕES NEGOCIAIS PARA ANDROID

Dissertação apresentada ao Curso de Mestrado Acadêmico em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências e Tecnologia da Universidade Estadual do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientador: Prof. Paulo Henrique Mendes Maia, Ph.D

FORTALEZA – CEARÁ

2016

Dados Internacionais de Catalogação na Publicação

Universidade Estadual do Ceará

Sistema de Bibliotecas

Freitas, Fabiano José Gadelha de.
JustModeling: Uma abordagem MDE para desenvolvimento de aplicações negociais para Android [recurso eletrônico] / Fabiano José Gadelha de Freitas. - 2016.
1 CD-ROM: il.; 4 ¾ pol.

CD-ROM contendo o arquivo no formato PDF do trabalho acadêmico com 93 folhas, acondicionado em caixa de DVD Slim (19 x 14 cm x 7 mm).

Dissertação (mestrado acadêmico) - Universidade Estadual do Ceará, Centro de Ciências e Tecnologia, Mestrado Acadêmico em Ciência da Computação, Fortaleza, 2016.
Área de concentração: Ciência da Computação.
Orientação: Prof. Dr. Paulo Henrique Mendes Maia.

1. Model-Driven Engineering. 2. Naked Objects. 3. Padrões de Projeto. 4. Geração de Código. 5. Android. I. Título.

Mestrado Acadêmico em Ciência da Computação UECE MACC

ATA DA NONAGÉSIMA TERCEIRA DEFESA PÚBLICA DE DISSERTAÇÃO DE MESTRADO

Ao décimo sexto dia do mês de dezembro de dois mil e dezesseis, no miniauditório do prédio de Pesquisa e Pós-Graduação em Computação, do Mestrado Acadêmico em Ciência da Computação – MACC, realizou-se a sessão pública de defesa da dissertação de **Fabiano José Gadelha de Freitas ,** aluno regularmente matriculado no Mestrado Acadêmico em Ciência da Computação–MACC, Intitulada: **"JustModeling: Uma abordagem MDE para desenvolvimento de aplicações negociais para Androide".** A Banca Examinadora reuniu-se no horário das 9h às 11 horas, sendo constituída pelos Professores Doutores **Paulo Henrique Mendes Maia, (Orientador/UECE); Mariela Inés Cortés (UECE); Cidcley Teixeira de Souza (IFCE).** Inicialmente o mestrando expôs seu trabalho e a seguir foi submetido à arguição pelos membros da Banca, dispondo cada membro de tempo para tal. Finalmente a Banca reuniu-se em separado e concluiu por considerar o mestrando aprovado, por sua dissertação e sua defesa pública. Eu, **Prof. Dr. Paulo Henrique Mendes Maia**, Orientador e Presidente da Banca, lavrei a presente Ata que será assinada por mim e demais membros da Banca. Fortaleza, 16 de dezembro de 2016.

Prof. Dr. Paulo Henrique Mendes Maia
Orientador – UECE

Profa. Dra. Mariela Inés Cortés
(UECE)

Prof. Dr.Cidcley Teixeira de Souza
(IFCE)

AGRADECIMENTOS

Primeiramente à Deus, por todas as graças alcançadas e pelos momentos de iluminação e superacão diante das dificuldades enfrentadas ao longo do desenvolvimento desde trabalho e da vida como um todo.

Aos meus pais Edite e Raimundo pelo exemplo de vida correta e batalhadora, e pelo paoio que sempre me deram em todos os momentos, principalmente diante dos maiores desafios.

Aos meus irmãos Bertiane, Adriano e Cristiane, pelo acolhimento e companheirismo ao longo dos anos e também a minha prima Angélica, a mais nova irmã, que me inspira com seu exemplo a cada vitória diária.

À minha esposa Jamilly, pelo seu apoio, paciência e compreensão em todos os nossos dias, principalmente nos dias de abdicação em razão do desenvolvimento deste trabalho.

Ao meu orientador Paulo Henrique, por compartilhar comigo seu tempo e conhecimento, e por ter acreditado na minha capacidade de realizar um bom trabalho.

À esta universidade, seu corpo docente, direção e administração pela opotunidade de poder aprimorar meus conhecimentos estudando em uma universidade pública e de qualidade.

“Obstáculos são aqueles perigos que você vê quando tira os olhos de seu objetivo.”

(Henry Ford)

RESUMO

O crescimento contínuo do mercado Android resultou em uma maior demanda por aplicações e ciclos de desenvolvimento mais curtos. Desenvolvedores e empresas estão adotando soluções para aumentar a produtividade e reduzir tempo de desenvolvimento e esforço. Dentre elas, a Engenharia Dirigida por Modelos, do inglês *Model-driven Engineering* (MDE), surgiu como uma alternativa concreta para gerar automaticamente aplicações Android. No entanto, os trabalhos atuais geram apenas uma parte da aplicação, forçando o desenvolvedor a gastar algum tempo em tarefas de execução. Para combater isso, este trabalho propõe JustModeling, uma abordagem MDE composta pelo JBModel, uma ferramenta de modelagem gráfica com a qual os usuários modelam as classes de negócio de aplicativos usando o diagrama de classe da UML e que fornece um conjunto de transformações de modelo para gerar o código para o *framework* JustBusiness, que gera automaticamente todos os recursos necessários da aplicação móvel. Isso permite que o desenvolvedor trabalhe em um nível maior de abstração, com foco no projeto do aplicativo, em vez de questões de implementação. A abordagem foi avaliada em dois exemplos de uso nos quais foi comparado a produtividade utilizando o JustModeling e o desenvolvimento tradicional. Em ambos os casos, o JustModeling melhorou consideravelmente o tempo de desenvolvimento e número de artefatos e linhas de código gerados.

Palavras-chave: Engenharia Dirigida por Modelos, Aplicação Android, Geração de Código.

ABSTRACT

The continuous growth of the Android market has resulted in greater demand for applications and shorter development cycles. Developers and companies are adopting solutions to increase productivity and reduce development time and effort. Among them, Model-driven Engineering (MDE) has emerged as a concrete alternative to automatically generate Android applications. However, the current works generate only part of the application, forcing the developer to spend time in implementation tasks. To tackle that, we propose JustModeling, an MDE approach formed by JBModel, a graphical modeling tool with which the user models the application business classes using UML class diagram and that provides a set of model transformations to generate code for the JustBusiness framework, which automatically generates all necessary resources of the mobile application. This allows the developer to work on a higher level of abstraction, focusing on the application design rather than implementation issues. The approach was evaluated in two examples of use in which productivity was compared using JustModeling and traditional development. In both cases, JustModeling greatly improved the development time and number of artifacts and lines of code generated.

Keywords: Model-driven Engineering, Android application, Code generation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADT	Android Developer Tools
API	Application Programming Interface
ART	Android Runtime
BIS	Business Information System
CIM	Computation Independent Model
CRM	Customer Relationship Management
DAO	Data Access Object
DSL	Domain Specific Language
EBNF	Extended Backus-Naur-Form
EMF	Eclipse Modeling Framework
GMF	Graphical Modeling Framework
IDE	Integrated Development Environment
ISM	Implementation Specific Model
J2ME	Java 2 Micro Edition
JDBC	Java Database Connectivity
JPA	Java Persistence API
M2M	Model to Model
M2T	Model to Text
MDA	Model-driven Architecture
MDD	Model-driven Development
MDE	Model-driven Engineering
MOF	Meta-Object Facility
NDK	Native Development Kit
OCL	Object Constraint Language
OMG	Object Management Group
OVM	Object Viewing Mechanism
PIM	Plataform Independent Model
PSM	Platform Specific Model
RH	Recursos Humanos
SDK	Software Development Kit
UML	Unified Modeling Language

XML eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

De acordo com dados da Net Marketshare [1] de maio de 2016, a plataforma Android domina cerca de 70% do mercado de sistemas operacionais para dispositivos móveis, seguindo uma tendência ascendente. Isso faz do Android o sistema operacional móvel mais utilizado no mundo hoje em dia, dando poder a mais de um bilhão de dispositivos entre *smartphones*, *tablets*, *smartwatches* e *smart tvs* [2].

O mercado em expansão resultou em uma demanda cada vez maior para aplicações e, consequentemente, a necessidade de ciclos de desenvolvimento mais curtos. Para atingir isso, desenvolvedores e empresas passaram a investir em *frameworks*, bibliotecas e outros artefatos para alcançar maior produtividade durante o desenvolvimento de aplicações, tais como Roboletric [3] e Ormlite [4], respectivamente, para tarefas de automação de testes e gerenciamento de código de persistência.

Apesar de úteis, esses *frameworks* não agilizam uma das principais tarefas necessárias para a criação de aplicações, que é o desenvolvimento de interfaces e códigos para criação, pesquisa, atualização e remoção (*create, read, update* e *delete* - CRUD) de objetos de negócio, na medida em que não provêm mecanismos de geração automática desses artefatos. Essa tarefa já é comumente realizada para sistemas web (MILOSAVLJEVIć *et al.*, 2003) (COSTA, 2011) e sistemas *desktop* (SILVA; OLIVEIRA, 2009) (COSTA, 2011) (CRUZ; FARIA, 2010). Segundo Pawson (PAWSON, 2004), o desenvolvimento de interfaces de usuário é, em sua grande maioria, a tarefa responsável por uma proporção significativa de todo o esforço envolvido no desenvolvimento de um sistema de negócios interativo, sendo esse esforço ocasionado não pela complexidade na codificação, mas com tempo gasto com detalhes de apresentação.

Existem alguns trabalhos e farramentas na literatura que se apresentam como soluções para a criação de aplicações negociais para a plataforma Android (RIBEIRO; SILVA, 2014a; RIBEIRO; SILVA, 2014b; SILVA; ABREU, 2014a; SILVA; ABREU, 2014b; VAUPEL *et al.*, 2014; **??**). Essas soluções embora resolvam o problema da criação de interfaces e códigos de CRUD, também possuem limitações, dentre as quais destacam-se a falta de integração com o ambiente de desenvolvimento Android, o que resulta em esforço extra para o programador utilizar a abordagem e, em alguns casos, a inexistência de mecanismos de persitência.

[1] https://www.netmarketshare.com/
[2] https://www.android.com/
[3] http://robolectric.org
[4] http://ormlite.com

Como alternativa à essas soluções, Freitas e Maia apresentam *JustBusiness* (FREITAS; MAIA, 2015; FREITAS; MAIA, 2016b), um *framework* para o desenvolvimento de aplicações negociais para a plataforma Android que permite a geração automática de interfaces de usuário e códigos de CRUD a partir de informações obtidas do mapeamento *object-user interface* das classes de negócio. Para tanto, o *framework* implementa o Naked Objects (PAWSON; MATTHEWS, 2001) (PAWSON, 2004), um padrão arquitetural no qual as principais partes da aplicação, ou seja, os objetos de domínio, estão expostos na interface e o usuário pode manipulá-los diretamente através dos métodos desses objetos. Ao utilizar o *JustBusiness*, o desenvolvedor fica responsável apenas pela implementação das classes de negócio, enquanto o *framework* realiza a geração e configuração de todas as classes e arquivos necessários para a execução de uma aplicação Android.

A utilização da solução Naked Objects aplicada pelo *framework JustBusiness* resulta em benefícios claros ao processo de desenvolvimento, como redução de tempo de desenvolvimento e do esforço de implementação de códigos de interface de usuário e persistência. Porém, para sua utilização efetiva, o JustBusiness exige que o programador tenha conhecimento em relação a estrutura e correta utilizacão de seu conjunto de anotações.

Outra alternativa que vem ganhando atenção na última década é o uso de técnicas de Engenharia Dirigida por Modelos, do inglês *Model-driven Engineering* (MDE) (SCHMIDT, 2006), para gerar de maneira automatizada aplicativos Android (PARADA; BRISOLARA, 2012; SABRAOUI; KOUTBI; KHRISS, 2012; RIBEIRO; SILVA, 2014b; LACHGAR; ABDALI, 2014). Na abordagem, a aplicação é modelada e, através de um conjunto de transformações de modelos, o código-fonte é gerado.

Contudo, essas soluções baseadas em MDE em sua maioria não geram uma aplicação móvel estruturalmente completa, o que inclui não somente a programação das classes, mas também as interfaces de usuário, código de persistência e arquivos de recursos. Em vez disso, eles costumam gerar apenas parte da aplicação, tanto interfaces ou a estrutura de classes, o que significa que o programador precisa completar a implementação das classes da aplicação e criar outros artefatos necessários, tais como telas, persistência, arquivos de recursos, dentre outros. Por conseguinte, ainda há uma falta de trabalhos que melhorem a produtividade e facilitem a geração de aplicações Android estruturalmente completas.

Para superar esses inconvenientes, este trabalho propõe *JustModeling* (FREITAS; MAIA, 2016a), uma abordagem MDE para facilitar e agilizar o desenvolvimento de aplicações negociais para a plataforma Android que combina a modelagem gráfica das classes de negócio à

geração não apenas do código da aplicação, mas também dos arquivos de recursos, persistência e telas utilizando o padrão Naked Objects. Para tanto, a abordagem é composta de três fases: na primeira, o desenvolvedor realiza a modelagem apenas das classes de negócio e seus relacionamentos através de *JBModel*, uma ferramenta de modelagem gráfica também proposta neste trabalho; na segunda, essa ferramenta transforma o diagrama de classe do aplicativo em classes Java acrescidas com as anotações fornecidas pelo *JustBusiness* e, por fim, o código de aplicação e sua estrutura de arquivos são gerados utilizando o *framework JustBusiness*.

JustModeling tem como objetivo acrescentar um nível extra de abstração no desenvolvimento de aplicações com JustBusiness, retirando do programador a necessidade de lidar com as anotações e detalhes de código. Como benefício, os desenvolvedores podem se concentrar apenas na tarefa de modelagem, reduzindo assim a complexidade do desenvolvimento.

1.1 OBJETIVOS

A seguir encontram-se listados os objetivos que deseja-se que sejam alcançados ao término do desenvolvimento do presente trabalho.

1.1.1 Objetivo Geral

Implementar a abordagem JustModeling voltada para desenvolvimento de aplicações negociais para dispositivos móveis, mais precisamente para a plataforma Android, que combina técnicas de Engenharia Dirigida por Modelos, como modelagem e transformações de modelos, com os mecanismos de geração de aplicações Android do *framework* JustBusiness.

1.1.2 Objetivos Específicos

Para alcançar o objetivo geral que foi descrito na Subseção 1.1.1, foram definidos os seguintes objetivos específicos:

a) Criação de um metamodelo que permita representar modelos de aplicações negociais.
b) Criação da ferramenta gráfica *JBModel* para a realização de modelagem gráfica de modelos de negócios.
c) Criação dos *scripts* de geração de código fonte.
d) Desenvolvimento de um mecanismo de integração da ferramenta *JBModel* com o JustBusiness.

e) Realização de estudos caso para avaliar o desempenho e a produtividade com a utilização da abordagem JustModeling.

1.2 ESTRUTURA DO TRABALHO

Este trabalho segue estruturado em um total de seis capítulos, incluindo a Introdução, os quais são descritos logo a seguir.

O capítulo 2 apresenta a fundamentação teórica que dá embasamento a este trabalho, apresentando definições e aspectos relativos à Engenharia Dirigida por Modelos como modelo, metamodelagem e a tranformação de modelos. Em seguida, o capítulo apresenta uma visão geral sobre a plataforma Android, abordando principais características da plataforma e desenvolvimento de aplicativos.

O capítulo 3 discute os principais trabalhos encontrados na literatura relacionados ao tema da dissertação. Nesse capítulo são apresentadas as principais contribuições de trabalhos com temas baseados em Engenharia Dirigida por Modelos para dispositivos móveis.

O capítulo 4 apresenta em detalhes o *framework* JustBusiness, abordando sua estrutura, detalhes de implementação e utilização, além dos benefícios e limitações da ferramenta.

O capítulo 5 se concentra em descrever a abordagem JustModeling, apresentando detalhes técnicos, utilização, benefícios gerados e limitações. A ferramenta *JBModel*, desenvolvida para dar suporte à abordagem JustModeling, também é descrita em detalhes.

O capítulo 6 descreve a avaliação realizada através do desenvolvimento de duas aplicações a fim de comparar a produtividade obtida com o uso da abordagem JustModeling em relação ao JustBusiness e desenvolvimento tradicional, bem como detalha os resultados obtidos no experimento.

Finalmente, o Capítulo 7 apresenta as conclusões referentes ao estudo realizado neste trabalho, suas contribuições, limitações encontradas, artigos publicados e destaca os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é realizado um estudo mais aprofundado sobre cada um conteúdos abordados na proposta. Primeiramente são abordados aspectos inerentes ao padrão Naked Objects, detalhando seus princípios, estrutura, bem como vantagens e desvantagens. Em seguida são apresentadas definições e aspectos relativos à Engenharia Dirigida por Modelos. Por fim, o capítulo apresenta uma visão geral sobre a plataforma Android.

2.1 NAKED OBJECTS

O padrão Naked Objects é uma abordagem orientada a objetos onde os objetos de domínio ficam expostos na interface de usuário, e o usuário tem o poder manipulá-los diretamente através da realização de invocações dos métodos implementados por esses objetos (PAWSON, 2004). Nessa abordagem, o projeto de codificação de todas as classes deve respeitar um dos mais importantes princípios da Orientação a Objetos, que é a completude comportamental dos objetos, ou seja, os objetos devem implementar por completo o que eles representam (PAWSON; MATTHEWS, 2002) (PAWSON, 2004) (RAJA; LAKSHMANAN, 2010). A aplicação desse padrão retira do programador a necessidade de implementar interface de usuário ou mecanismos de segurança e persistência, tendo responsabilidade apenas sobre a criação dos objetos de domínio da aplicação, os quais devem implementar de forma completa todo o comportamento que eles propõem representar.

De acordo com Raja e Lakshmanan (RAJA; LAKSHMANAN, 2010), a abordagem Naked Objects se fundamenta em três princípios básicos que juntos caracterizam o padrão:

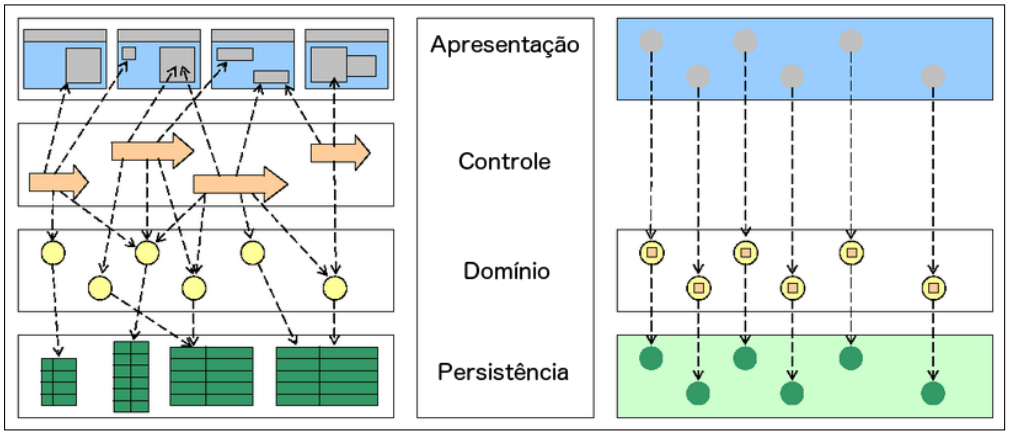
- Toda a lógica de negócio deve ser encapsulada pelos objetos de negócio.
- A interface de usuário deve refletir os objetos de negócio.
- A geração da interface de usuário deve ser obtida de maneira automatizada a partir dos objetos de domínio.

Ainda segundo os autores, esses princípios promovem rapidez no ciclo de desenvolvimento de software, facilidade na análise dos requisitos, maior agilidade e interfaces de usuário mais eficientes.

O padrão Naked Objects é uma alternativa ao padrão de arquitetura 4-camadas. Nele é utilizada uma relação 1:1 entre os elementos das diferentes camadas, onde para objeto de domínio existe apenas uma correspondência em cada uma das demais camadas. Já no padrão 4-camadas, podem haver mapeamentos mais complexos entre as camadas de apresentação,

controle, domínio e persistência (BRANDAO; CORTES; GONCALVES, 2012). A comparação entre os dois padrões de arquitetura é ilustrada pela Figura 1.

Figura 1 – Arquitetura 4-camadas e Arquitetura Naked Objects



Fonte: Adaptado de (PAWSON, 2004)

Em (PAWSON; WADE, 2003), Pawson e Wade realizaram um estudo sobre os possíveis benefícios da utilização da abordagem Naked Objects em um processo de desenvolvimento ágil de software. Dentre os benefícios encontrados, autores destacam que essa abordagem permite o conceito da fase de exploração, onde os usuários e ou clientes, juntamente com o time de desenvolvimento, realizam a prototipação da interface de usuário simultaneamente à realização da atividade de modelagem dos objetos de negócio.

Assim como a aplicação de qualquer padrão, seja ele de projeto ou de arquitetura, no planejamento ou desenvolvimento de aplicações, podem ser encontradas vantagens que encorajem a sua utilização, bem como desvantagens que venham a desfavorecer o seu uso. Dentre as vantagens de utilizar Naked Objects no desenvolvimento de software, pode-se citar a rapidez no ciclo de desenvolvimento de software, facilidade na análise dos requisitos, e melhoria na comunicação entre os usuários e o time de desenvolvedores. Um desvantagem citada em (BRANDãO; CORTéS; GONçALVES, 2012) é a limitação do mapeamento de um para um entre o modelo e a visão.

Foram encontradas apenas duas ferramentas disponíveis *online* que representam em parte soluções Naked Objects voltadas para a plataforma móvel Android: *Isis Android Viewer*[1] e *Naked Object for Android*[2].

O *framework Isis Android Viewer* não chega a ser uma ferramenta para auxiliar no

[1] https://github.com/DImuthuUpe/ISIS_Android_Viewer
[2] http://sourceforge.net/projects/noforandorid

desenvolvimento de aplicações baseadas em Naked Objects, mas um projeto em plataforma Android para se comunicar com uma aplicação *web* desenvolvida a partir do *framework* Apache Isis[3]. A aplicação gerada partir desse projeto se comunica com a aplicação *web* e cria representações tanto de interface de usuário quanto de persistência com base nos objetos de domínio presentes na aplicação *web*.

A ferramenta *Naked Object for Android*, segundo o próprio site, é o primeiro *framework* implementado com foco em agilizar o desenvolvimento de aplicações para a plataforma Android utilizando os princípios do padrão arquitetural Naked Objects. Essa ferramenta não trabalha com geração automática de interfaces de usuário nem mecanismos de persistência ou busca. Atualmente não encontra-se em total funcionamento, além de estar sem atualizações ou sem atividade manutenção recente.

2.2 ENGENHARIA DIRIGIDA POR MODELOS

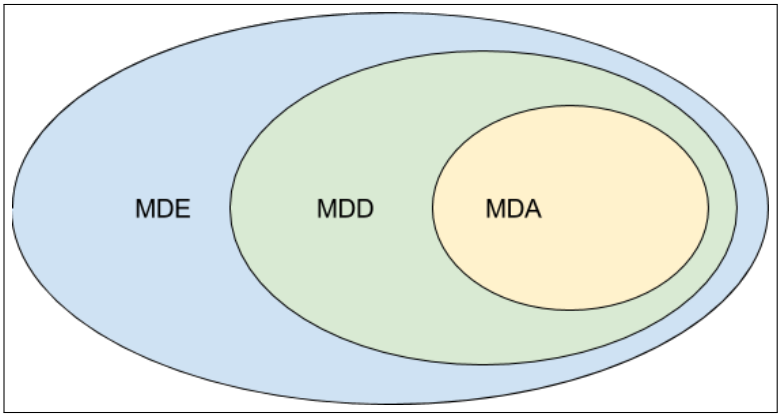
Engenharia Dirigida por Modelos (MDE) é uma abordagem na qual os modelos representam o núcleo do processo de desenvolvimento de *software*, que é totalmente realizado através da construção e transformação de modelos até obter o código-fonte da aplicação (SCHMIDT, 2006). MDE define mecanismos para usar modelos na automação de tarefas de desenvolvimento de *software*, como a configuração, análise, projeto, geração de código, refatoração, e a tradução para outras linguagens e plataformas (FRANCE; RUMPE, 2007). Seu principal objetivo é produzir tecnologias que ajudem desenvolvedores de *software* a abstrair as complexidades das plataformas de implementação subjacentes.

O principal objetivo da MDE é aumentar o nível de abstração utilizando modelos, ao invés de interagir manualmente com todo o código-fonte, ficando o desenvolvedor protegido dessas complexidades nas diversas plataformas de programação. A definição de MDE expande o conceito de Desenvolvimento Orientado por Modelos (Model-driven Development - MDD), que engloba a definição da Arquitetura Orientada por Modelos (Model-driven Architecture - MDA), conforme mostra a Figura 2 (MAIA *et al.*, 2016).

MDD consiste em uma técnica que tem como objetivo reduzir os atrasos e os custos por capitalização dos esforços de projetar modelos em cada fase do ciclo de vida de desenvolvimento de *software*, além de automatizar as transições entre eles. Uma das características fundamentais do MDD é a separação da lógica de negócios de alto nível da arquitetura e

[3] http://isis.apache.org

Figura 2 – Relação entre MDE, MDD e MDA



Fonte: Adaptado de (BRAMBILLA; CABOT; WIMMER, 2012)

implantação de plataforma de sistema (NIKIFOROVA; CERNICKINS; PAVLOVA, 2009).

MDD expande o papel de modelos no processo de desenvolvimento de *software* a partir de uma oportunidade de pensar questões complexas em alto nível antes do desenvolvimento de código (SILVA; BARBOSA; MALDONADO, 2011). Uma das motivações do MDD é o aumento da produtividade da empresa através da diminuição do esforço necessário aos desenvolvedores, aumentando a quantidade de entregas, além da redução na taxa de artefatos que se tornam obsoletos (ATKINSON; KUHNE, 2003).

Já MDA (OMG, 2016), introduzida pela OMG [4] em 2001, consiste em uma visão específica e padrozinada pela OMG sobre o MDD, sendo assim considerada uma subcategoria do MDD (NIKIFOROVA; CERNICKINS; PAVLOVA, 2009) (ELLEUCH; KHALFALLAH; AHMED, 2007).

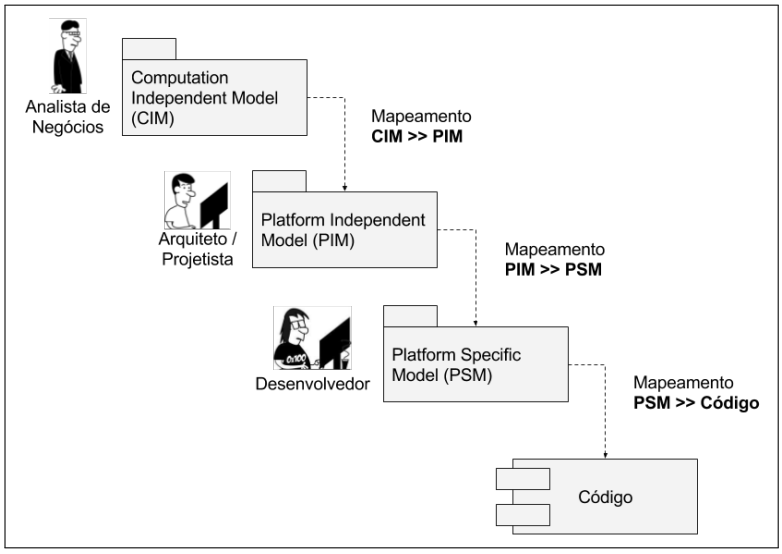
A abordagem MDA considera o modelo em si como a parte central do processo de desenvolvimento de *software*, enquanto que o MDD considera a atividade de modelagem como sendo a mais central (NIKIFOROVA; CERNICKINS; PAVLOVA, 2009). MDA separa a lógica de negócio da lógica de aplicação. Ainda segundo os autores, a lógica de negócio mostra menor complexidade se comparada com tecnologias de implementação subjacentes à lógica da aplicação por apresentar aspectos estáticos e dinâmicos do sistema em alto nível de abstração e esconder os detalhes de implementação (SINGH; SOOD, 2009).

De acordo com (KRIOUILE *et al.*, 2014), a MDA agrupa os tipos de modelos em quatro níveis de abstração: Modelo Independente de Computação (Computation Independent Model - CIM), que se concentra no contexto e requisitos do sistema sem considerar sua estrutura ou processamento; Modelo Independente de Plataforma (Plataform Independent Model - PIM),

[4] http://www.omg.org

que não depende e não restrito à detalhes de tecnologia; Modelo Específico de Plataforma (Platform Specific Model - PSM), que é descrito como um modelo de plataforma que combina especificações de PIM com detalhes específicos de como o sistema deverá se comportar em uma determinada plataforma; e Modelo Específico de Implementação (Implementation Specific Model - ISM), ou Código, que descreve detalhes relativos à programação. Essa relação entre os quatro níveis de modelos encontra-se ilustrada na Figura 3.

Figura 3 – Níveis de Abstração de Modelos em MDA



Fonte: Adaptado de (QUINTERO; ANAYA, 2007)

A Figura 3 descreve o ciclo de tranformação entre modelos partindo do nível mais abstrato até o nível mais conceto. Primeiramente, o processo inicia com o Modelo Independente de Computação (CIM) tranformado em um Modelo Independente de Plataforma (PIM). Em seguida, o Modelo Independente de Plataforma (PSM) é convertido em Modelo Específico de Plataforma, que ao final é transformado em código.

2.2.1 Sintaxe Concreta

Na abordagem MDE, quatro conceitos são considerados como fundamentais: modelo, sintaxe concreta, transformação de modelos e geração de código (BRAMBILLA; CABOT; WIMMER, 2012). Cada um desses conceitos será abordado em detalhes nas seções a seguir.

2.2.2 Modelos

Os modelos são artefatos usados para capturar e representar o conhecimento adquirido durante o processo de desenvolvimento de *software* utilizando abstrações com um certo nível de precisão e detalhe (GOMAA, 2011). Eles ajudam a identificar e especificar a estrutura e o comportamento do sistema e representam a principal fonte de documentação, análise, concepção, construção, implantação e manutenção de um sistema.

Outro conceito importante em MDE é o metamodelo, que especifica a sintaxe abstrata de uma linguagem de modelagem, incluindo os conceitos e relacionamentos. A sintaxe abstrata é considerada o centro da definição da linguagem de modelagem e todas as outras preocupações, como a sintaxe concreta e a semântica são definidas a partir desses metamodelos (BIFFL *et al.*, 2015).

Os tipos de modelos, de acordo com o nível previsto de abstração e especificidade, são agrupados em quatro camadas de abstração, como ilustrado pela Figura 4: metametamodelo (M3), o nível mais abstrato que representa linguagens para especificação de metamodelos; metamodelo (M2), que consiste em linguagens utilizadas para definir as regras para a construção de modelos; modelo (M1), criado a partir de uma metamodelo e usado para representar uma situação específica ou tipo de problema; e objeto real (M0), que consiste em uma instância construída a partir de uma definição de modelo.

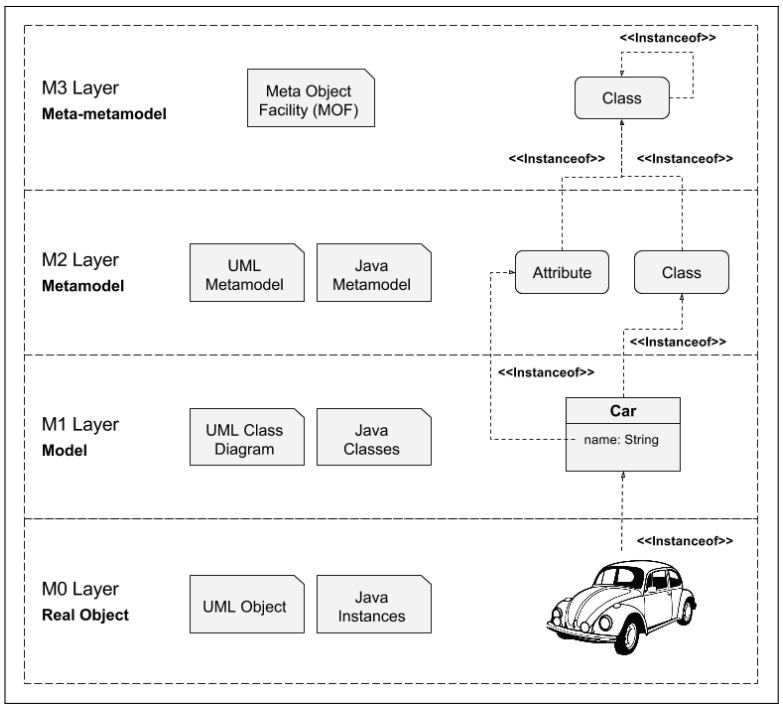
2.2.3 Sintaxe Concreta

A sintaxe concreta pode ser definida como sendo uma linguagem superficial que atua como uma interface fazendo uma ponte entre instâncias e conceitos de uma linguagem e permite que o ser humano possa produzir ou ler instâncias de acordo com a estrutura definida pelo modelo. Uma sintaxe concreta pode ser de natureza gráfica ou textual, mas muitas vezes tem-se uma mistura de ambos (BRAMBILLA; CABOT; WIMMER, 2012).

A sintaxe gráfica tem como objetivo especificar a maneira como componentes pertencentes a um metamodelo serão representados visualmente. Para se obter uma representação consistente de uma sintaxe gráfica, todos os elementos concretos contidos no metamodelo devem possuir uma representação visual associada, para a qual define-se um ícone, imagem ou figura geométrica, além de ser necessário definir quais propriedades deverão ser exibidas na interface. (FONDEMENT, 2007)

A sintaxe textual pode ser dividida em duas categorias: genérica ou específica. A

Figura 4 – Camadas de Abstração de Modelos



Fonte: Elaborado pelo autor

genérica, como o próprio nome já diz, pode ser aplicada de maneira geral a qualquer tipo de modelo. Apesar de abrangente, pode haver casos em que não seja possível ajustá-la de maneira adequada para lidar com as especificidades de alguns domínios. Já a específica é expressa em forma de um conjunto de regras em conformidade com uma gramática do tipo EBNF, podendo ser processada por compiladores para a criação de processadores de texto para a linguagem projetada. Nesse tipo de sintaxe se enquandram as linguagens específicas de domínio (Domain Specific Language - DSL) (BRAMBILLA; CABOT; WIMMER, 2012).

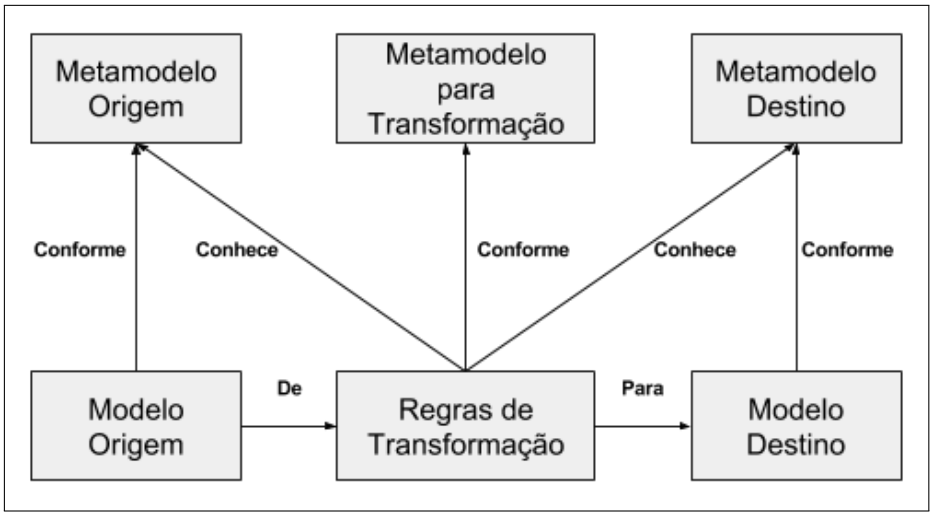
DSL é um tipo linguagem que, a partir do intermédio de notações e de abstrações apropriadas, permite expressar um determinado domínio de problema (DEURSEN; KLINT, 2002). Uma DSL tem como objetivo criar uma linguagem de programação ou especificar uma linguagem própria para o domínio de um determinado problema, assim como representar uma solução em particular para esse problema.

2.2.4 Transformação

Um conceito-chave que da suporte à MDE é a transformação de modelos, que consiste em gerar automaticamente um modelo destino a partir de um modelo origem através do processamento de um conjunto definido de regras de conversão (KLEPPE; WARMER; BAST, 2003). As regras são criadas em termos dos elementos de ambos os metamodelos (origem e

destino), de maneira que o modelo resultante está em conformidade com o seu metamodelo, i.e., a transformação não produz modelos inválidos. As transformações de modelos se enquandram em dois tipos: Modelo para Modelo (M2M) e Modelo para Texto (M2T). A Figura 5 ilustra o processo de tranformação de modelo para modelo, enquanto a Figura 6 descreve a tranformação de modelo para texto.

Figura 5 – Transformação de Modelo para Modelo



Fonte: Adaptado de (MA; HE, 2016)

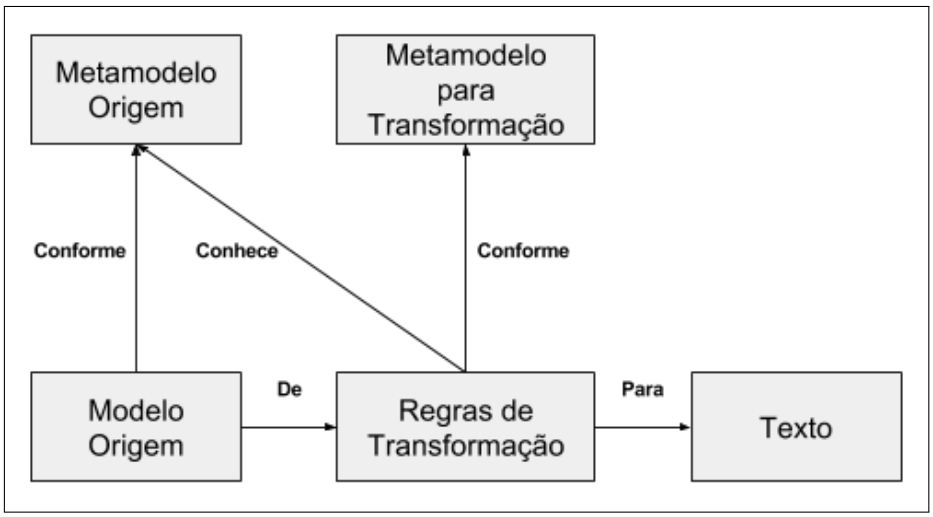
A tranformação M2M é um procedimento que realiza a transformação de um modelo origem em um modelo destino, que podem ou não ser instâncias do mesmo metamodelo, a partir da aplicação de regras de conversão. O procedimento para transformação dos modelos divide-se em quatro etapas: (i) construção dos metamodelos origem e destino; (ii) definição do conjunto de regras de transformação; (iii) criação de um modelo instância com base no metamodelo origem; e (iv) obtenção do modelo destino (BRAMBILLA; CABOT; WIMMER, 2012).

A tranformação M2T consiste em um processo de transformação de modelos no qual o produto final da transformação é texto, mais precisamente, código-fonte para uma determinada linguagem de programação. Esse tipo de tranformação também é conhecido como Modelo para Código (*Model-To-Code* - M2C), ou simplesmente geração de código.

2.3 ANDROID

Atualmente, Android é o sistema operacional para dispositivos móveis mais utilizado no mundo, dando poder a mais de um bilhão de dispositivos móveis entre *smartphones* e *tablets*

Figura 6 – Transformação de Modelo para Texto



Fonte: Elaborado pelo autor

em todo um o mundo, segundo dados o proprio site da plataforma Android [5].

Sistema operacional e *software* de código aberto, o Android abrange um vasto conjunto de dispositivos móveis, além de compor um projeto de código aberto desenvolvido e mantido pelo grupo Open Handset Alliance [6], grupo composto por grandes companhias de *hardware*, *software* e telecomunicações como Samsung, LG e Intel, com liderança do Google, que visa a criação e definição de padrões abertos para telefonia móvel. Um desses projetos, e talvez o mais importante, é a matuenção e expansão da plataforma Android.

2.3.1 Arquitetura

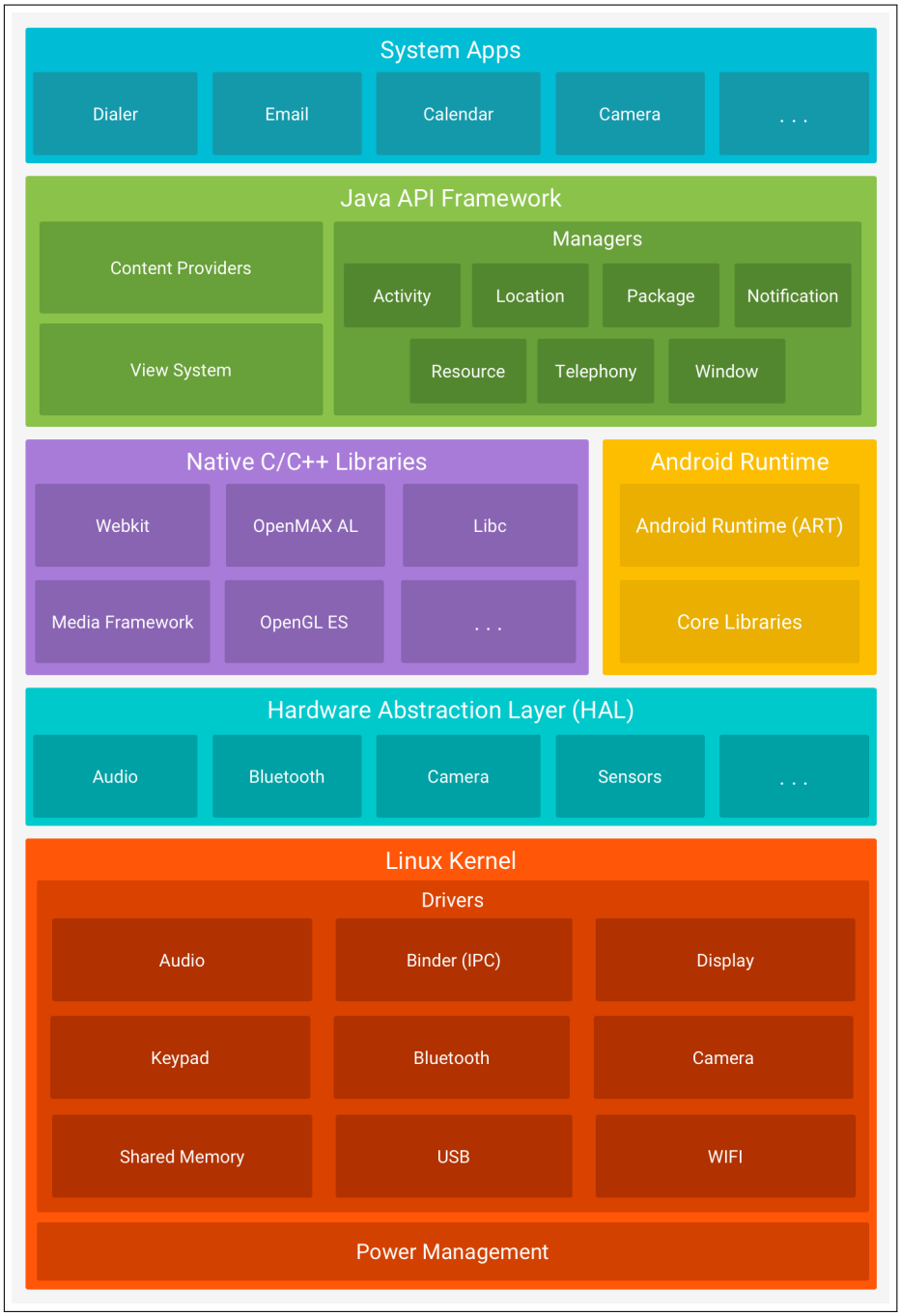
Segundo o guia para desenvolvimento Android (ANDROID..., 2016), o Android é mais que um sistema operacional embarcado, é uma pilha de *softwares*, baseada em Linux [7], de código aberto criada para dar suporte a um ampla gama de dispositivos com as mais variadadas configurações e propriedades.

A arquitetura do sistema operacional é composta por seis camadas, as quais são estruturadas em cinco níveis de hierarquia, como mostra a Figura 7. Bem na base do Android encontra-se uma versão modificada e otimizada do *kernel* Linux, que fornece ao Android vários serviços essenciais, como segurança, rede de dados, gerenciamento de memória e processos, além de uma camada de abstração de *hardware* para as demais camadas de *software*. A utilização de *kernel* do Linux, permite ainda que fabricantes construam *hardwares* direcionados para uma

[5] https://www.android.com/
[6] http://www.openhandsetalliance.com/
[7] https://www.linux.com/

estrutura de *kernel* que já é bem conhecida.

Figura 7 – Arquitetura da Plataforma Android



Fonte: (ANDROID. . . , 2016)

Acima do nível do *kernel* do Linux encontra-se a camada de abstração de *hardware*. Essa camada fornece uma interface padrão que expõe as capacidades e funcionalidades de *hardware* para um nível de abstração mais alto para permitir a comunicação com as camadas superiores. A camada é constituída por vários módulos de biblioteca, sendo que cada um deles implementa uma interface para um tipo específico de componente de *hardware*, tais como os

módulos de câmera, *Bluetooth* e *Wi-fi*.

No próximo nível de abstração encontram-se as camadas de tempo de execução e bibliotecas nativas C/C++. A primeira é responśavel por atividades com compilação de *software*, otimização da coleta de lixo para liberação de memória, além de oferecer suporte à atividades de *debugging*, *profiler*, diagnóstico de exceções e reporte de falhas. Até a versão Android 5.0 (API 21), o Android utilizava a máquina virtual Dalvik, tendo ela sido substituída pela Android Runtime (ART). Na segunda, as bibliotecas C/C++ são utilizadas por diversos dos componentes e serviços do sistema, como: elementos das camadas *hardware* e tempo de execução. Além disso, a camada oferece suporte a *software* que utilize a biblioteca padrao do C (libc), bibliotecas para suporte a formatos de áudio, vídeo e imagens, gerenciador de acesso ao *display*, navegador WebKit e bibliotecas para processamento gráfico 2D (SGL) e 3D (OpenGL ES). A camada de bibliotecas nativas também é responsável por fornecer suporte a aplicações nativas desenvolvidas utilizando o Android NDK.

Logo acima na hierarquia encontra-se a camada Java API *Framework*, que corresponde ao conjunto de recursos do sistema operacional Android que encontram-se disponíveis para o programador através de APIs escritas em linguagem Java, e que têm como objetivo facilitar o desenvolvimento de aplicativos a partir do reuso de componentes. Dentre os recursos disponíveis nas APIs, encontram-se o componentes de interface de usuário, gerenciamento de notificações, recurso, atividades dentre outros.

Por fim, no topo a pilha de camadas encontra-se a camada de aplicações de sistema, onde encontram-se aplicações como e-mail, contatos, navegador, calendário e demais aplicativos construídos utilizando recursos provenientes do Android SDK.

2.3.2 Desenvolvimento de Aplicativos

Na plataforma Android, aplicações são desenvolvidas utilizando a linguagem de programação Java, em conjunto com kit de desenvolvimento de *software* do Android, o Android SDK. Nas versões iniciais do Android, as aplicações eram executadas através da maquina virtual Dalvik, porém hoje encontra-se descontinuada tendo sido substituída pela maquina virtual ART. Além das aplicações que executam sobre a máquina virtual, o Android também oferece suporte para o desenvolvimento e execução de aplicacoes nativas, que são escritas em linguagens C e C++, através da utilização de recursos do kit de desenvolvimento Android, o Android NDK.

Para permitir grandes experiências ao usuário, a plataforma Android disponibiliza

uma gama de recursos que permitem ao desenvolvedor criar aplicativos e jogos, sendo que esses recursos permitem que os aplicativos criados utilizem todo o potencial dos dispositivos, além de oferecer suporte para que eles sejam construídos de maneira otimizada a especificidade de cada tipo de dispositivo que venha a executar o aplicativo. Esses ajustes e adaptações abrangem recursos de interface de usuário com ajustes para o tamanho e qualidade da tela, configurações de localização, utilização de redes de dados, dentre outros.

O Android possui um ambiente de desenvolvimento integrado próprio, o Android Studio [8], e também oferece suporte ao desenvolvimento através de outros ambientes, como: Eclipse IDE [9] e NetBeans IDE [10], permitindo que o desenvolvedor tenha liberdade na escolha do ambiente de senvolvimento. No Android Studio, assim como nas outras ferramentas citadas, o desenvolvedor tem a disposição recursos para auxiliar nas tarefas de desenvolvimento, *debug*, simulação, testes e empacotamento dos aplicativos.

Por fim, os aplicativos ou jogos desenvolvidos para a plataforma Android podem ser distribuídos ou vendidos para usuários do Android através da loja de aplicativos, jogos e conteúdo digital do Android, a *Google Play*, que nas versões inciais do Android era chamada de *Android Market*. De acordo com informações disponibilizadas na página do Android, mais de 1,5 bilhões de downloads são realizados todo mês na *Google Play*, dentre aplicativos, jogos e conteúdos digitais como livros e filmes.

2.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma revisão teórica relativa aos principais conceitos que fundamentam o padrão arquitetural Naked Objects. Os conceitos apresentados enaltecem os benefícios gerados pela utilização dessa abordagem, tanto na própria aplicação, a partir da obtenção de interfaces de usuário padronizadas, quanto no processo de desenvolvimento em si, em virtude do rápido ciclo de análise e desenvolvimento.

Em seguida, foram abordados conceitos e definições sobre a Engenharia Dirigida por Modelos como modelo, metamodelagem, sintaxe concreta, tranformação de modelos e geração de código. De maneira geral, os conceitos abordados ressaltam a importância e poder oferecido por essa abordagem que possui uma visão alternativa em relação aos papéis que os artefatos, principalmente os modelos, desempenham no processo de desenvolvimento de software.

[8] https://developer.android.com/studio
[9] https://eclipse.org/
[10] https://netbeans.org/

Ao final, foi apresentada uma visão geral sobre a plataforma Android, na qual foram abordadas as principais características da plataforma, arquitetura, e desenvolvimento de aplicativos. Os aspectos abordados ajudam a comprender o um a respeito do plano de fundo que dá sustentação a plataforma móvel mais utilizada no planeta.

No próximo capítulo serão apresentados os trabalhos de maior relevância relacionados a esta pesquisa.

3 TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar os trabalhos relacionados às principais abordagens propostas na literatura com base em técnicas para o desenvolvimento de aplicações para dispositivos móveis, focando especialmente nos trabalhos desenvolvidos dentro do contexto do padrão Naked Objects e da abordagem de Engenharia Dirigida por Modelos.

3.1 NAKED OBJECTS PARA APLICAÇÕES DE DISPOSITIVOS MÓVEIS

Keranen e Abrahamsson apresentam em (KERÄNEN; ABRAHAMSSON, 2005) o primeiro estudo de caso empírico feito sobre a aplicação de *Naked Objects* como complemento a técnicas de desenvolvimento ágil de aplicativos para dispositivos móveis. Esse estudo foi realizado a partir do desenvolvimento de um aplicativo para acessar um sistema para controle e visualização de estoque de mercadorias, a ser desenvolvido dentro de período de tempo de oito semanas. Os resultados obtidos a partir da coleta de dados e de entrevistas com os participantes do projeto mostraram que a abordagem se adequa bem ao desenvolvimento ágil de software, porém o mecanismo não era maduro o suficiente para sistemas que tinham que suportar uma grande quantidade de objetos e que necessitem de alta *performance*. Outro problema relatado é a documentação escassa a respeito de como utilizar o *Naked Objects Framework*. Para os autores, a geração rápida de aplicativos é uma característica em potencial do *framework* que pode mudar o mercado de desenvolvimento de *software*, caso as limitações e problemas atuais sejam superadas.

Keranen e Abrahamsson descrevem em (KERANEN; ABRAHAMSSON, 2005) um estudo comparativo entre dois projetos de desenvolvimento ágil de uma aplicação para plataforma móvel Java utilizando duas tecnologias diferentes para a concepção do produto final. O projeto desenvolvido consistia em um aplicativo para acessar um sistema para controle e visualização de estoque de mercadorias para funcionar em dispositivos móveis que utilizassem a plataforma móvel Java (J2ME). As tecnologias avaliadas foram: o desenvolvimento tradicional, onde o programador além de criar a camada de negócio também é responsável por criar as demais camadas, e o desenvolvimento utilizando *Naked Objects*. Para isso, os autores desenvolveram o MIDP-OVM, uma extensão para o *Naked Objects* Framework, adaptando o mecanismo de visualização de objeto (Object Viewing Mechanism - OVM) para criação de aplicativos para a plataforma J2ME. Com esse estudo, os autores identificaram que a utilização de *Naked Objects* reduziu em 79% o código da aplicação e em 91% o código de interface de usuário, em relação

ao desenvolvimento tradicional.

Ambos trabalhos descritos anterioremte usam o *Naked Objects* como solução para o desenvolvimento de aplicações para dispositivos móveis, porém as soluções aplicadas são destinadas a plataformas móveis consideradas obsoletas atualmente. Considerando plataformas mais atuais, como o Android, não foram encontrados trabalhos na literatura, apenas dois projetos disponíveis em repositórios *online*: Isis Android Viewer[1] e Naked Object for Android[2]. Ambos os projetos encontram-se detalhados no Capítulo 2.

3.2 ENGENHARIA DIRIGIDA POR MODELOS PARA APLICAÇÕES DE DISPOSITIVOS MÓVEIS

3.2.1 Aplicações Negociais

Uma abordagem generativa com base em técnicas de MDE para mitigar os problemas relacionados ao desenvolvimento de sistemas de informação negociais, do inglês Business Information Systems (BIS), para dispositivos móveis é apresentado por Silva e Abreu em (SILVA; ABREU, 2014a). A abordagem recebe como entrada um modelo independente de plataforma representado como um diagrama de classe UML combinado com restrições expressas em Object Constraint Language (OCL). O resultado do processo é uma aplicação BIS totalmente funcional codificada para a plataforma alvo escolhida.

Silva e Abreu também apresentam em (SILVA; ABREU, 2014b) uma abordagem para a produção de protótipos de interfaces gráficas totalmente funcionais para aplicações negociais para Android especificadas utilizando diagramas de classe UML acrescidos de anotações. As interfaces de usuários são geradas com base na estrutura das entidades de domínio que foram especificadas no modelo e que permitem diferentes tipos de *screens*, em termos de tamanho, orientação e resolução. Segundo os autores, a abordagem concede uma separação de interesses (dados, comportamento e apresentação), aumentando assim a capacidade de manutenção.

Ribeiro e da Silva apresentam em (RIBEIRO; SILVA, 2014a) a linguagem específica de domínio XIS-Mobile, que consiste em um *profile* UML para o desenvolvimento de aplicações para dispositivos móveis, com o objetivo de atacar os problemas relativos à complexidade do processo de desenvolvimento de software e à fragmentação das plataformas móveis. O XIS-Mobile dispõe de seis tipos de visualizações para modelagem: domínio, entidades de negócio,

[1] https://github.com/DImuthuUpe/ISIS_Android_Viewer
[2] http://sourceforge.net/projects/noforandorid

casos de uso, arquitetura, interação e navegação, com as quais o usuário irá modelar o problema e de mecanismos de transformação de modelo para modelo e modelo para texto para a geração do código nativo da aplicação Android.

Vaupel et al (2014) descrevem uma solução MDE para aplicações negociais Android que permite aos usuários finais, alterar configurações e realizar modificações na aplicação após a geração de código, i.e., com a aplicação em execução, permitindo assim variações da aplicação originalmente criada.

Benouda et al (2016) apresentam uma abordagem para construção de aplicações negociais para Android que utiliza o diagrama de classes UML para a modelagem do domínio e o *framework* Acceleo para a criação dos scripts de geração de código. Segundo os autores, a aplicação gerada possui toda a estrutura de CRUD para permitir a operação das instâncias das classes de negócio contidas no modelo. Embora relatem sucesso na aplicação da abordagem proposta, os autores não informam detalhes referentes ao tipo de persistência utilizada, tipo de integração entre o ambiente de modelagem e o ambiente de desenvolvimento Android, nem disponibilizam o código-fonte do projeto para realização de testes.

Os trabalhos descritos nesta subseção seguem a mesma linha da abordagem Just-Modeling para a criação de aplicações negocias para Android, gerando todo o mecanismo de interface e persistência de dados a partir da modelagem das classes de negócio. A abordagem JustModeling, por sua vez, difere-se ao propor uma estratégia com base em reuso de *software*, ao criar um mecanismo de integração para utilizar o mecanismo de geração de aplicação Android utilizado pelo *framework* JustBusiness, atuando dessa forma, como um nível de abstração acima do JustBusiness, representando uma abordagem híbrida de *Naked Objects* e MDE.

3.2.2 Aplicações de Propósito Geral

Em (PARADA; BRISOLARA, 2012), os autores apresentam uma abordagem MDE para o desenvolvimento de aplicações Android na qual os usuários usam modelos UML, mais precisamente diagramas de classe e sequência, para projetar a estrutura e o comportamento da aplicação, respectivamente. Uma extensão do GenCode (PARADA; SIEGERT; BRISOLARA, 2011) é usada para geração de código, que realiza a transformação dos modelos desenvolvidos em código fonte do Android.

Amendola e Favre descrevem em (AMéNDOLA; FAVRE, 2013) um processo de reengenharia de *software* resultante de uma combinação de técnicas de engenharia reversa com

aspectos de MDE que permitem a migração de aplicativos *desktop* para plataformas móveis. Para validar o processo descrito, os autores desenvolveram um estudo de caso mostrando a modernização de um aplicativo de *desktop* CRM, desenvolvido totalmente em Java usando Swing e JDBC, para a uma aplicação Android.

Em (RIBEIRO; SILVA, 2014b), os autores apresentam *metamodelo-Mobile*, uma abordagem formada por uma linguagem específica de domínio expressa por um perfil UML, combinada com um *framework* baseado em desenvolvimento dirigido por modelos, que visa gerar automaticamente código para aplicações móveis multi-plataformas a partir de especificações feitas em modelos independentes de plataforma. Essa abordagem não gera uma aplicação estruturalmente completa, mas apenas um esqueleto para aplicação, que deve ser concluído pelo programador.

Lachgar e Abdali (LACHGAR; ABDALI, 2014) propõem uma abordagem para o projeto de interfaces de usuário de aplicativos móveis que foi utilizada para gerar aplicações Android. Nesse artigo, os autores descrevem duas estratégias: na primeira, propõem um metamodelo para a criação de aplicativos Android, mas que poderia gerar aplicações para outras plataformas através de transformações M2M, enquanto na segunda propõem um metamodelo genérico para aplicações móveis, que também usando transformações M2M, poderia gerar modelos para qualquer plataforma. A primeira abordagem foi tratada como o foco do trabalho, enquanto a segunda foi considerada como possível evolução. O metamodelo criado de uma aplicação Android consiste em duas categorias: elementos Java e XML, e a partir desses elementos foi projetado o *DSL GUI Android*, uma DSL para modelar aplicativos usando os elementos do Android. O processo definido pelos autores para gerar aplicações é composto por três etapas: modelagem, validação do modelo e geração de código.

Lachgar e Abdali também apresentam em (LACHGAR; ABDALI, 2015) um trabalho muito semelhante ao descrito em (LACHGAR; ABDALI, 2014). Nesse trabalho, eles definiram uma linguagem independente de plataforma como a base para a geração de código nativo para aplicativos para a plataforma iOS utilizando uma abordagem de arquitetura dirigida por modelos. O metamodelo foi baseado na estrutura de projeto e tipos de elementos de um aplicativo iOS e, a partir disso, uma DSL foi especificada para projetar e construir modelos de aplicativos para a plataforma iOS. Os autores argumentam que a utilização de uma DSL no processo de modelagem torna-o mais fácil e intuitivo, se comparado com um processo com base em diagramas gráficos como a UML.

Todos esses trabalhos tratam do desenvolvimento de aplicações de propósito geral,

onde a modelagem é relacionada diretamente com os componentes pertences ao *framework* da plataforma móvel como serviços, atividades, visualizações, objetos de acesso a dados entre outros, não gerando tantos benefícios quando o nicho de aplicações abordado é o de aplicações negociais, onde os projetistas ou desenvolvedores necessitam realizar apenas a modelagem do domínio da aplicação.

3.3 CONSIDERAÇÕES FINAIS

Esta seção apresentou os principais trabalhos encontrados na literatura relacionados a técnicas para o desenvolvimento de aplicações para dispositivos móveis utilizando *Naked Objects* e Engenharia Dirigida por Modelos. Foi também realizada uma comparação entre os trabalhos existentes com a proposta apresentada nesta dissertação.

No próximo capítulo será apresentado o *framework* JustBusiness, apresentando detalhes técnicos, como arquitetura, implementacão e utilização.

4 JUSTBUSINESS

JustBusiness (FREITAS; MAIA, 2015; FREITAS; MAIA, 2016b) é um *framework* baseado no padrão arquitetural Naked Objetcs (PAWSON; MATTHEWS, 2001) que fornece suporte para o desenvolvimento de aplicativos negociais orientados a objetos para plataforma Android. De acordo com o padrão, JustBusiness expõe os objetos de domínio na interface do usuário, permitindo que os usuários possam manipulá-los diretamente através de invocações aos métodos implementados por esses objetos. O *framework* remove do programador a responsabilidade de construir interfaces de usuário e mecanismos de persistência, uma vez que gera esses artefatos de maneira automatizada.

Usando JustBusiness, o desenvolvedor implementa apenas as classes de negócios, que devem ser acrescidas com as anotações fornecidos pelo *framework* e, a partir disso, códigos de interface de usuário e persistência, juntamente com os recursos do Android e arquivos de configuração, podem ser gerados.

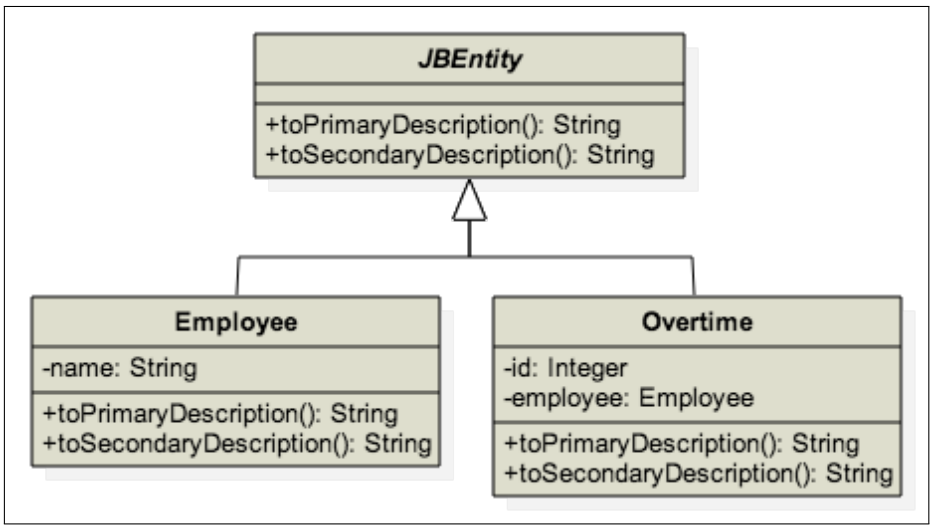
Nas subseções a seguir serão detalhados aspectos relativos à arquitetura do *framework*, à utilizacão de anotações e à geração de código da aplicação. Por fim, será apresentado um exemplo de utilização do JustBusiness, detalhando a configuração de um classe de negócio com anotações e as telas obtidas da aplicação resultante.

4.1 ARQUITETURA

O *framework* JustBusiness possui, em sua estrutura de classes, uma superclasse abstrata *JBEntity*, a partir da qual todas as classes de negócio da aplicação devem extendê-la. A classe *JBEntity* é uma classe abstrata sem atributos, contendo apenas dois métodos, *toPrimaryDescription* e *toSecondaryDescription*, também abstratos, os quais devem ser implementados pelas classes que a estenderem, como mostra a Figura 8. Esses métodos servem para fornecer informações sobre os objetos na interface de usuário, tendo como foco telas de listagem (*ListActivities*). Por padrão, as células de listagem que são utilizadas pelas *ListActivities* possuem um ou dois campos para exibir informações.

As classes de negócio devem herdar a superclasse *JBEntity* e, devem possuir um construtor padrão sem argumentos e os atributos privados com seus respectivos métodos *get* e *set*. A estrutura das classes de negócio é semelhante a classes do tipo POJO (Plain Old Java Object), porém diferem ao estenderem a superclasse *JBEntity* e utilizarem anotações predefinidas.

Figura 8 – Arquitetura do *framework* JustBusiness



Fonte: Elaborado pelo autor

4.2 ANOTAÇÕES

Para obter todas as informações necessárias das classes de negócio, JustBusiness fornece um conjunto de anotações que permitem o mapeamento de informações e a configuração das classes de negócio. Com essas anotações, o desenvolvedor pode detalhar informações para configurar detalhes da apresentação e configurações de persistência. A partir das informações mapeadas pelo uso de anotações, o JustBusiness faz uso de um processador de anotação que analisa, em tempo de compilação, cada anotação utilizada na configuração das classes de negócio, onde todas as informações obtidas a partir do processamento são armazenadas num dicionário de dados.

4.2.1 Anotações para Interface de Usuário

Essas anotações definem configurações como quais classes de negócio serão reconhecidas, quais atributos serão exibidos na tela, em que ordem os atributos estarão dispostos, quais métodos estarão disponíveis na interface, entre outras configurações. O JustBusiness utiliza as anotações @Entity, @Attribute, @Action, @Parameter e @Enumeration para identificar e configurar as classes, atributos, métodos, parâmetros de métodos e enumerações, respectivamente. Essas anotações encontram-se detalhadas na Tabela 2.

Vale ressaltar que o atributo *views* pertencente à anotação *@Attribute*, representa o conjunto de telas em que o atributo será exibido. Esse atributo recebe valores do tipo enumeração *KindView*, o qual é composto pela seguinte coleção de literais: *ALL*, que representa todas as telas;

Tabela 2 – Anotações para Interface de Usuário

Anotação	Descrição
@Entity	Descrever e configurar as classes que serão reconhecidas. Parâmetros: **label** - Nome que será exibido para a entidade **collectionLabel** - Nome no plural para a entidade **icon** - Nome da imagem que será utilizada como ícone pela entidade. A imagem deve estar na pasta (res/drawable)
@Attribute	Descrever e configurar os atributos da classe que serão reconhecidos. Parâmetros: **name** - Nome que será exibido para o atributo **order** - Ordem em que o atributo é mostrado na interface **views** - Telas em que o atributo será exibido.
@Action	Descrever e configurar os métodos de classe que serão reconhecidos. Parâmetros: **name** - Nome que será exibido para o método **order** - Ordem em que o método é mostrado no menu
@Parameter	Descrever e configurar parâmetros de métodos de classe que serão reconhecidos. Parâmetros: **name** - Nome que será exibido para o parâmetro **order** - Ordem em que o parâmetro é mostrado no formulário
@Enumaration	Descrever e configurar as enumerações que serão reconhecidas.

Fonte: Produzido pelo autor

INSERT, que representa apenas a tela de inserção; *EDIT*, que representa apenas a tela de edição; *DETAIL*, que representa apenas a tela de detalhes; e *FIND*, que representa apenas a tela de busca.

4.2.2 Anotações para Persistência

A plataforma móvel Android utiliza a base de dados SQLite, uma base de dados bastante simples e também muito limitada. Uma das limitações do SQLite é referente aos tipos de dados por ele suportado, retringindo-se aos tipos *INTEGER*, *TEXT*, *NONE*, *REAL* e *NUMERIC*. Além disso, a plataforma Android não oferece suporte à Java Persistence API (JPA) e também não reconhece o pacote *javax.persistence*, que é composto por um conjunto de anotações e outras classes voltadas para mapeamento de informações de persistência.

O JustBusiness agrega algumas funcionalidades de persistência para prover o mapeamento objeto-relacional entre as classes de negócio e tabelas na base de dados SQLite. Para isso, foi definido um conjunto de anotações próprias tomando como base as anotações contidas no pacote *javax.persistence*, além de prover um mecanismo para criação de tabelas e acesso à base de dados SQLite a partir das informações mapeadas com essas anotações de persistência. O conjunto de anotações utilizadas para o mapeamento de informações relativas à persistência de dados encontra-se listado nas Tabelas 3, 4 e 5.

Tabela 3 – Anotações para Persistência de Dados - Parte 1

Anotação	Descrição
@Table	Identificar e configurar uma classe como uma tabela. Parâmetros: **catalog** - Nome do catálogo **name** - Nome da tabela **schema** - Nome do esquema
@Id	Identificar um atributo como sendo uma chave na tabela.
@Column	Identificar um atributo como uma coluna na tabela. Parâmetros: **name** - Nome da coluna **unique** - Identifica se a coluna é única **nullable** - Identifica se a coluna recebe valor nulo
@JoinColumn	Identificar e configurar um atributo como uma coluna para realizar a operação *JOIN* com uma coluna de outra tabela. Parâmetros: **name** - Nome da coluna chave estrangeira **table** - Nome da tabela que contem a coluna **referencedColumnName** - Nome da coluna referenciada pela chave estrangeira **unique** - Identifica se a coluna é única **nullable** - Identifica se a coluna recebe valor nulo
@JoinTable	Identificar e configurar um atributo como uma coluna para realizar a operacão *JOIN* com uma outra tabela. Parâmetros: **catalog** - Nome que será exibido para a entidade **name** - Nome no plural para a entidade **schema** - Nome no plural para a entidade **joinColumns** - Nome no plural para a entidade **inverseJoinColumns** - Nome no plural para a entidade

Fonte: Produzido pelo autor

Na Tabela 4, o atributo *cascade*, pertencente às anotações @*OneToMany*, @*OneToOne*, @*ManyToOne* e @*ManyToMany*, representa o tipo de operação cascata que o atributo anotado irá realizar. Esse atributo recebe como valor uma coleção de itens do tipo enumeração *CascadeType*, o qual é composto pelos literais: *ALL*, que representa todas as operações; *DETACH*, que representa operações de *detach*; *PERSIST*, que representa operações de persistência; *REFRESH*, que representa operações de atualização; e *REMOVE*, que representa operações de remoção. Atualmente, as operações referentes a esse atributo não encontram-se implementadas.

Ainda na Tabela 4, o atributo *fetch*, pertencente também às anotações @*OneToMany*, @*OneToOne*, @*ManyToOne* e @*ManyToMany*, representa a forma como os valores da associação serão carregados. Esse atributo recebe valor do tipo enumeração *FetchType*, que é composto pelos literais: *EAGER*, que indica que os valores da associação serão carregados juntamente com a instância da classe e *LAZY*, que indica que os valores da associação serão carregados somente sobre demanda. Esse atributo está definido por padrão como *LAZY*.

Na Tabela 5, o atributo *value* pertencente à anotação @*Temporal* representa que

Tabela 4 – Anotações para Persistência de Dados - Parte 2

Anotação	Descrição
@OneToMany	Identificar e configurar um atributo como uma relação 1:N. Parâmetros: **cascade** - Identifica se a associação realiza operações em cascata **fetch** - Identifica se a associação deve ser carregada da forma *LAZY* ou *EAGER* **mappedBy** - Identifica o atributo onde a associação foi mapeada. **optional** - Identifica se a associação é opcional **targetEntity** - Identifica a classe destido da associação
@OneToOne	Identificar e configurar um atributo como uma relação 1:1. Parâmetros: **cascade** - Identifica se a associação realiza operações em cascata **fetch** - Identifica se a associação deve ser carregada da forma *LAZY* ou *EAGER* **mappedBy** - Identifica o atributo onde a associação foi mapeada. **targetEntity** - Identifica a classe destido da associação
@ManyToOne	Identificar e configurar um atributo como uma relação N:1. Parâmetros: **cascade** - Identifica se a associação realiza operações em cascata **fetch** - Identifica se a associação deve ser carregada da forma *LAZY* ou *EAGER* **optional** - Identifica se a associação é opcional **targetEntity** - Identifica a classe destido da associação
@ManyToMany	Identificar e configurar um atributo como uma relação N:M. Parâmetros: **cascade** - Identifica se a associação realiza operações em cascata **fetch** - Identifica se a associação deve ser carregada da forma *LAZY* ou *EAGER* **mappedBy** - Identifica o atributo onde a associação foi mapeada. **targetEntity** - Identifica a classe destido da associação

Fonte: Produzido pelo autor

Tabela 5 – Anotações para Persistência de Dados - Parte 3

Anotação	Descrição
@Transient	Identificar um atributo que não será mapeado na tabela.
@Enumerated	Identificar um atributo como uma uma enumeração.
@Temporal	Configurar um atributo que armazena informações temporais. Parâmetros: **value** - Identificar como a informação temporal será tratada
@Lob	Identificar e configurar um atributo 'grande', que vai armazenar grande quantidade de informação.

Fonte: Produzido pelo autor

o atributo anotado armazena informações de tempo, mais precisamente presentado pelo tipo *java.util.Date*. Esse atributo recebe valor do tipo enumeração *TemporalType*, que é composto pela seguinte coleção de literais: *DATE*, que representa informações apenas de data; *TIME*, que representa apenas de horário; e *TIMESTAMP*, que representa informações do tipo data e horário.

4.3 GERAÇÃO DE CÓDIGO

O JustBusiness dispõe de um processador de anotações que analisa, em tempo de compilação, as anotações utilizadas na configuração das classes de negócio e armazena as informações obtidas em um dicionário de dados. No momento em que é realizado um *rebuild* do projeto, as informações salvas são processadas pelos geradores de código, que são responsáveis pela criação e configuração dos arquivos necessários para o funcionamento do projeto e obtenção da aplicação.

Após o *rebuild* do projeto, para cada classe de negócio são criados, no mínimo, 25 novos arquivos entre classes de *Controller*, classes (Data Access Object - DAO) e arquivos de recurso subdivididos em *layout* e *menu*. A estrutura de arquivos gerada para as classes de negócio encontra-se ilustrada no lado esquerdo da Figura 9. Além dos arquivos que são gerados para as classes de negócio, o JustBusiness também realiza a construção de um conjunto de classes e arquivos de recursos com base em informações para o projeto, os quais encontram-se ilustrados no lado direito da Figura 9.

O código fonte resumido de parte das classes que compõem o *framework* JustBusiness encontra-se detalhado nos apêndices deste trabalho. O projeto completo contendo todas as classes encontra-se disponível na seção *download* página do JustBusiness [1].

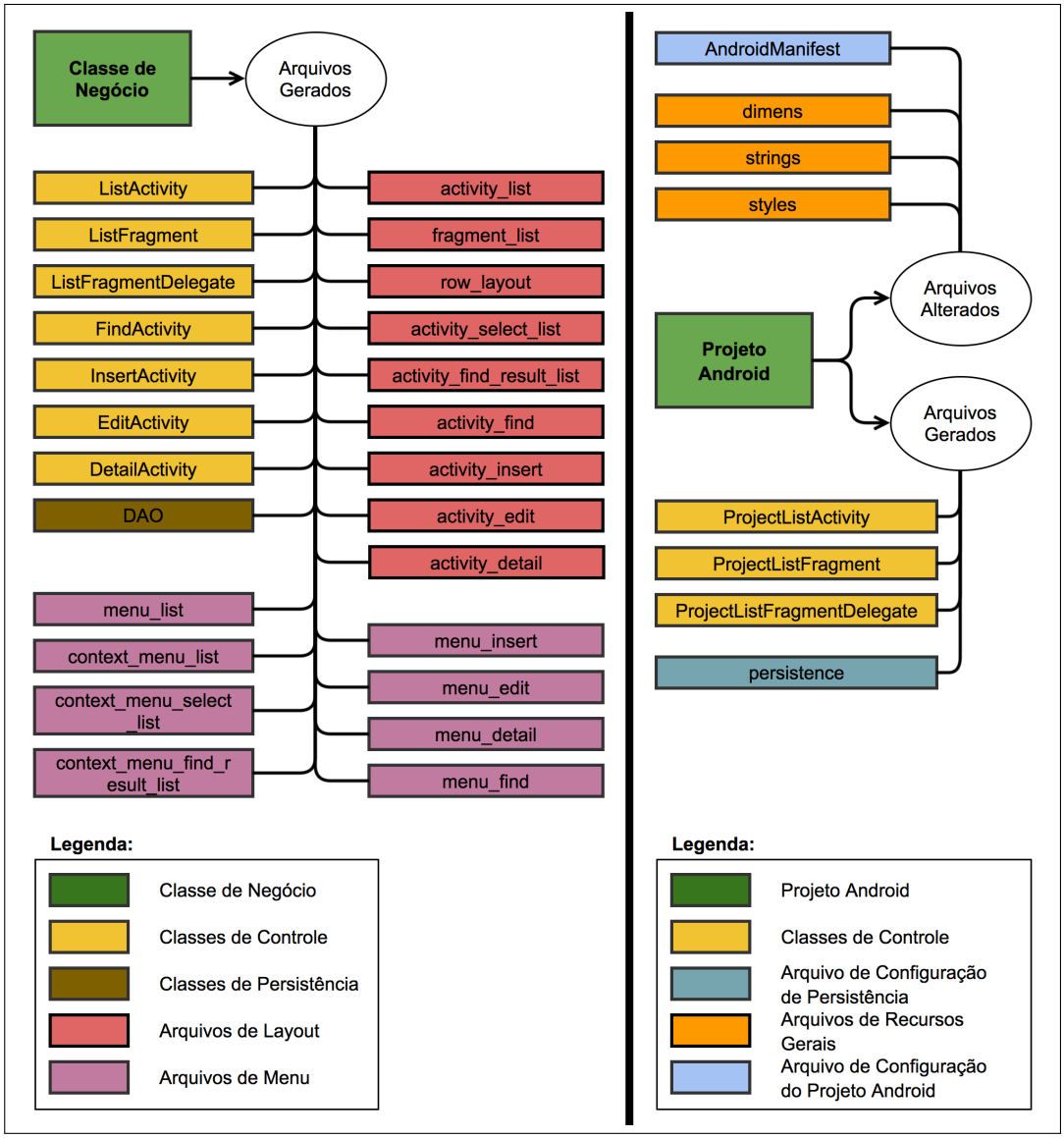
4.3.1 Geração de Interface de Usuário

Através das informações obtidas do mapeamento de classes, atributos, relacionamentos e métodos utilizando as anotações descritas anteriormente, o *framework* se encarrega de gerar todo o mecanismo de interface de usuário. As interfaces de usuário geradas pelo *framework* são geradas em tempo de compilação durante a construção do projeto. Nesse momento, classes de interfaces e arquivos de recurso são criados, e os arquivos de configuração do projeto, como é o caso do AndroidManifest.xml, são alterados para incorporar as alterações realizadas pelo *framework* ao projeto.

Dada uma classe de negócio, a qual possui um conjunto de atributos, o *framework* realiza a construção das interfaces de inserção, edição, detalhes e busca como formulários, onde os componentes associados a cada atributo são dispostos na tela em uma coluna única vertical de acordo com a sequência determinada pelo programador na própria classe de negócio. Os componentes associados aos atributos podem ser habilitados ou desabilitados na interface

[1] https://jbframework.wordpress.com/

Figura 9 – Arquivos Gerados pelo *framework* JustBusiness



Fonte: Elaborado pelo autor

mediante configuração da anotação *@Attribute* na classe de negócio, na qual o programador pode determinar em quais telas cada componente deve ou não aparecer.

4.3.2 Geração de Persistência

Através do mapeamento das classes e seus atributos e relacionamentos utilizando as anotações descritas anteriormente, o *framework* se encarrega de gerar toda a base de dados em SQLite de forma automática, criando todas as tabelas e chaves, dentre outras coisas. Contudo, para que isso funcione, é necessário que o programador faça um mapeamento com informações consistentes. Além disso, para cada classe mapeada no projeto, o *framework* gera automaticamente uma classe de acesso a dados usando o padrão DAO.

4.3.3 Configuração de Arquivos de Projeto

Além da geração das classes controladoras (*Activities*) e arquivos de *layout* para interface, o JustBusiness também é responsável pela atualização e reformulação dos arquivos de configuração do projeto. Um desses arquivos alterados é o AndroidManifest.xml.

O arquivo AndroidManisfest deve ser alterado em tempo de compilação para que o projeto possa identificar todas as classes de controle que foram adicionadas ao projeto, bem como receber informações como nomenclatura e ícone da aplicação antes que aconteça a compilação dos recursos do projeto android. Além do AndroidManifest, outros arquivos de recurso também são alterados, como *strings*, *dimens* e *styles*.

4.3.4 Internacionalização

Todas as informações textuais utilizadas e mapeadas pelo programador através do emprego de anotações nas classes de negócio são compiladas e armazenadas no arquivo de recurso *strings*, que armazena as palavras que são utilizadas dentro do contexto da aplicação.

Essa abordagem permite que, posteriormente, a aplicação possa ser facilmente adaptada para oferecer suporte a uma ou mais diferentes linguagem ou dialetos.

4.4 AMBIENTES DE DESENVOLVIMENTO COM JUSTBUSINESS

Em (FREITAS; MAIA, 2015; FREITAS; MAIA, 2016b), o *framework* JustBusiness foi disponibilizado apenas para o ambiente de desenvolvimento integrado Eclipse [2], o qual era bastante popular e utilizado pela comunidade de desenvolvedores da plataforma Android.

O Eclipse se conecta com a plataforma de desenvolvimento Android através do *plugin* Android Developer Tools (ADT) [3], disponibilidado pela Google [4] no site da plataforma Android. Através do *plugin* ADT, o Eclipse utilizava as bibliotecas e simuladores Android, bem como buscava atualizacões diretamente do repositório da Google. Entretanto, a partir da segunda metade de 2015, a Google anunciou que o Android Studio seria o ambiente de desenvolvimento oficial da plataforma Android e descontinuou o ADT.

Com a descontinuação do ADT, o JustBusiness foi migrado para o ambiente Android Studio [5], que consiste em um ambiente de desenvolvimento completo que permite a criação de

[2] https://eclipse.org/
[3] https://developer.android.com/studio/tools/sdk/eclipse-adt.html
[4] https://www.google.com/
[5] https://developer.android.com/studio

aplicativos para todos os tipos de dispositivos Android. Segundo o site própria IDE, a ferramenta oferece recursos como edição de código de nível global, depuração, ferramentas de desempenho, sistema flexível de compilação e criação/implantação instantâneas.

4.4.1 Migração do Eclipse para o Android Studio

Para realizar a migração do JustBusiness do ambiente Eclipse para o Android Studio, foi primeiramente necessário observar as divergências entre ambos. A primeira dessas divergências é que o Eclipse permite utilizar caminho relativo para acessar arquivos pertencentes ao projeto, enquanto o Android Studio não permite esse tipo de acesso.

Esse detalhe tem impacto significativo uma vez que o JustBusiness necessita acessar arquivos e diretórios do projeto para leitura e escrita. Para isso, fez-se necessário trocar todas as referências de acesso a arquivos com base em caminho relativo para acesso na forma de caminho absoluto.

A segunda diferença entre os ambientes com impacto sobre o JustBusiness está na forma como o ambiente de desenvolvimento realiza a construção (*build*) do projeto. O Eclipse gerencia de maneira implícita a sequência de tarefas executadas pelo compilador para se construir o projeto e obter a aplicação, não sendo possível nem necessário intervenções do programador nesse processo. O Android Studio por sua vez, utiliza por padrão o Gradle[6] para gerenciar de maneira explícita a execução das tarefas necessárias para a criacão de aplicações. Com o Gradle o programador pode alterar a forma como a construção da aplicação será conduzida.

Na migração do JustBusiness fez-se necessário modificar o processo de construção através da inserção de uma tarefa específica de "pré-compilação"para forçar a geração da estrutura de arquivos provida pelo JustBusiness antes da compilação original do projeto Android. Esse procedimento já era implícito no Eclipse, mas teve que ser definido no Android Studio.

4.5 CONFIGURANDO UM PROJETO COM JUSTBUSINESS

Para utilizar o *framework* JustBusiness em um projeto de uma aplicação Android, é necessário seguir um conjunto de passos, como mostra a Figura 10. O JustBusiness encontra-se disponível para os ambientes de desenvolvimento integrado Eclipse e Android Studio.

Primeiramente deve-se importar, utilizando a IDE escolhida, o projeto vazio JustBu-

[6] https://gradle.org/

siness que encontra-se disponível para ser baixado na página web do *framework* JustBusiness [7]. Para cada uma das duas IDEs existe um projeto base diferente, uma vez que os ambientes possuem particularidades na estrutura de projeto e configuração e execução de suas rotinas.

Importado o projeto, o próximo passo é implementar as classes de negócio e somente elas. Elas devem ser construídas como especializações da superclasse *JBEntity*. Logo após, as classes de negócio devem ser configuradas utilizando as anotações fornecidas pelo *framework*, onde o desenvolvedor deve fornecer as informações necessárias e que sejam consistentes para a construção da interface e dos mecanismos de persistência.

Por fim, deve-se limpar e reconstruir o projeto, pois a geração de código ocorre em tempo de compilação. Concluída essa etapa, o projeto está pronto para ser executado. Caso seja realizada uma nova alteração nas classes de negócio, o desenvolvedor deve verificar a necessidade de reconfigurar as classes utilizando as anotações e, em seguida, repetir a etapa de limpar e reconstruir o projeto.

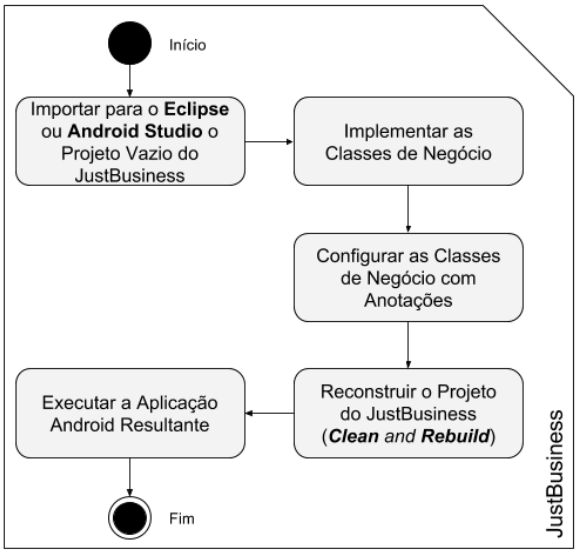


Figura 10 – Fluxograma para a utilização do JustBusiness

4.6 EXEMPLO DE USO DO JUSTBUSINESS

Um exemplo de uso das anotações do JustBusiness para mapear informações nas classes de negócio é ilustrado na Figura 11 através da classe *Person*, a qual é composta: pelos atributos *id, name, address* e *clients*; por um construtor padrão sem parâmetros; pelo método *addAddress*, pelos métodos padrões provenientes da classe *JBEntity*; e por fim os métodos *get* e *set* para cada um dos atributos.

[7] https://jbframework.wordpress.com/

Por uma questão de simplicidade, algumas informações, tais como métodos *get* e *set*, foram omitidos no código-fonte. Ao observar a Figura 11, na linha 1, encontra-se a descrição do pacote em que a classe está contida, enquanto que entre as linhas 3 e 10 encontra-se o código das importações necessárias para a classe *Person*.

Figura 11 – Código fonte da classe Pessoa

```
1  package br.com.macc;
2
3  import org.jb.annotation.model.*;
11
12 @Table(name="person")
13 @Entity(label="Person", collectionLabel="People")
14 public class Person extends JBEntity {
15     @Attribute(order=0, name="Identifier", views={KindView.DETAIL})
16     @Id
17     @Column(name="id_person", nullable=false, unique=false)
18     private Integer id;
19
20     @Attribute(order=1, name="name", views={KindView.ALL})
21     @Column(name="name", nullable=false, unique=false)
22     private String name;
23
24     @Attribute(order=2, name="Address", views={KindView.ALL})
25     @OneToMany(mappedBy="person",
26         targetEntity="br.com.macc.Address")
27     private List<Address> addresses;
28
29     @Attribute(order=3, name="Clients", views={KindView.ALL})
30     @OneToMany(mappedBy="person",
31         targetEntity="br.com.macc.Client")
32     private List<Client> clients;
33
34     public Person() {
35         super();
36         addresses = new ArrayList<Address>();
37         clients = new ArrayList<Client>();
38     }
39
40     @Action(order=0, name="Add Address")
41     public Address addAddress(
42             @Parameter(order=0, name="Description") String Description) {
43         //TODO
44         return null;
45     }
46
47     /**JustBusiness Default Methods*/
48
49     /**GET and SET methods*/
50 }
```

Fonte: Elaborado pelo autor

Na linha 12, *@Entity* indica que a classe *Person* será reconhecida como uma classe negocial. Com essa anotação, o desenvolvedor deve informar os atributos *label* e *collectionLabel*, que é equivalente ao nome da classe no plural. Na linha 13, *@Table* é usado para criar a tabela *person* no banco de dados do aplicativo. Na linha 14 encontra-se a assinatura da classe *Person*, juntamente com o seu relacionamento de generalização com a classe *JBEntity*.

Depois, na linha 15, *@Attribute* especifica que o atributo *id* será reconhecido como

um atributo da classe com *label Identifier*, será o primeiro elemento na tela (*order=0*), e aparecerá apenas na tela de detalhes (*views=KindView.DETAIL*). Na linha seguinte, *@Id* indica que o atributo *id* será uma chave na tabela *person*. Na linha 17, *@Column* determina que o atributo *id* corresponde à coluna *id_person* na tabela *person*.

Na linha 20, *@Attribute* é usado para informar que *name* é um atributo com *label Name*, é o segundo elemento na tela (*order=1*), e vai aparecem em todas as telas (*views=KindView.ALL*). Na linha 21, *@Column* defini que o atributo *name* corresponde à coluna *name* na tabela *person*.

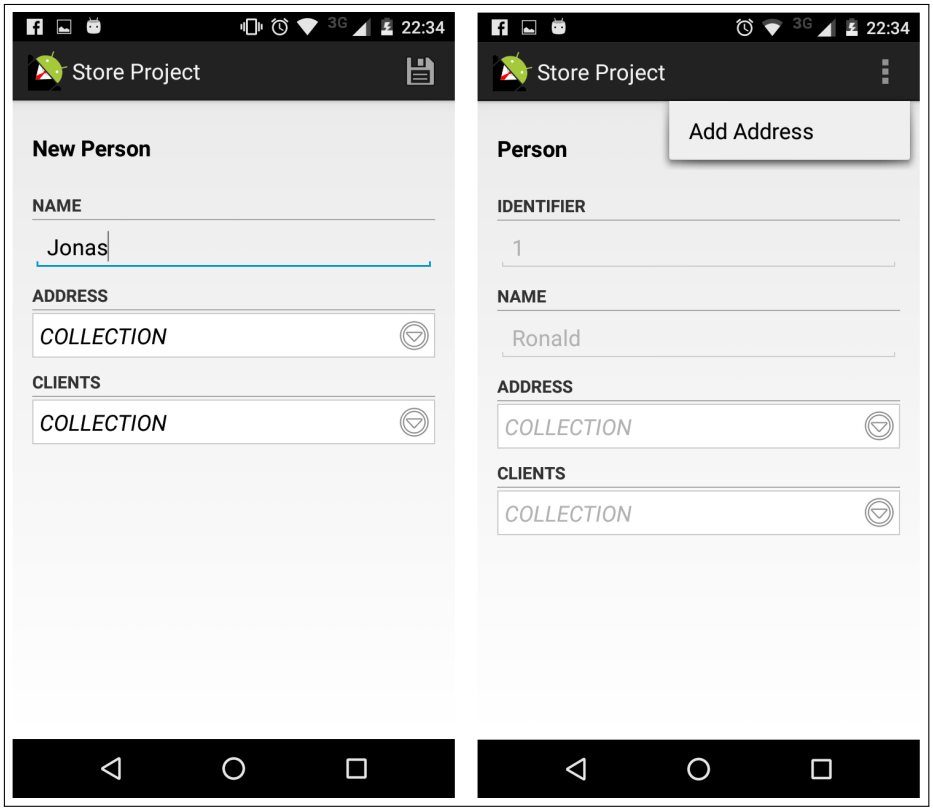
Na linha 24, *@Attribute* determina que *address* deve ser reconhecido como um atributo com label *Address*, será o terceiro elemento da tela, e aparecerá em todas as telas na interface do usuário. Na linha 25, *@OneToMany* especifica que o atributo *address* representa um relacionamento 1:N com a classe *Person*. Uma configuração semelhante à que é usada entre as linhas 29 e 32 é utilizada para realizar o mapeamento do atributo *clients*. Entre a linha 34 e 38 aparece o construtor padrão vazio da classe.

Finalmente, a definição do método *addAddress* pode ser visto na linha 40. A anotação *@Action* é utilizada para identificar que método *addAddress* estará disponível na interface do usuário. A anotação *@Parameter* é utilizada para identificar o parâmetro do método com nome *description*. Para simplificar, os métodos padrões *toPrimaryDescription* e *toSecondaryDescription* do JustBusiness, bem como os métodos *get* e *set* foram omitidos desta explicação.

JustBusiness usa um mecanismo de geração de código que, com as informações obtidas a partir do mapeamento classes de negócio, gera e modifica automaticamente todas as classes, interfaces, arquivos de recursos e configurações necessárias para a produção e implantação de um aplicativo Android.

Figura 12 representa exemplos de telas criadas a partir de informações obtidas a partir do mapeamento da classe *Person* descrita na Figura 11. No lado esquerdo da figura, encontra-se a tela de inserção, na qual os atributos são sequencialmente organizados de acordo com a ordem determinada, enquanto no lado direito encontra-se a tela de detalhes, que possui a mesma organização de atributos, mas apresentando no topo da tela uma barra de menu com uma lista dos métodos de objeto exibidos de acordo com o *label* e ordem determinados.

Figura 12 – Exemplo de aplicação negocial Android gerada pelo JustBusiness



Fonte: Elaborado pelo autor

4.7 CONSIDERAÇÕES FINAIS

Esta seção apresentou o *framework* JustBusiness, abordando questões relativas à estrutura do *framework*, sua arquitetura, além de explorar detalhes de implementação e exemplos de sua utilização.

No próximo capítulo será apresentada a abordagem proposta JustModeling, apresentando detalhes técnicos, sua utilização e integração com o *framework* JustBusiness.

5 ABORDAGEM JUSTMODELING

Nesta seção é detalhada a abordagem proposta JustModeling (FREITAS; MAIA, 2016a), que visa facilitar e acelerar a modelagem e criação de aplicações negociais Android através de uma integração de técnicas de MDE com o *framework* JustBusiness.

O núcleo da abordagem JustModeling é o JBModel, uma ferramenta de modelagem gráfica desenvolvida usando EuGENia [1], que consiste em uma ferramenta que gera automaticamente os recursos necessários para implementar um editor a partir de informações de um metamodelo Ecore [2].

A abordagem consiste em três atividades principais, como ilustrado pela Figura 13: primeiramente, os usuários (programadores, analistas ou projetistas) modelam as classes de negócio e relacionamentos de sua aplicação Android na forma de um diagrama de classe UML usando a ferramenta JBModel. Além disso, o usuário pode definir em cada modelo de classes os valores dos parâmetros de anotações usados pelo JustBusiness, abstraindo assim, detalhes de implementação do *framework*.

Depois de modelar as classes de negócio, o usuário solicita ao JBModel a geração do código-fonte, com base no JustBusiness, para cada uma das entidades, incluindo as suas anotações. Esse procedimento é realizado por um conjunto de transformações M2T implementadas usando a ferramenta Acceleo [3].

Finalmente, as classes de negócio implementadas e anotadas serão utilizadas como entrada para o *framework* JustBusiness que, a partir dessa entrada, gera todas as interfaces de usuário, código de persistência e recursos de aplicações, produzindo um aplicativo Android completo.

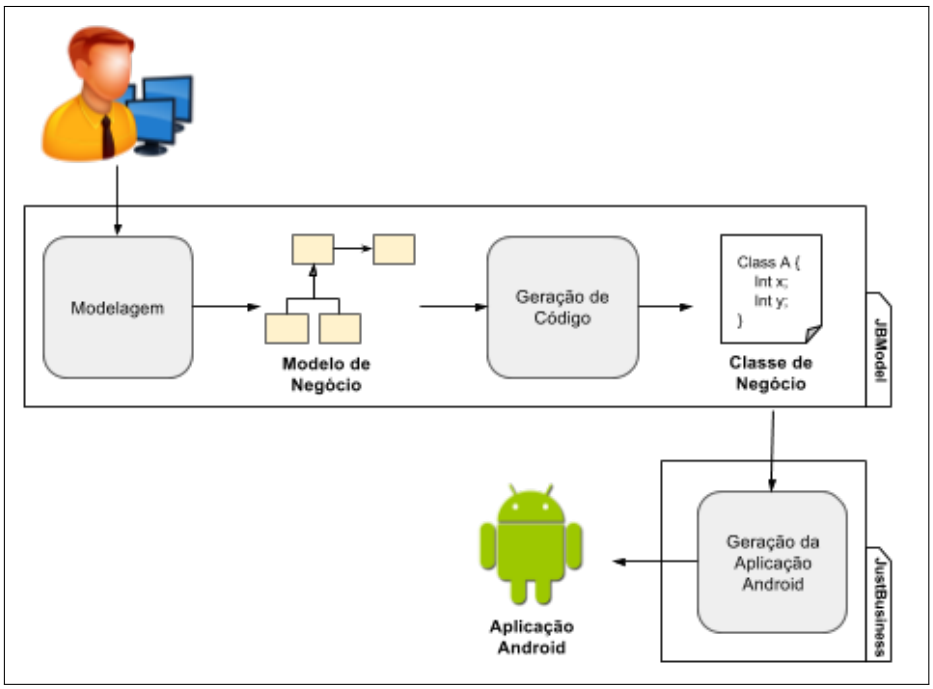
Nas próximas seções serão detalhados o metamodelo utilizado pelo JBModel, além dos principais componentes de modelagem gráfica da ferramenta.

5.1 METAMODELO

Os modelos criados utilizando o JBModel devem estar em conformidade com seu metamodelo, que foi concebido como uma versão resumida do metamodelo Ecore padrão, mantendo-se a estrutura mais básica do diagrama original, mas com a adição de novas espe-

[1] http://www.eclipse.org/epsilon/doc/eugenia/
[2] https://eclipse.org/modeling/emf/
[3] http://www.eclipse.org/acceleo/

Figura 13 – Abordagem Proposta JustModeling



Fonte: Elaborado pelo autor

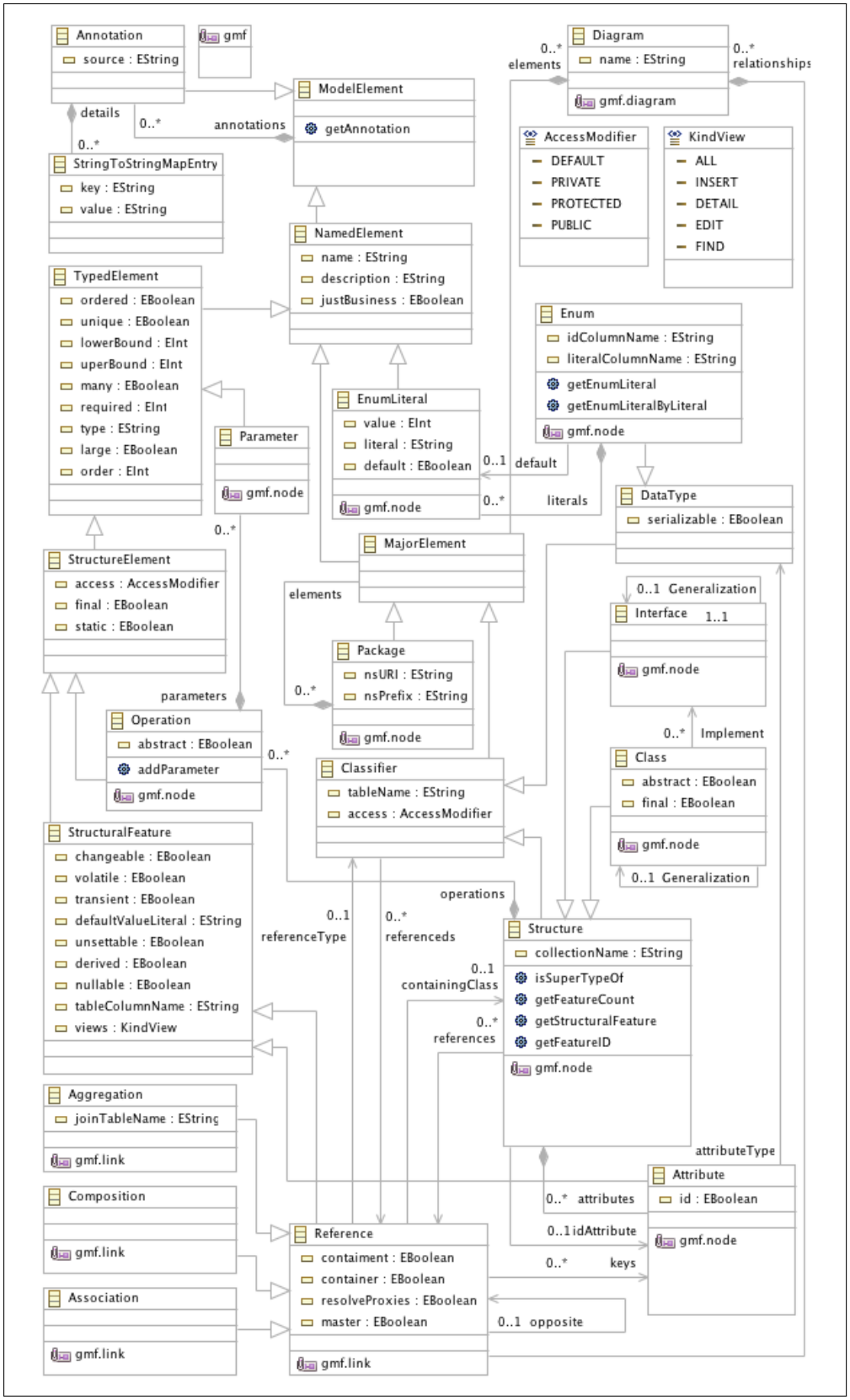
cializações para alguns componentes, tais como o elemento *Reference* e suas especializações *Aggregation*, *Composition* e *Association*.

A estrutura do metamodelo construído foi projetada com finalidade de representar domínios de aplicações suportados pelo *framework* JustBusiness, uma vez que a abordagem proposta representa o acréscimo de um nível de abstração a mais sobre o JustBusiness. O metamodelo em questão encontra-se representado na Figura 14.

O componente principal do metamodelo é a classe *Diagram*, que corresponde ao elemento que representa o diagrama de classe a ser concebido. O diagrama é constituído por duas superclasses abstratas que representam coleções de elementos: *MajorElement* e *Reference*. O primeiro é especializado pelos elementos *Package* e *Classifier*, que engloba elementos *Structure* e *DataType*. Já o segundo é uma classe genérica que agrupa os tipos de relacionamento disponíveis entre elementos *Structure* e *Classifier* e é especializada pelas subclasses concretas *Association*, *Aggregation* e *Composition*. *Structure*, por sua vez, é uma classe abstrata e engloba a estrutura e comportamento comum as suas subclasses *Class* e *Interface*.

A classe *Package* é uma classe concreta, que consiste em uma coleção de objetos de subclasses oriundas de *MajorElement*. Essa classe implementa o padrão de projeto Composite (GAMMA *et al.*, 1995), uma vez que pode representar pacotes contendo apenas classes e enumerações ou outros pacotes. *Class* é o elemento chave do metamodelo e representa a classe

Figura 14 – Metamodelo utilizado pelo JBModel



Fonte: Elaborado pelo autor

de negócio que será modelada contendo coleções de métodos, atributos e referências. Outra entidade também de grande importância é *Interface*, que assim como *Class*, possui coleções de métodos, atributos e referências. Finalmente, a classe abstrata *DataType* é especializada por *Enum*, que é responsável por representar enumerações e é composta por elementos do tipo *EnumLiteral*.

ModelElement é uma superclasse abstrata que agrega características gerais dos elementos do modelo e é especializada em duas subclasses: *Annotation*, que insere anotações no modelo e tem associação com a classe *StringToStringMapEntry*; e *NamedElement*, que representa os elementos do modelo que possuem um atributo de nome, uma vez que alguns elementos não têm esse atributo, como a classe *Annotation*.

Há três subclasses de *NamedElement*: *TypedElement*, *EnumLiteral* e *MajorElement*. *TypedElement* representa os elementos que têm um tipo de dados associado, como um retorno de método ou um tipo de parâmetro ou attributo. *TypedElement* é extendida pelas classes *Parameter*, que representa parâmetros em métodos de classe e pela classe abstrata *StructureElement*, que reune informações sobre modificadores de acesso; e é especializada pelas classes *Operation*, que representa métodos de classe; e *StructuralFeature*, classe genérica herdada por *Attribute* e *Reference*. Por fim, *Attribute* descreve atributos de classe.

5.2 MODELAGEM

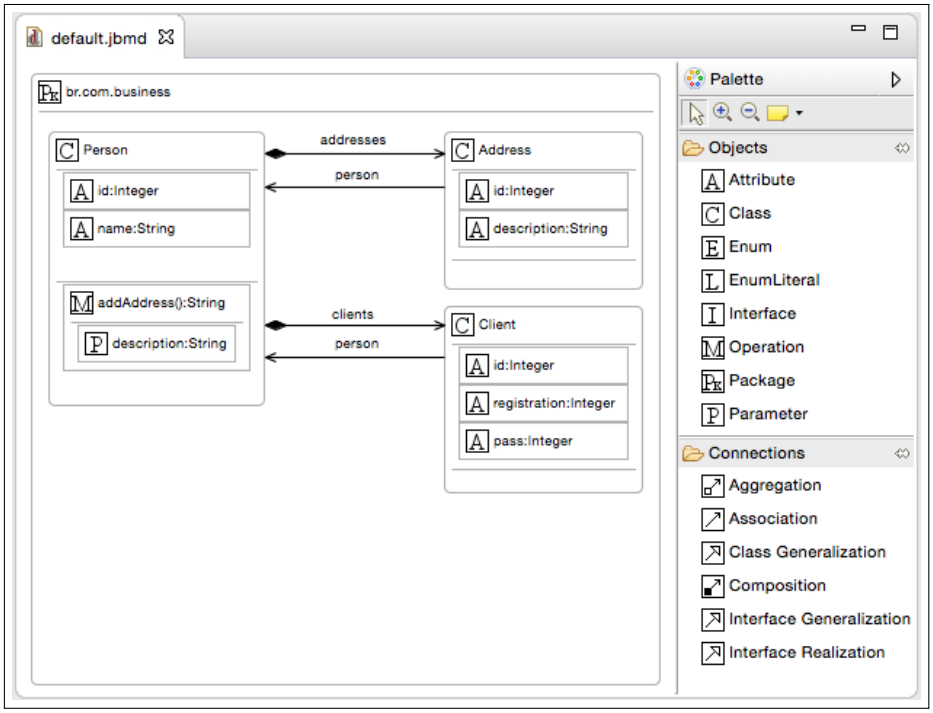
Como mencionado anteriormente, JBModel é uma ferramenta gráfica para modelar classes de negócio estruturadas como um diagrama de classes. Para isso, JBModel fornece uma sintaxe concreta gráfica, que combina elementos visuais com os elementos que compõem o seu metamodelo para criar modelos.

A sintaxe gráfica foi desenvolvida utilizando a ferramenta *Graphical Modeling Framework* (GMF), juntamente com o *Eclipse Modeling Framework* (EMF) e EuGENia [4]. EuGENia proporciona uma facilidade a partir da qual é possível mapear e associar cada elemento específico pertencente ao metamodelo a um componente visual de acordo com as seguintes categorias: diagrama (*@gmf.diagram*), nó (*@gmf.node*) e ligação (*@gmf.link*). A partir desse mapeamento, GMF pode gerar um ambiente visual para a construção de modelos.

Figura 15 mostra a ferramenta correspondente ao ambiente de modelagem. No lado direito, existem os elementos visuais que podem ser usados para criar os modelos de negócios,

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

Figura 15 – Ambiente de modelagem JBModel



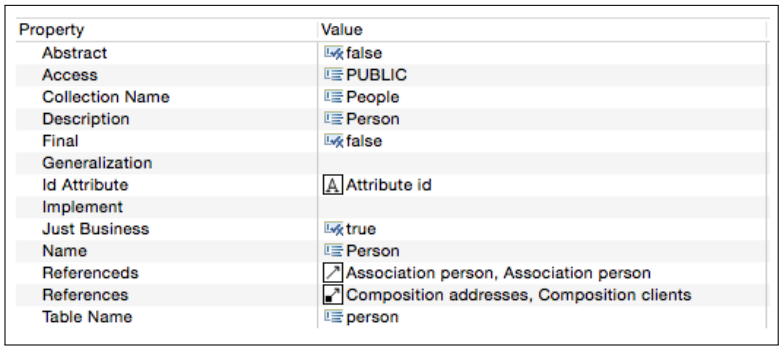
Fonte: Elaborado pelo autor

divididos em duas categorias: *Objects* e *Connections*. No primeiro caso, o elemento *Attribute* representa atributos de uma classe, a qual é representada pelo elemento *Class*. O elemento *Enum* representa as enumerações, enquanto *EnumLiteral* é usado para compor os literais que fazem parte de objetos do tipo *Enum*. O elemento *Interface* é utilizado para representar interfaces que podem ser implementadas por classes. *Opperation* é usado para definir métodos de classe, cujos parâmetros são representados pelo elemento *Parameter*. Finalmente, *Package* é usado para definir os pacotes de componentes, contendo objetos do tipo *Class*, *Enum*, *Interface* e *Package*. Neste último, os elementos *Aggregation*, *Composition* e *Association* definem relações entre as classes, como seus nomes sugerem. Finalmente, o elemento *Class Generalization* define um relacionamento de generalização entre elementos do tipo *Class*, *Interface Generalization* define relacionamento de generalização entre elementos *Interface* e *Interface Realization* define relacionamento de realização entre uma *Class* e um ou mais objetos do tipo *Interface*.

No lado esquerdo da Figura 15, encontra-se uma instância de modelo que consiste em um pacote chamado *br.com.macc* que contém três classes: *Person*, *Address* e *Client*. A classe *Person* possui os atributos de *id* e *name*, o método *addAddress* e uma relação de composição com as classes *Address* e *Client*. A classe *Address* tem os atributos *id* e *description* e uma associação com a classe *Person*. Finalmente, a classe *Client* tem os atributos *id*, *registration* e *pass*, e uma associação com a classe *Person*.

Apesar dos recursos fornecidos pelo JBModel para modelar as classes de negócio e suas relações, algumas informações importantes necessárias à geração de código, particularmente em relação às anotações de código, não pode ser definida utilizando apenas os componentes gráficos do modelo. Assim, alguns desses valores de atributo devem ser definidos usando a tela de exibição de propriedades para os elementos gráficos do JBModel.

Figura 16 – Propriedades da classe Person



Fonte: Elaborado pelo autor

A tela de visualização de propriedades para os atributos de classe *Person* encontra-se ilustrada na Figura 16. A propriedade *Abstract* indica se a classe está configurada para ser abstrata ou concreta. *Access* indica o modificador de acesso. *Collection Name* e *Description* correspondem ao valor passado para os atributos *collectionName* e *name*, respectivamente, pertencentes à anotação @*Entity*. *Final* indica se a classe é do tipo final, a qual não pode ser especializada. @*Id* é usado para informar que atributo é o elemento chave-primária da classe. *Just Business* indica se será gerado código com anotações para esta classe. Finalmente, a propriedade *Table Name* corresponde ao valor a ser definido no parâmetro *name* da anotação @*Table*. Os outros elementos que não foram mencionadas, como *Generalization*, *Implement*, *Referenceds* e *References*, são geralmente definidos utilizando a tela de modelagem, em vez da tela de visualização de propriedades.

5.3 GERAÇÃO DE CÓDIGO

O mecanismo de geração de código-fonte fornecido pelo JBModel foi construído usando Acceleo integrado com Eclipse Modeling Framework (EMF). Acceleo é uma ferramenta para a transformação de modelos em texto que fornece recursos na forma de arquivos *template*, que são usados para definir conjuntos de regras para a conversão de cada objeto pertencente ao modelo em código-fonte. No contexto deste trabalho, Acceleo é usado para criar classes Java a

partir das classes de negócio modelados pelo usuário.

O mecanismo de geração de código assume que a ordem em que os atributos e métodos de classe serão apresentados na tela é a mesma em que esses elementos foram configurados no modelo onde as classes de negócio foram projetadas. Por exemplo, considerando a classe *Person* representada pela Figura 15 e seu código gerado ilustrado pela Figura 11, vemos que atributos *id* e *name* têm em sua anotação *@Attribute* os valores do parâmetro *order* definidos como 0 e 1, respectivamente, que é a mesma ordem em que os atributos foram inseridos no modelo.

Sobre as telas em que os atributos serão exibidos na aplicacão, a estratégia de geração de código define por padrão que o atributo *id* (chave primária) para ser mostrado apenas na tela de detalhes (*KindView.DETAIL*), enquanto os outros serão apresentados em todas as telas (*KindView.ALL*). Apesar de haver uma definição padrão, o usuário tem a possibilidade de modificar isso, através do atributo *Views* contido no elemento *Attribute*, onde ele pode definir explicitamente em quais telas cada atributo deverá ser exibido.

As anotações sobre relacionamentos entre classes, como o *@OneToMany*, são gerados automaticamente a partir das relações modeladas pelo usuário no JBModel. Por exemplo, na Figura 11, as linhas 25 e 26 são geradas a partir da relação de associação entre a classes *Person* e *Address*, como modelado na Figura 15.

5.3.1 Geração de Código com Acceleo

Acceleo é uma ferramenta de código aberto disponibilizada na forma de um plugin para o Eclipse IDE, tendo como objetivo a geração de código e artefatos de texto a partir de instâncias de modelos, promovendo assim transformações M2T. De acordo com (WEBER; JANUARIO; MATOS, 2010), o Acceleo tem como processo as seguintes etapas: prototipação do *template*, validação através dos critérios de qualidade adotados, e conversão em um gerador, módulo que encapsula os *templates* de transformação.

Segundo (MUSSET *et al.*, 2006), a ferramenta Acceleo foi projetada para aumentar a produtividade no desenvolvimento de *software*, possibilitando a geração de arquivos usando módulos UML, MOF e EMF, oferecendo uma linguagem baseada em modelos para a definição de modelos de geração de código.

O JBModel faz uso de recursos disponibilizados pelo Acceleo, como módulos, *templates* e *queries*, com objetivo de transformar os modelos de negócio construídos utilizando

o ambiente gráfico em código-fonte de classes codificadas em linguagem Java e configuradas com as anotações fornecidas pelo *framework* JustBusiness. A Figura 25 exemplifica o módulo de geração de código para elementos do tipo *Class*. O código completo das sub-rotinas que aparecem na Figura 25 encontra-se detalhado nos apêndices, enquanto o projeto completo contendo todos os códigos encontra-se disponível na página do JBModel[5].

Figura 17 – Módulo de Geração de Código para o tipo *Class*

```
1  [comment encoding = UTF-8 /]
2  [module generateClass('http://www.example.org/JBModel')]
3
4  [import org::eclipse::acceleo::module::JBModel::common::queries/]
5  [import org::eclipse::acceleo::module::JBModel::common::generateClassAttributes/]
6  [import org::eclipse::acceleo::module::JBModel::common::generateClassReferences/]
7  [import org::eclipse::acceleo::module::JBModel::common::generateClassConstructor/]
8  [import org::eclipse::acceleo::module::JBModel::common::generateClassGettersSetters /]
9  [import org::eclipse::acceleo::module::JBModel::common::generateClassMethods /]
10
11 [import org::eclipse::acceleo::module::JBModel::common::generateImplementClassMethods /]
12 [import org::eclipse::acceleo::module::JBModel::common::generateImplementInterfaceMethods /]
13
14 [import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessImports /]
15 [import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessExtending /]
16 [import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessMethods /]
17 [import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationEntity/]
18 [import org::eclipse::acceleo::module::JBModel::common::jb::persistence::generateAnnotationTable /]
19 [import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessGeneralImports /]
20
21 [template public generateClass(aClass : Class)]
22 [file (aClass.getFileName(), false, 'UTF-8')]
23 [aClass._package.getPackage()/]
24
25 [aClass.generateJustBusinessImports()/]
26
27 [aClass.generateJustBusinessGeneralImports()/]
28 [aClass.generateAnnotationTable()/]
29 [aClass.generateAnnotationEntity()/]
30 [aClass.getClassModifiers()/]class [aClass.getClassifierName()/] [aClass.generateJustBusinessExtending()/] {
31     [aClass.generateAttributes()/]
32     [aClass.generateReferences()/]
33     [aClass.generateEmptyConstructor()/]
34     [aClass.generateMethods()/]
35     [aClass.generateImplementClassMethods()/]
36     [aClass.generateImplementInterfaceMethods()/]
37     [aClass.generateJustBusinessMethods()/]
38     [aClass.generateGettersSetters()/]
39 }
40 [/file]
41 [/template]
42
```

Fonte: Elaborado pelo autor

Na Figura 25, a linha 2 apresenta a declaração e início do código do módulo *generateClass*. Entre as linhas 4 e 19 encontram-se as importações de outros módulos necessários os quais o módulo atual depende. Na linha 21 encontra-se a declaração e o início do *template generateClass*, especificando que esse *template* será utilizado por elementos do tipo *Class*. Na linha 22, encontra-se a declaração de criação do arquivo de texto (classe java), no qual todo o texto escrito entre as linhas 22 e 40 fará parte do corpo do arquivo criado. A linha 23 inclui o código relativo à inclusão das importações. Nas linha 25 e 27 encontram-se códigos relativos à inclusão das importações relacionadas ao JustBusiness. As linhas 27 e 28 incluem o código das anotações *@Table* e *@Entity*, respectivamente. Na linha 30 encontra-se a declaração da classe a ser criada, especificando seu nome, a superclasse que ela estende e as interfaces que ela

[5] https://jbframework.wordpress.com/download/

implementa. Nas linhas 31 e 32 encontram-se a declaração de atributos (atributos e referências), enquanto que na linha 33 está o código relativo ao construtor padrão da classe. Na linha 34 encontra-se o código de escrita de operações ou métodos. Nas linhas 35 e 36 encontram-se o código de escrita de métodos provenientes de herança de superclasses e de implementação de interfaces, respectivamente. Na linha 37 está o código de escrita dos métodos padrões do JustBusiness, provenientes da class *JBEntity*. Por fim, a linha 38 contém o código de escrita dos métodos *get* e *set*, e a linha 39 encerra o corpo da classe.

5.4 CONFIGURANDO O AMBIENTE E EXECUTANDO UMA APLICAÇÃO ANDROID COM JBMODEL E JUSTBUSINESS

Para usar JBModel, um conjunto de passos deve ser seguido, como representado na Figura 18. Inicialmente, é necessário instalar no Eclipse o plugin JBModel, disponível para *download* na página web do *framework* JustBusiness [6]. Com o ambiente Eclipse configurado corretamente, o próximo passo consiste em criar um projeto genérico vazio, no qual um arquivo de modelo JBModel é criado e usado para projetar os modelos de negócio da aplicação Android. Depois, o código-fonte Java é gerado a partir das classes de negócio modeladas. Nessa fase, o conjunto de classes gerado está pronto para ser usado como entrada para o *framework* JustBusiness.

Na próxima etapa, o projeto vazio JustBusiness deve ser importado utilizando a IDE que foi escolhida. Em seguida, as classes de negócio geradas pelo JBModel devem ser inseridas no diretório de código fonte do projeto importado. Caso a IDE escolhida para se trabalhar seja o Eclipse, esse diretório é a pasta *src* que encontra-se na raiz no projeto. Caso esteja sendo utilizado o Android Studio, o diretório a ser utilizado é o *app/src/main/java*, que também encontra-se na raiz no projeto.

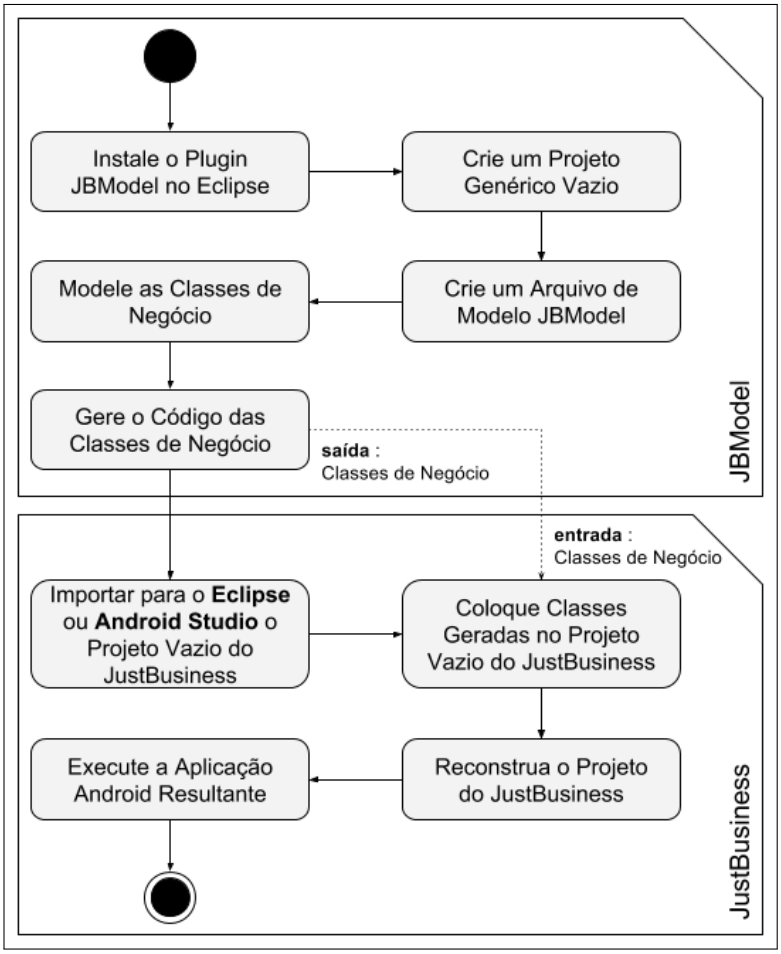
Por fim, é necessário reconstruir o projeto JustBusiness, utilizando os comandos de *clean* e *rebuild*, e executar o projeto para obter o aplicativo Android.

5.5 CONSIDERAÇÕES FINAIS

Esta seção apresentou a abordagem proposta JustModeling, que combida técnicas de Engenharia Dirigida por Modelos com o *framework* baseado em Naked Objects JustBusiness. A ferramenta de modelagem JBModel foi descrita, bem como detalhes técnicos da abordagem,

[6] https://jbframework.wordpress.com/

Figura 18 – Configurando um projeto com JBModel



Fonte: Elaborado pelo autor

como metamodelo, geração de código, e como se dá sua integração com o JustBusiness.

No próximo capítulo serão apresentados dois estudos de caso desenvolvidos para realizar a avaliação da abordagem JustModeling em relação ao JustBusiness e desenvolvimento tradicional.

6 AVALIAÇÃO

Para demonstrar o ganho de produtividade ocasionado pelo uso da abordagem proposta, foi realizado o desenvolvimento de aplicações negociais para Android visando demonstrar o impacto da utilização do JustModeling em relação ao desenvolvimento tradicional, i.e., sem nenhum *framework*. Como os projetos também poderiam ser construídos utilizando apenas o *framework* JustBusiness, o JustBusiness também será considerado na análise. O primeiro descreve a criação de uma aplicação simples envolvendo quatro entidades, enquanto o segundo detalha o desenvolvimento de uma aplicação mais complexa envolvendo 21 entidades.

6.1 METODOLOGIA UTILIZADA

A realizacão dessa avaliação contou com um único desenvolvedor, com três anos de experiência com desenvolvimento Android, o qual o qual desenvolveu as duas aplicações de três maneiras: com o JustModeling, apenas com o JustBusiness, e com o desenvolvimento tradicional, i.e., sem utilizar nenhuma tecnologia a mais.

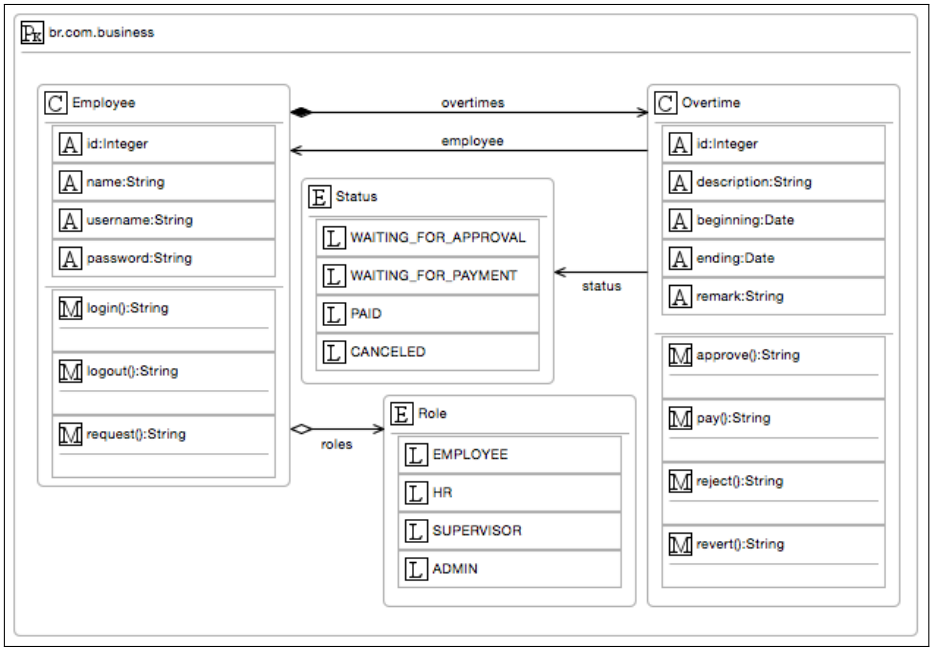
6.2 PRIMEIRA APLICAÇÃO

A aplicação selecionada foi a mesma usada em (FREITAS; MAIA, 2015; FREITAS; MAIA, 2016b) e consiste de um *software* para solicitação e aprovação de horas extras de serviço. O cenário começa com o funcionário realizando a solicitação de realização de horas extras, indicando a justificativa e a data inicial e final. Os pedidos são primeiramente revistos pelo supervisor, que tanto pode autorizar ou rejeitá-las. Os pedidos autorizados serão analisados pelo setor de Recursos Humanos (RH) para calcular e aprovar o pagamento das horas. Se o setor de RH tiver algumas perguntas, o pedido pode voltar ao supervisor.

O primeiro passo é a modelagem das classes de negócio da aplicação usando a ferramenta JBModel. A tarefa de modelagem tem início com a criação de um elemento *Package* identificado como *br.com.company.overtime* que contém quatro entidades, duas enumerações (*Status* e *Role*) e duas classes de negócio (*Employee* e *Overtime*). A enumeração *Role* consiste em quatro elementos do tipo *EnumLiteral*: *HR*, *ADMIN*, *EMPLOYEE* e *SUPERVISOR*, enquanto a enumeração *Status* tem em sua composição os elementos do tipo *EnumLiteral* *WAITING_FOR_APPROVAL*, *WAITING_FOR_PAYMENT*, *PAID* e *CANCELED*.

A classe *Employee* possui os atributos *id*, *name*, *username* e *password*, e os métodos

Figura 19 – Modelo da aplicação de Horas Extras



Fonte: Elaborado pelo autor

de *login*, *logout* e *request*, enquanto a classe *Overtime* tem os atributos de *id*, *description*, *begging*, *ending* e *remark*, e os métodos *approve*, *pay*, *cancel* e *revert*.

Também podem ser identificados no diagrama da Figura 19 algumas relações entre entidades: um relacionamento de composição entre a classe *Employee* e *Overtime*, uma associação entre *Employee* e *Overtime*, uma associação entre *Overtime* e *Status* e, por último, uma associação entre *Employee* e *Role*. O modelo resultante é ilustrado pela Figura 19. Em seguida, os parâmetros não visuais de cada uma das classes foram definidos na tela seção de propriedades e o código para as classes foi gerado de maneira automatizada. Para ilustrar, o código-fonte da classe *Overtime* encontra-se detalhado na Figura 20.

Na Figura 20, na linha 1, encontramos a descrição do pacote que a classe está contida, enquanto que entre as linhas 3 e 9 encontra-se o código refrente às importações necessárias para a classe *Overtime*. Nas linhas 11 e 12 encontra-se o código de mapeamento da classe *Overtime* por meio de anotações @*Table* e @*Entity*. Na linha 14 encontra-se a assinatura da classe *Overtime*, juntamente com o seu relacionamento de generalização com a superclasse *JBEntity*. Entre as linhas 14 e 35, destaca-se a definição dos atributos *id*, *description*, *begging*, *ending* e *remark* com suas anotações devidamente configuradas.

Entre as linhas 37 e 45, encontram-se as definições das referências para outras entidades. A variável *employee* representa uma associação com a classe *Employee* e *status*, uma associação com a enumeração *Status*. As referências são vistas na mesma maneira como

Figura 20 – Código fonte da classe Overtime

```java
package br.com.company.overtime;

import org.jb.annotation.model.*;

@Table(name="overtime")
@Entity(label="Overtime", collectionLabel="Overtimes")
public class Overtime extends JBEntity {
    @Attribute(order=0, name="Identifier", views={KindView.DETAIL})
    @Id
    @Column(name="id_overtime", nullable=false, unique=false)
    private Integer id;

    @Attribute(order=1, name="Description", views={KindView.ALL})
    @Column(name="description", nullable=false, unique=false)
    private String description;

    @Attribute(order=2, name="Beginning", views={KindView.ALL})
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="beginning", nullable=false, unique=false)
    private Date beginning;

    @Attribute(order=3, name="Ending", views={KindView.ALL})
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="ending", nullable=false, unique=false)
    private Date ending;

    @Attribute(order=4, name="Remark", views={KindView.ALL})
    @Column(name="remark", nullable=false, unique=false)
    private String remark;

    @Attribute(order=5, name="employee", views={KindView.ALL})
    @ManyToOne
    @JoinColumn(name="employee", nullable=false, unique=false)
    private Employee employee;

    @Attribute(order=6, name="Status", views={KindView.ALL})
    @ManyToOne
    @JoinColumn(name="id_status", nullable=false, unique=false)
    private Status status;

    public Overtime() {
        super();
        employee = new Employee();

        status = Status.WAITING_FOR_APPROVAL;
    }

    /**Class Methods*/

    @Action(order=0, name="Approve")
    public String approve() {
        //TODO
        return null;
    }
    @Action(order=1, name="Pay")
    public String pay() {
        //TODO
        return null;
    }
    @Action(order=2, name="Reject")
    public String reject() {
        //TODO
        return null;
    }
    @Action(order=3, name="Revert")
    public String revert() {
        //TODO
        return null;
    }

    /**JustBusiness Default Methods*/

    /**Get and Set Methods*/

}
```

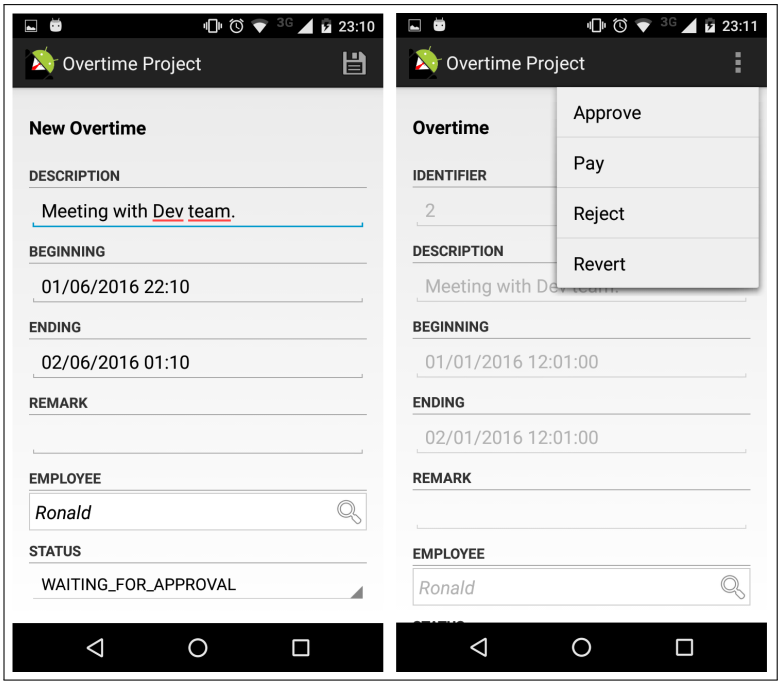
Fonte: Elaborado pelo autor

os atributos na classe Java e são configurados com anotações similares. Entre as linhas 47 e 52, tem-se o construtor padrão vazio da classe. Por fim, entre as linhas 56 e 75, encontra-se a especificação dos métodos de classe *approve*, *pay*, *cancel* e *revert*.

O conjunto de classes de negócio produzidos por JBModel através do mecanismo de geração de código é então colocado dentro da pasta *src* dentro de um projeto vazio equipado com JustBusiness no Eclipse. Ao construir o projeto JustBusiness, toda a estrutura necessária do aplicativo é gerada, e então o aplicativo Android pode ser executado.

Em relação à classe *Overtime*, a Figura 21 mostra duas telas: uma para a criação de objetos de horas extras (esquerda) e a outra para visualizar os detalhes e manipular objeto Overtime (à direita), permitindo que o usuário possa aprovar, pagar, rejeitar ou reverter solicitação de horas extras.

Figura 21 – Telas de criação e detalhes de objetos do tipo Overtime



Fonte: Elaborado pelo autor

Portanto, é possível afirmar que a abordagem proposta foi aplicada com sucesso para este estudo de caso, onde a partir da modelagem das classes de negócio foi obtida a geração do aplicativo Android.

6.2.1 Análise dos Resultados

Para avaliar os benefícios do uso JustModeling, uma experiência comparativa foi realizada, em que foi desenvolvido um projeto do mesmo aplicativo Android com base no cenário descrito no estudo de caso detalhado anteriormente. O cenário utilizado é o mesmo que foi utilizado na avaliação do *framewrok* JustBusiness (FREITAS; MAIA, 2015; FREITAS; MAIA, 2016b).

Depois de modelar as classes e gerar a aplicação com JustModeling, verificou-se que no projeto desenvolvido com a abordagem proposta JustModeling, o desenvolvedor necessitou de 18 minutos para concluir o desenvolvimento da aplicação, o que representa uma redução de 34% em relação aos 27 minutos gastos pelo JustBusiness e 97,5% em relação ao tempo de 12 horas de trabalho quando utilizado o Desenvolvimento Tradicional. Em termo de quantidade de arquivos criados pelo desenvolvedor, com JustModeling foi necessário criar apenas 1 arquivo, o que representa uma redução de 75% em relação aos 4 arquivos criados utilizando somente o JustBusiness e 98,5% em relação aos 66 arquivos criados no Desenvolvimento Tradicional. A quantidade de linhas escritas não é relevante nessa comparação, uma vez que na tarefa de modelagem não há codificação baseada em linhas. Os dados comparativos para os dois projetos desenvolvidos encontram-se detalhados na Tabela 6.

Tabela 6 – Análise comparativa do desenvolvimento tradicional, JustBusiness e JustModeling

Tipo de Desenvolvimento	Tradicional	JustBusiness	JustModeling
Tempo	12 horas	27 minutos	18 minutos
Linhas escritas pelo Desenvolvedor	4315	376	0
Arquivos criados pelo Desenvolvedor	66	4	1
Total de arquivos no Projeto	66	66	67

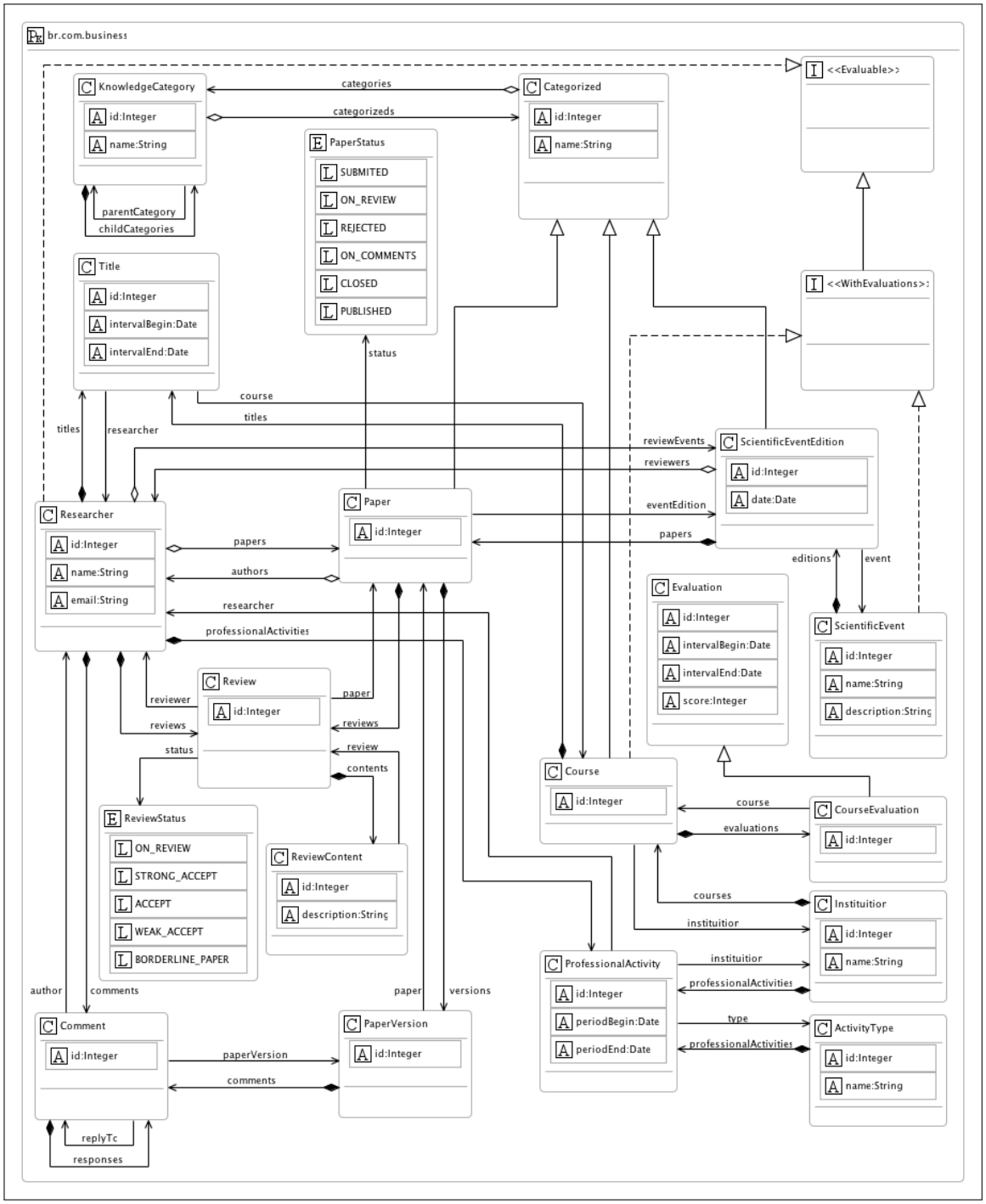
Fonte: Produzido pelo autor

6.3 SEGUNDA APLICAÇÃO

O projeto de aplicação utilizado nesse estudo de caso foi retirado de (TORRES, 2009) e consiste de um sistema de avaliação para produção científica. De acordo com o autor, esse sistema tem como objetivo especificar uma estrutura única de informações, permitindo a implementação de um processo único para avaliar pesquisadores, publicações científicas, eventos

e instituições de ensino. A estrutura das classes de negócio que compõem o modelo de negócio dessa aplicação, modelada utilizando a ferramenta JBModel, é ilustrada na Figura 22.

Figura 22 – Modelo da aplicação de Submissão de Artigo em Evento Científico



Fonte: Elaborado pelo autor

Na Figura 22, a classe *Categorized* é uma classe abstrata que abrange elementos que podem ser categorizados na forma de uma árvore de conhecimento e especializada pelas subclasses *Paper*, que representa artigos científicos; *Course*, que representa cursos e *Scienti-*

ficEventEdition, que corresponde a edições de eventos científicos ou de periódicos. A classe *KnowledgeCategory* representa categorias de áreas conhecimento, as quais podem se relacionar umas com as outras. *ScientificEvent* representa eventos científicos ou revistas.

A interface *Evaluable* representa os serviços de avaliações disponíveis, enquanto *WithEvaluations*, uma especializacão de *Evaluable*, permite o armazenamento de um histórico de avaliações. A classe *Researcher* presenta os pesquisadores e a classe *Title*, os títulos por eles adquiridos, como mestrado e douturado.

A classe *PaperVersion* representa a versão de um artigo, enquanto *Review* representa uma revisão referente a uma versão de um artigo. *Comment* representa os comentários realizados sobre uma versão de um artigo. *ReviewContent* representa o conteúdo de uma revisão. As enumerações *PaperStatus* e *ReviewStatus* representam respectivamente as situações em que se encontram um *Paper* e um *Review*.

Evaluation é uma classe abstração que representa avaliações e é especializada por *CourseEvaluation*, que consiste na avaliação de um curso. A entidade *Institution* representa instituições ligadas às atividades de ensino e pesquisa. *ProfessionalActivity* descreve as atividades que um pesquisador desenvolve em uma instituição. Por fim, *ActivityType* representa os tipos de atividades que podem ser exercidas.

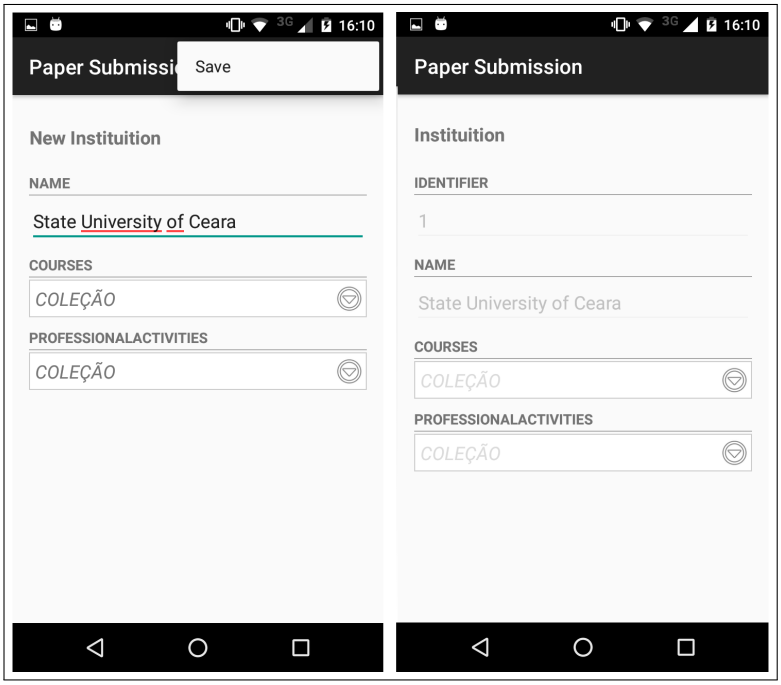
O conjunto de classes de negócio produzidos por JBModel através do mecanismo de geração de código é então colocado dentro de um projeto vazio equipado com JustBusiness. Ao construir o projeto JustBusiness, toda a estrutura do aplicativo é gerada.

Em relação à classe *Institution*, a Figura 23 mostra duas telas: uma para o cadastro de instituições (esquerda) e a outra para visualizar os detalhes e manipular um objeto instituição (à direita).

Ainda em relação à classe *Institution*, a Figura 24 mostra duas telas de listagem de instutuições: uma onde existe as opções cadastro e busca (esquerda) e a outra onde aparecem as opções para visualizar os detalhes, editar ou deletar um instituição (à direita).

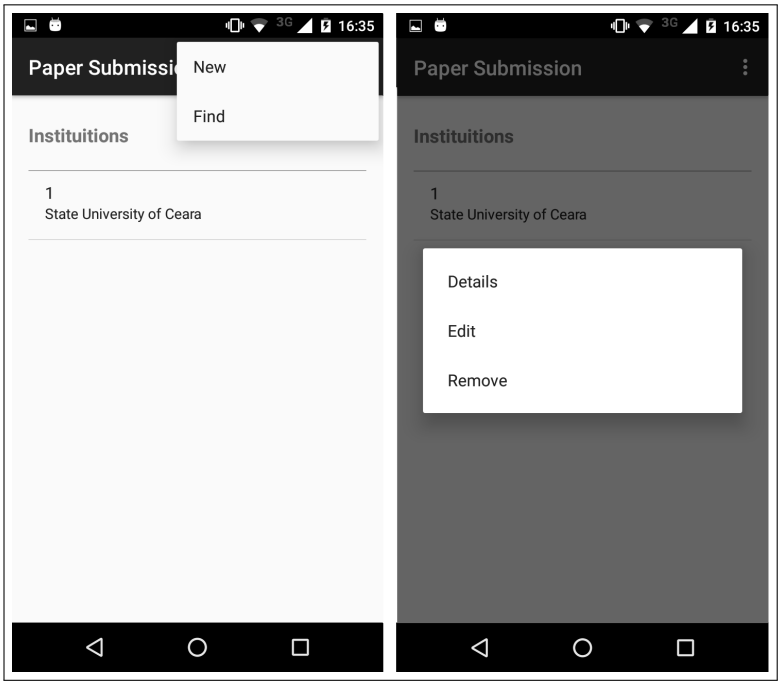
Portanto, assim como no primeiro estudo de caso, é possível afirmar que a abordagem JustModeling foi aplicada com sucesso, obtendo assim a partir da modelagem das classes de negócio, a aplicação Android.

Figura 23 – Telas de criação e detalhes de objetos do tipo Institution



Fonte: Elaborado pelo autor

Figura 24 – Telas Listagem de objetos do tipo Institution



Fonte: Elaborado pelo autor

6.3.1 Análise dos Resultados

Para avaliar os benefícios do uso JustModeling, uma experiência comparativa foi realizada, em que foi desenvolvido um projeto do mesmo aplicativo Android com base no cenário

descrito no estudo de caso detalhado anteriormente.

Depois de modelar as classes e gerar a aplicação com JustModeling, verificou-se que no projeto desenvolvido com a abordagem proposta JustModeling, o desenvolvedor necessitou de 1 hora e 27 minutos para concluir a aplicação, o que representa uma redução de 37,4% em relação às 2 horas e 19 minutos gastos pelo JustBusiness e 97,5% em relação ao tempo de 59 horas e 53 minutos de trabalho quando utilizado o Desenvolvimento Tradicional. Em termo de quantidade de arquivos criados pelo desenvolvedor, com JustModeling foi necessário criar apenas 1 arquivo, o que representa uma redução de 95,2% em relação aos 21 arquivos criados utilizando somente o JustBusiness e 99% em relação aos 451 arquivos criados no Desenvolvimento Tradicional. A quantidade de linhas escritas não é relevante nessa comparação, uma vez que na tarefa de modelagem não há codificação baseada em linhas. Os dados comparativos para os dois projetos desenvolvidos encontram-se detalhados na Tabela 7.

Tabela 7 – Análise comparativa do desenvolvimento tradicional, JustBusiness e JustModeling

Tipo de Desenvolvimento	Tradicional	JustBusiness	JustModeling
Tempo	59 horas e 53 minutos	2 horas e 19 minutos	1 hora e 27 minutos
Linhas escritas pelo Desenvolvedor	34560	1289	0
Arquivos criados pelo Desenvolvedor	451	21	1
Total de arquivos no Projeto	456	459	460

Fonte: Produzido pelo autor

6.4 AMEAÇAS À VALIDAÇÃO

Como ameaças a validação do estudo realizado neste capítulo, podemos indentificar: a quantidade de desenvolvedores utilizada em cada uma das aplicações desenvolvidas, uma vez que não é possível calcular uma média com os resultados obtidos por outros de desenvolvedores; o nível de experiência do desenvolvedor, o que pode ter favorecido alguma abordagem específica em razão do seu nível de conhecimento prévio; e a sequência em que os projetos foram desenvolvidos, uma vez que a partir do desenvolvimento do primeiro projeto em diante, o desenvolvedor adquiriu experiência sobre o domínio do problema.

6.5 CONSIDERAÇÕES FINAIS

Essa seção apresentou dois estudos de caso que foram realizados a fim de avaliar os benefícios da utilização da abordagem JustModeling em realção ao uso apenas do JustBusiness e do desenvolvimento tradicional. O primeiro estudo, mais simples, tinha como objetivo principal mostrar a utilização da abordagem proposta, enquanto o segundo, mais complexo, buscava demonstrar o real impacto da utilização do JustModeling em um cenário mais realista.

Os resultados obtidos ao término da realização de ambos os experimentos mostraram superioridade do JustModeling em relação às outras abordagens; indicando ganho de produtividade para desenvolvedor através da redução de tempo de desenvolvimento e esforço.

No próximo capítulo serão apresentadas as conclusões deste trabalho, bem como contribuições, limitações, trabalhos futuros e resultados alcançados em termos de publicações.

7 CONCLUSÃO

Este trabalho apresentou JustModeling, uma abordagem para facilitar e acelerar o desenvolvimento de aplicações negociais para Android através da integração de técnicas de MDE com um *framework* baseado em Naked Objects, o JustBusiness. Através do JBModel, uma ferramenta que permite modelar e gerar código fonte que é usado pelo *framework* JustBusiness, é possível criar de maneira automatizada uma aplicação Android estruturalmente completa, incluindo a geração de código de interface de usuário, mecanismos de persistência e todas as informações e recursos necessárias. Desta forma, o usuário só precisa modelar as classes de negócio da aplicação e as ferramentas envolvidas na abordagem executam as outras etapas que resultarão na construção da aplicação. Dois estudos de caso baseados em cenários encontrados na literatura foram realizados para avaliar o desempenho da abordagem JustModeling sobre JustBusiness e o desenvolvimento tradicional. Ambos revelaram que JustModeling melhorou o tempo de desenvolvimento, reduziu a quantidade de linhas de código escritas e arquivos criados pelo desenvolvedor, quando comparado com às outras duas abordagens.

7.1 CONTRIBUIÇÕES DO TRABALHO

Além da abordagem proposta JustModeling, também considera-se como contribuições resultantes deste trabalho:

- A ferramenta gráfica JBModel, utilizada para realização de modelagem gráfica de modelos de negócios e geração de código.
- O projeto base do JustBusiness disponibilizado para o ambiente desenvolvimento integrado Android Studio.
- Código-fonte dos estudos de caso realizados disponibilizados na página web do projeto JustBusiness.

7.2 LIMITAÇÕES

Acredita-se que os objetivos que foram definidos no início deste trabalho foram plenamente alcançados ao longo do seu desenvolvimento. Entretanto, algumas limitacões referentes à abordagem JustModeling, bem como suas ferramentas de apoio JBModel e JustBusiness, podem ser identificadas:

7.2.1 JustModeling

- Só permite modelar a estrutura das classes, enquanto o código dos corpos dos métodos precisa de ser inserido manualmente pelo programador depois da conclusão da atividade de geração de código pelo *JBModel*.
- O código das classes de negócio gerado pelo *JBModel* precisa ser copiado e inserido dentro do projeto do *JustBusiness*, não havendo assim um integração total entre ambas as ferramentas.

7.2.2 JBModel

- O ambiente gráfico de modelagem só permite modelar a estrutura das classes, não sendo possível capturar o comportamento dos métodos das classes.

7.2.3 JustBusiness

- Em sua versão atual é possível apenas a utilização de persistência local através da utilizacão da base de dados SQLite.
- Não há possibilidade de se realizar customizações na camada de apresentação da aplicação gerada pelo *framework*.
- Os únicos tipos de dados suportados nas classes de negócio são booleanos, números, textos, datas e tipos definidos pelo programador, não oferecendo suporte à imagens, vídeos ou dados de localizacão.
- Não existe mecanismo para validação de valores ou criação de máscaras em campos de formulários.

7.3 TRABALHOS FUTUROS

Ainda existem trabalhos que podem ser desenvolvidos no contexto da abordagem JustModeling, bem como suas ferramentas de apoio JBModel e JustBusiness, para dar segmento as suas evoluções. Como trabalhos futuros podem ser citados:

7.3.1 JustModeling

- Realizar um estudo comparativo da abordagem JustModeling em relação as ferramentas propostas em alguns trabalhos relacionados.

7.3.2 JBModel

- Evoluir a ferramenta *JBModel* para oferecer suporte à modelagem comportamental, proporcionando assim uma representação mais completa de aplicações negociais.
- Realizar um estudo com finalidade de avaliar a usabilidade da ferramenta de modelagem JBModel.

7.3.3 JustBusiness

- Disponibilizar configurações e mecanismos que possibilitem a aplicação utilizar persistência externa e operações remotas.
- Fornecer suporte a outros tipos de dados mais complexos, como imagens, vídeos localização.
- Introduzir mecanismos para realizar validações de valores e mascaras nos campos e dados em formulários.
- Disponibilizar recursos que permitam ao programador realizar a customização da interface de usuário.
- Realizar um estudo sobre uma melhor adequação e usabilidade das interfaces de usuário geradas pelo *framework*.

7.4 ARTIGOS PUBLICADOS

Ao longo do desenvolvimento do presente trabalho, foram elaborados e publicados quatro artigos científicos. Esses artigos serviram de base para o desenvolvimento da abordagem *JustModeling* e consequentemente para a concepção da presente dissertação. São eles:

- Fabiano Freitas and Paulo Henrique M. Maia. 2015. **JustBusiness: A Framework for Developing Android Applications Using Naked Objects**. In Proceedings of the 2015 IX Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS '15). IEEE Computer Society, Washington, DC, USA, 11-20. DOI=10.1109/SBCARS.2015.12. Esse artigo está disponível no endereço: <http://dl.acm.org/citation.cfm?id=2866956>
- Freitas F. and M. Maia P. (2016). **A Naked Objects based Framework for Developing Android Business Applications**. In Proceedings of the 18th International Conference on Enterprise Information Systems - Volume 1: ICEIS, ISBN 978-989-758-187-8, pages 348-358. DOI: 10.5220/0005872203480358. Esse artigo está disponível no endereço:

<http://scitepress.org/DigitalLibrary/PublicationsDetail.aspx?ID=IgJM8C9V/uA=&t=1>.

- Maia, Paulo Henrique Mendes and Freitas, Fabiano and Borges, Marcos and Muniz, Laryssa Lima and da Silva, Amanda Souza and Ximenes, Janaide. (2016). **Práticas e Experiências no Ensino de Engenharia Dirigida por Modelos**. iSys-Revista Brasileira de Sistemas de Informação. Vol.9. N.2. Esse artigo está disponível no endereço: <http://www.seer.unirio.br/index.php/isys/article/view/5441>.
- Fabiano Freitas and Paulo Henrique M. Maia. 2016. **JustModeling: an MDE Approach to Develop Android Business Applications**. In Proceedings of the 2016 VI Brazilian Symposium on Computing Systems Engineering. Esse artigo está disponível no endereço: <http://sbesc.lisha.ufsc.br/sbesc2016/proceedings2016/163172.pdf>.

REFERÊNCIAS

AMéNDOLA, F.; FAVRE, L. Adapting crm systems for mobile platforms: An mda perspective. In: **Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on**. [S.l.: s.n.], 2013. p. 323–328.

ANDROID Develop. 2016. <https://developer.android.com/guide/index.html>. [Online; accessed 31-July-2016].

ATKINSON, C.; KUHNE, T. Model-driven development: a metamodeling foundation. **Software, IEEE**, v. 20, n. 5, p. 36–41, Sept 2003. ISSN 0740-7459.

BENOUDA, H.; ESSBAI, R.; AZIZI, M.; MOUSSAOUI, M. Modeling and code generation of android applications using acceleo. **International Journal of Software Engineering and Its Applications**, v. 10, n. 3, p. 83–94, March 2016.

BIFFL, S.; MAETZLER, E.; WIMMER, M.; LUEDER, A.; SCHMIDT, N. Linking and versioning support for automationml: A model-driven engineering perspective. In: **2015 IEEE 13th International Conference on Industrial Informatics (INDIN)**. [S.l.: s.n.], 2015. p. 499–506. ISSN 1935-4576.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. **Model-Driven Software Engineering in Practice**. [S.l.]: Morgan & Claypool Publishers, 2012.

BRANDAO, M.; CORTES, M.; GONCALVES, E. Entities: A framework based on naked objects for development of transient web transientes. In: **Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En**. [S.l.: s.n.], 2012. p. 1–10.

BRANDãO, M.; CORTéS, M.; GONçALVES, E. Naked objects view language. In: **InfoBrasil**. [S.l.: s.n.], 2012.

COSTA, S. L. da. **Uma Abordagem Baseada em Modelos para Construção Automática de Interfaces de Usuário para Sistemas de Informação**. Dissertação (Mestrado) — Universidade Federal de Goiás, Instituto de Informática, 2011.

CRUZ, A. M. R. da; FARIA, J. P. A metamodel-based approach for automatic user interface generation. In: PETRIU, D. C.; ROUQUETTE, N.; HAUGEN, (Ed.). **Model Driven Engineering Languages and Systems**. Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6394). p. 256–270. ISBN 978-3-642-16144-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-16145-2_18>.

DEURSEN, A. V.; KLINT, P. Domain-specific language design requires feature descriptions. **CIT. Journal of computing and information technology**, SRCE-Sveučilišni računski centar, v. 10, n. 1, p. 1–17, 2002.

ELLEUCH, N.; KHALFALLAH, A.; AHMED, S. B. Software architecture in model driven architecture. In: **Computational Intelligence and Intelligent Informatics, 2007. ISCIII '07. International Symposium on**. [S.l.: s.n.], 2007. p. 219–223.

FONDEMENT, F. **Concrete syntax definition for modeling languages**. Tese (Doutorado) — École Polytechnique Fédérale de Lausanne, France, 2007.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: IEEE COMPUTER SOCIETY. **2007 Future of Software Engineering**. [S.l.], 2007. p. 37–54.

FREITAS, F.; MAIA, P. H. M. Just business: A framework for developing android applications using naked objects. In: **Components, Architectures and Reuse Software (SBCARS), 2015 IX Brazilian Symposium on**. [S.l.: s.n.], 2015. p. 11–20.

FREITAS, F.; MAIA, P. H. M. Justmodeling: an mde approach to develop android business applications. In: **2016 VI Brazilian Symposium on Computing Systems Engineering**. [S.l.: s.n.], 2016.

FREITAS, F.; MAIA, P. H. M. A naked objects based framework for developing android business applications. In: **Proceedings of the 18th International Conference on Enterprise Information Systems**. [S.l.: s.n.], 2016. p. 348–358. ISBN 978-989-758-187-8.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

GOMAA, H. **Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures**. 1st. ed. New York, NY, USA: Cambridge University Press, 2011. ISBN 0521764149, 9780521764148.

KERÄNEN, H.; ABRAHAMSSON, P. A case study on naked objects in agile software development. In: ____. **Extreme Programming and Agile Processes in Software Engineering: 6th International Conference, XP 2005, Sheffield, UK, June 18-23, 2005. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 189–197. ISBN 978-3-540-31487-5. Disponível em: <http://dx.doi.org/10.1007/11499053_22>.

KERANEN, H.; ABRAHAMSSON, P. Naked objects versus traditional mobile platform development: a comparative case study. In: **Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on**. [S.l.: s.n.], 2005. p. 274–281.

KLEPPE, A. G.; WARMER, J.; BAST, W. **MDA Explained: The Model Driven Architecture: Practice and Promise**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 032119442X.

KRIOUILE, A.; ADDAMSSIRI, N.; GADI, T.; BALOUKI, Y. Getting the static model of pim from the cim. In: **2014 Third IEEE International Colloquium in Information Science and Technology (CIST)**. [S.l.: s.n.], 2014. p. 168–173. ISSN 2327-185X.

LACHGAR, M.; ABDALI, A. Generating android graphical user interfaces using an mda approach. In: **2014 Third IEEE International Colloquium in Information Science and Technology (CIST)**. [S.l.: s.n.], 2014. p. 80–85. ISSN 2327-185X.

LACHGAR, M.; ABDALI, A. Dsl and code generator for accelerating ios apps development. In: **2015 Third World Conference on Complex Systems (WCCS)**. [S.l.: s.n.], 2015. p. 1–8.

MA, Z.; HE, X. A model-driven approach for model transformations. In: **2016 SAI Computing Conference (SAI)**. [S.l.: s.n.], 2016. p. 1199–1205.

MAIA, P. H. M.; FREITAS, F.; BORGES, M.; MUNIZ, L. L.; SILVA, A. S. da; XIMENES, J. Práticas e experiências no ensino de engenharia dirigida por modelos. **iSys-Revista Brasileira de Sistemas de Informação**, v. 9, n. 2, 2016.

MILOSAVLJEVIć, B.; VIDAKOVIć, M.; KOMAZEC, S.; MILOSAVLJEVIć, G. User interface code generation for ejb-based data models using intermediate form representations. In: **Proceedings of the 2Nd International Conference on Principles and Practice of Programming in Java**. New York, NY, USA: Computer Science Press, Inc., 2003. (PPPJ '03), p. 125–132. ISBN 0-9544145-1-9. Disponível em: <http://dl.acm.org/citation.cfm?id=957289.957327>.

MUSSET, J.; JULIOT, É.; LACRAMPE, S.; PIERS, W.; BRUN, C.; GOUBET, L.; LUSSAUD, Y.; ALLILAIRE, F. **Acceleo user guide**. 2006.

NIKIFOROVA, O.; CERNICKINS, A.; PAVLOVA, N. Discussing the difference between model driven architecture and model driven development in the context of supporting tools. In: **Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on**. [S.l.: s.n.], 2009. p. 446–451.

OMG. **MDA - The Architecture Of Choice For A Changing World**. 2016. Disponível em http://www.omg.org/mda/. Acessado em 01/05/2016.

PARADA, A. G.; BRISOLARA, L. B. d. A model driven approach for android applications development. In: **Proceedings of the 2012 Brazilian Symposium on Computing System Engineering**. Washington, DC, USA: IEEE Computer Society, 2012. (SBESC '12), p. 192–197. ISBN 978-0-7695-4929-3. Disponível em: <http://dx.doi.org/10.1109/SBESC.2012.44>.

PARADA, A. G.; SIEGERT, E.; BRISOLARA, L. B. de. Generating java code from uml class and sequence diagrams. In: **2011 Brazilian Symposium on Computing System Engineering**. [S.l.: s.n.], 2011.

PAWSON, R. **Naked Objects**. Tese (Doutorado) — University of Dublin, Trinity College, 2004.

PAWSON, R.; MATTHEWS, R. Naked objects: A technique for designing more expressive systems. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 36, n. 12, p. 61–67, dez. 2001. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/583960.583967>.

PAWSON, R.; MATTHEWS, R. Naked objects. In: **Companion of the 17th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications**. New York, NY, USA: ACM, 2002. (OOPSLA '02), p. 36–37. ISBN 1-58113-626-9. Disponível em: <http://doi.acm.org/10.1145/985072.985091>.

PAWSON, R.; WADE, V. Agile development using naked objects. In: **Proceedings of the 4th International Conference on Extreme Programming and Agile Processes in Software Engineering**. Berlin, Heidelberg: Springer-Verlag, 2003. (XP'03), p. 97–103. ISBN 3-540-40215-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1763875.1763892>.

QUINTERO, J. B.; ANAYA, R. MDA Y EL PAPEL DE LOS MODELOS EN EL PROCESO DE DESARROLLO DE SOFTWARE. **Revista EIA**, scieloco, p. 131 – 146, 12 2007. ISSN 1794-1237. Disponível em: <http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1794-12372007000200011&nrm=iso>.

RAJA, A.; LAKSHMANAN, D. Article: Naked objects framework. **International Journal of Computer Applications**, v. 1, n. 20, p. 37–41, February 2010. Published By Foundation of Computer Science.

RIBEIRO, A.; SILVA, A. R. da. Xis-mobile: A dsl for mobile applications. In: **Proceedings of the 29th Annual ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2014. (SAC '14), p. 1316–1323. ISBN 978-1-4503-2469-4. Disponível em: <http://doi.acm.org/10.1145/2554850.2554926>.

RIBEIRO, A.; SILVA, A. Rodrigues da. Evaluation of xis-mobile, a domain specific language for mobile application development. **Journal of Software Engineering and Applications**, v. 7, p. 906–919, 10 2014.

SABRAOUI, A.; KOUTBI, M. E.; KHRISS, I. Gui code generation for android applications using a mda approach. In: **Complex Systems (ICCS), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 1–6.

SCHMIDT, D. C. Model-driven engineering. **Computer**, IEEE Computer Society, v. 39, n. 2, p. 0025–31, 2006.

SILVA, L. M. P. T. d.; ABREU, F. B. e. A mde generative approach for mobile business apps. In: **Quality of Information and Communications Technology (QUATIC), 2014 9th International Conference on the**. [S.l.: s.n.], 2014. p. 312–317.

SILVA, L. P. da; ABREU, F. B. e. Model-driven gui generation and navigation for android bis apps. In: **Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on**. [S.l.: s.n.], 2014. p. 400–407.

SILVA, M.; BARBOSA, E.; MALDONADO, J. Model-driven development of learning objects. In: **Frontiers in Education Conference (FIE), 2011**. [S.l.: s.n.], 2011. p. F4E–1–F4E–6. ISSN 0190-5848.

SILVA, W. C. da; OLIVEIRA, J. L. Gerência de interface homem-computador para sistemas de informação empresariais: uma abordagem baseada em modelos. **iSys - Revista Brasileira de Sistemas de Informação**, v. 2, p. 5–16, 2009.

SINGH, Y.; SOOD, M. Model driven architecture: A perspective. In: **Advance Computing Conference, 2009. IACC 2009. IEEE International**. [S.l.: s.n.], 2009. p. 1644–1652.

TORRES, A. **MD-JPA : um perfil UML para modelagem do mapeamento objeto-relacional com JPA em uma abordagem dirigida por modelos**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação, 2009.

VAUPEL, S.; TAENTZER, G.; HARRIES, J. P.; STROH, R.; GERLACH, R.; GUCKERT, M. Model-driven engineering languages and systems: 17th international conference, models 2014, valencia, spain, september 28 – october 3, 2014. proceedings. In: ____. Cham: Springer International Publishing, 2014. cap. Model-Driven Development of Mobile Applications Allowing Role-Driven Variants, p. 1–17. ISBN 978-3-319-11653-2. Disponível em: <http://dx.doi.org/10.1007/978-3-319-11653-2_1>.

WEBER, A. R. H.; JANUARIO, C. A.; MATOS, S. N. Geração do modelo psm em uma ferramenta de código aberto para um sistema help desk. **Revista Gestão Industrial**, v. 6, n. 4, 2010.

APÊNDICES

APÊNDICE A – Geração de Código com Acceleo

Figura 25 – Módulo de Geração de Código para o tipo *Class*

```
[comment encoding = UTF-8 /]
[module generateClass('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::generateClassAttributes/]
[import org::eclipse::acceleo::module::JBModel::common::generateClassReferences/]
[import org::eclipse::acceleo::module::JBModel::common::generateClassConstructor/]
[import org::eclipse::acceleo::module::JBModel::common::generateClassGettersSetters /]
[import org::eclipse::acceleo::module::JBModel::common::generateClassMethods /]

[import org::eclipse::acceleo::module::JBModel::common::generateImplementClassMethods /]
[import org::eclipse::acceleo::module::JBModel::common::generateImplementInterfaceMethods /]

[import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessImports /]
[import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessExtending /]
[import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessMethods /]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationEntity/]
[import org::eclipse::acceleo::module::JBModel::common::jb::persistence::generateAnnotationTable /]
[import org::eclipse::acceleo::module::JBModel::common::jb::generateJustBusinessGeneralImports /]

[template public generateClass(aClass : Class)]
[file (aClass.getFileName(), false, 'UTF-8')]
[aClass._package.getPackage()/]

[aClass.generateJustBusinessImports()/]

[aClass.generateJustBusinessGeneralImports()/]
[aClass.generateAnnotationTable()/]
[aClass.generateAnnotationEntity()/]
[aClass.getClassModifiers()/]class [aClass.getClassifierName()/] [aClass.generateJustBusinessExtending()/] {
    [aClass.generateAttributes()/]
    [aClass.generateReferences()/]
    [aClass.generateEmptyConstructor()/]
    [aClass.generateMethods()/]
    [aClass.generateImplementClassMethods()/]
    [aClass.generateImplementInterfaceMethods()/]
    [aClass.generateJustBusinessMethods()/]
    [aClass.generateGettersSetters()/]
}
[/file]
[/template]

```

Fonte: Elaborado pelo autor

Figura 26 – Módulo de Geração de *Imports* do JustBusiness

```
[comment encoding = UTF-8 /]
[module generateJustBusinessImports('http://www.example.org/JBModel')]


[template public generateJustBusinessImports(aClassifier : Classifier)]
import org.jb.annotation.model.*;
import org.jb.annotation.model.enums.*;
import org.jb.annotation.persistence.*;
import org.jb.annotation.persistence.enums.*;
import org.jb.model.JBEntity;
[/template]

```

Fonte: Elaborado pelo autor

Figura 27 – Módulo de Geração de *Imports* Gerais

```
[comment encoding = UTF-8 /]
[module generateJustBusinessGeneralImports('http://www.example.org/JBModel')]


[template public generateJustBusinessGeneralImports(aClass : Class)]
[if (aClass.attributes->select(a : Attribute | a.type.equalsIgnoreCase('Date'))->notEmpty())]
import java.util.Date;
[/if]
[if (aClass.references->select(reff : Reference | reff.oclIsTypeOf(Aggregation)
                                                  or reff.oclIsTypeOf(Composition))->notEmpty())]
import java.util.List;
import java.util.ArrayList;
[/if]
[/template]

```

Fonte: Elaborado pelo autor

Figura 28 – Módulo de Geração da anotação *@Table*

```
[comment encoding = UTF-8 /]
[module generateAnnotationTable('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries /]

[template public generateAnnotationTable(aClassifier : Classifier)]
@Table(name="[aClassifier.getTableName()/]")
[/template]

```

Fonte: Elaborado pelo autor

Figura 29 – Módulo de Geração da anotação *@Entity*

```
[comment encoding = UTF-8 /]
[module generateAnnotationEntity('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]

[template public generateAnnotationEntity(anClass : Class)]
@Entity(label="[anClass.getAppropriateDescription()/]",
            collectionLabel="[anClass.getAppropriateCollectionName()/]")
[/template]

```

Fonte: Elaborado pelo autor

Figura 30 – Módulo de Geração de Código de Extensão de Superclasse e Realização de Interfaces

```
[comment encoding = UTF-8 /]
[module generateJustBusinessExtending('http://www.example.org/JBModel')]

[template public generateJustBusinessExtending(aClass : Class)]
[aClass.generateExtending()/][aClass.generateImplementing()/]
[/template]

[template public generateExtending(aClass : Class)]
[if (aClass.Generalization.oclIsUndefined())]extends JBEntity
[else]extends [aClass.Generalization.name/]
[/if]
[/template]

[template public generateImplementing(aClass : Class)]
[if ((not aClass.Implement->oclIsUndefined()) and aClass.Implement->notEmpty())]
    [for (inter : Interface | aClass.Implement)]
        [if (inter = aClass.Implement->first())]
            implements [inter.name/]
        [else], [inter.name/]
        [/if]
    [/for]
[/if]
[/template]

```

Fonte: Elaborado pelo autor

Figura 31 – Módulo de Geração de Código para Declaração de Atributos

```
[comment encoding = UTF-8 /]
[module generateClassAttributes('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationAttribute/]
[import org::eclipse::acceleo::module::JBModel::common::jb::persistence::columns::generateJustBusinessAttributeAnnotations /]

[template public generateAttributes(aClass : Structure)]
[for (attr : Attribute | aClass.attributes)]
[if (attr = aClass.attributes->last())]
[attr.generateItem()/][lineSeparator()/][else][attr.generateItem()/][lineSeparator()/]
[/if]
[/for]
[/template]

[template public generateItem(anAttribute : Attribute)]
[if (anAttribute.justBusiness)][anAttribute.generateAnnotationAttribute()/][anAttribute.generateAttributeAnnotations()/][/if]
[anAttribute.getStructuralFeatureModifiers()/][anAttribute.type/] [anAttribute.name/];
[/template]

```

Fonte: Elaborado pelo autor

Figura 32 – Módulo de Geração de Código para Declaração de Referências

```
[comment encoding = UTF-8 /]
[module generateClassReferences('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationAttribute/]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationReference /]
[import org::eclipse::acceleo::module::JBModel::common::jb::persistence::columns::generateJustBusinessAttributeAnnotations /]
[import org::eclipse::acceleo::module::JBModel::common::jb::persistence::columns::generateJustBusinessReferenceAnnotations /]

[template public generateReferences(aClass : Structure)]
[for (reff : Reference | aClass.references)]
[if (reff = aClass.references->last())]
[reff.generateItem()/][lineSeparator()/][else][reff.generateItem()/][lineSeparator()/]
[/if]
[/for]
[/template]

[template public generateItem(aReference : Reference)]
[if (aReference.justBusiness)][aReference.generateAnnotationReference()/][aReference.generateReferenceAnnotations()/][/if]
[aReference.getStructuralFeatureModifiers()/][aReference.getReferenceType()/] [aReference.name/];
[/template]

```

Fonte: Elaborado pelo autor

Figura 33 – Módulo de Geração de Código para Construtor de Classe

```
[comment encoding = UTF-8 /]
[module generateClassConstructor('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::generateClassAttributesReferencesInitialization /]

[template public generateEmptyConstructor(aClass : Class)]
public [aClass.getClassifierName()/]() {
    super();
    [aClass.generateClassAttributesReferencesInitialization()/]
}
[/template]

```

Fonte: Elaborado pelo autor

Figura 34 – Módulo de Geração de Código para Métodos de Classe

```
[comment encoding = UTF-8 /]
[module generateClassMethods('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationAction /]
[import org::eclipse::acceleo::module::JBModel::common::generateMethodParameters /]

[template public generateMethods(aClass : Structure)]
[if (aClass.operations->notEmpty())]

/**Class Methods*/

    [for (opt : Operation | aClass.operations)]
        [if (opt <> aClass.operations->last())][opt.generateItem()/][lineSeparator()/][lineSeparator()/]
        [else][opt.generateItem()/]
        [/if]
    [/for]
[/if]
[/template]

[template public generateItem(anOperation : Operation)]
[if (anOperation.justBusiness)][anOperation.generateAnnotationAction()/][/if]
[anOperation.getOperationModifiers()/][anOperation.type/] [anOperation.getAppropriateOperationName()/]
([anOperation.generateParameters()/])
    [if (anOperation.abstract)];
    [else] {
        //TODO
        [if (not anOperation.type.equalsIgnoreCase('void'))]
            return null;
    [/if]
}
[/if]
[/template]

```

Fonte: Elaborado pelo autor

Figura 35 – Módulo de Geração de Código para Métodos Extendidos de Superclasse

```
[comment encoding = UTF-8 /]
[module generateImplementClassMethods('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationAction /]
[import org::eclipse::acceleo::module::JBModel::common::generateMethodParameters /]

[template public generateImplementClassMethods(aClass : Class)]
[if (not aClass.abstract)]
    [if (not aClass.Generalization.oclIsUndefined())]
[aClass.Generalization.generateImplementClassMethod()/]
    [/if]
[/if]
[/template]

[template public generateImplementClassMethod(aClass : Class)]
[if (not aClass.Generalization.oclIsUndefined())][aClass.Generalization.generateImplementClassMethod()/][/if]
[if (not aClass.operations->select(abstract)->isEmpty())]
/**Superclass [aClass.name/] Methods Implementing*/

    [for (op : Operation | aClass.operations->select(op | op.abstract))]
[op.generateItem()/][lineSeparator()/]
    [/for]
[/if]
[/template]

[template public generateItem(anOperation : Operation)]
[if (anOperation.justBusiness)][anOperation.generateAnnotationAction()/][/if]
[anOperation.getOperationModifiers()/][anOperation.type/] [anOperation.getAppropriateOperationName()/]
([anOperation.generateParameters()/]) {
    //TODO
[if (not anOperation.type.equalsIgnoreCase('void'))]
    return null;
}[/if]
[/template]

```

Fonte: Elaborado pelo autor

Figura 36 – Módulo de Geração de Código para Métodos Implementados de Interfaces

```
[comment encoding = UTF-8 /]
[module generateImplementInterfaceMethods('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]
[import org::eclipse::acceleo::module::JBModel::common::jb::gui::generateAnnotationAction /]
[import org::eclipse::acceleo::module::JBModel::common::generateMethodParameters /]

[template public generateImplementInterfaceMethods(aClass : Class)]
[if (not aClass.abstract)]
    [if (not aClass.Implement->isEmpty())]
        [for (it : Interface | aClass.Implement)]
[it.generateImplementInterfaceMethod()/]
        [/for]
    [/if]
[/if]
[/template]

[template public generateImplementInterfaceMethod(aInterface : Interface)]
[if (not aInterface.Generalization.oclIsUndefined())][aInterface.Generalization.generateImplementInterfaceMethod()/]
[lineSeparator()/][/if][if (not aInterface.operations->isEmpty())]
/**Interface [aInterface.name/] Methods Implementing*/

[for (op : Operation | aInterface.operations)]
[if (op <> aInterface.operations->last())][op.generateItem()/][lineSeparator()/]
[else][op.generateItem()/]
[/if][/for][/if]
[/template]

[template public generateItem(anOperation : Operation)]
[if (anOperation.justBusiness)][anOperation.generateAnnotationAction()/][/if]
[anOperation.getOperationModifiers()/][anOperation.type/] [anOperation.getAppropriateOperationName()/]
([anOperation.generateParameters()/]) {
    //TODO
[if (not anOperation.type.equalsIgnoreCase('void'))]
    return null;
}[/if]
[/template]

```

Fonte: Elaborado pelo autor

Figura 37 – Módulo de Geração de Código para Métodos Padrões do JustBusiness

```
1  [comment encoding = UTF-8 /]
2  [module generateJustBusinessMethods('http://www.example.org/JBModel')]
3
4
5  [template public generateJustBusinessMethods(aClass : Class)]
6  [if (aClass.justBusiness)]
7  /**Methods to display object information*/
8
9  [aClass.generatePrimaryDescriptionMethod()/]
10
11 [aClass.generateSecondaryDescriptionMethod()/]
12 [/if]
13 [/template]
14
15 [template public generatePrimaryDescriptionMethod(aClass : Class)]
16 public String toPrimaryDescription() {
17     return this.toString();
18 }
19 [/template]
20
21 [template public generateSecondaryDescriptionMethod(aClass : Class)]
22 public String toSecondaryDescription() {
23     return this.toString();
24 }
25 [/template]
26
```

Fonte: Elaborado pelo autor

Figura 38 – Módulo de Geração de Código para Métodos *Get* e *Set*

```
[comment encoding = UTF-8 /]
[module generateClassGettersSetters('http://www.example.org/JBModel')]

[import org::eclipse::acceleo::module::JBModel::common::queries/]

[template public generateGettersSetters(aClass : Class)]
/**GET and SET methods*/
[for (attr : Attribute | aClass.attributes.oclAsSet())]

[attr.generateItem()/]
[/for]
[for (reff : Reference | aClass.references.oclAsSet())]

[reff.generateItem()/]
[/for]
[/template]



[template public generateGetter(anAttribute : Attribute)]
public [anAttribute.type/] [anAttribute.getGetName()/]() {
    return this.[anAttribute.name/];
}
[/template]

[template public generateSetter(anAttribute : Attribute)]
public void [anAttribute.getSetName()/]([anAttribute.type/] [anAttribute.name/]) {
    this.[anAttribute.name/] = [anAttribute.name/];
}
[/template]

[template public generateItem(anAttribute : Attribute)]
[anAttribute.generateGetter()/]

[anAttribute.generateSetter()/]
[/template]

```

Fonte: Elaborado pelo autor

Figura 39 – Módulo de Consultas

```
[comment encoding = UTF-8 /]
[module queries('http://www.example.org/JBModel')]

[query public getClassifierFileName (aClassifier : Classifier) :
    String = aClassifier.name.replaceAll(' ', '').concat('.java')/]

[query public getClassifierName (aClassifier : Classifier) :
    String = aClassifier.name.replaceAll(' ', '').toUpperFirst()/]

[query public getClassifierFullName (aClassifier : Classifier) :
    String = aClassifier._package.getPackageName().concat('.').concat(aClassifier.getClassifierName())/]

[query public getClassModifiers (aClass : Class) : String =
    aClass.access.getAccessModifier().concat(aClass.abstract.getAbstractModifier().concat(aClass.final.getFinalModifier()))
/]

```

Fonte: Elaborado pelo autor

APÊNDICE B – Geração de Código Android com JustBusiness

Figura 40 – Código da classe processadora de anotações *JBProcessor*

```
1  package org.jb.annotation.processor;
2
3  import java.util.List;
23
24 @SupportedAnnotationTypes(value= {"org.jb.annotation.model.Entity",
25         "org.jb.annotation.model.Enumeration",
26         "org.jb.annotation.persistence.Table"})
27
28 public class JBProcessor extends AbstractProcessor {
29     private ProcessingEnvironment environment;
30     private Filer filer;
31     private Messager messager;
32     ClassLoader loader;
33
35     public void init(ProcessingEnvironment env) {
42
43     @Override
44     public boolean process(Set<? extends TypeElement> elements, RoundEnvironment env) {
45         DataDictionary dictionary = new DataDictionary();
46         ModelVisitor mv = new ModelVisitor(dictionary, this.messager);
47         PersistenceVisitor pv = new PersistenceVisitor(dictionary, this.messager);
48         Element element = null;
49         ProjectXMLReader projectConfig = new ProjectXMLReader(filer, messager, element);
50
51         if(!projectConfig.isCodeGenerationEnabled()) {
52             return true;
53         }
54
55         Set<? extends Element> annotatedClass = env.getElementsAnnotatedWith(Entity.class);
56         for (Element e : annotatedClass) {
57             element = e;
58             mv.doSomething(e);
59             List<? extends Element> annotatedInType = e.getEnclosedElements();
60             for (Element eAttribute : annotatedInType) {
61                 mv.doSomething(eAttribute);
62             }
63         }
64         Set<? extends Element> annotatedEnum = env.getElementsAnnotatedWith(Enumeration.class);
65         for (Element e : annotatedEnum) {
66             element = e;
67             mv.doSomething(e);
68         }
69         Set<? extends Element> annotatedTable = env.getElementsAnnotatedWith(Table.class);
70         for (Element e : annotatedTable) {
71             element = e;
72             pv.doSomething(e);
73             List<? extends Element> annotatedInTable = e.getEnclosedElements();
74             for (Element eField : annotatedInTable) {
75                 pv.doSomething(eField);
76             }
77         }
78
79         ProjectAnalise pa = new ProjectAnalise(dictionary, this.messager, this.filer, element);
80         pa.analise();
81
82         mv.buildResources(this.messager, this.filer, element);
83         return true;
84     }
85 }
```

Fonte: Elaborado pelo autor

Figura 41 – Código da classe *ProjectAnalise*, que analisa informações das classes de negócio

```
1  package org.jb.builder;
2
3  import javax.annotation.processing.Filer;
12
13 public class ProjectAnalise {
14     DataDictionary dictionary;
15     Messager messager;
16     Filer filer;
17     Element element;
18
19     public ProjectAnalise(DataDictionary dictionary, Messager messager,
27
28     public void analise() {
29         for(ClassInfo c : dictionary.getClassValues()) {
30             analiseClass(c);
31         }
32         for(ClassInfo c : dictionary.getClassValues()) {
33             print(c);
34         }
35     }
36
37     public void analiseClass(ClassInfo c) {
38         if(!c.getAllAttributes().isEmpty()
39                 || !c.getAllActions().isEmpty()
40                 || !c.getAllMethods().isEmpty())
41             return;
42
43         if(c.getSuperClass().equals(JBEntity.class.getName())) {
44             c.getAllAttributes().clear();
45             c.getAllAttributes().addAll(c.getAttributes());
46             c.getAllMethods().clear();
47             c.getAllMethods().addAll(c.getMethods());
48             c.getAllActions().clear();
49             c.getAllActions().addAll(c.getActions());
50             return;
51         }
52         else {
53             ClassInfo sc = dictionary.getClassInfo(c.getSuperClass());
54             analiseClass(sc);
55
56             c.getAllAttributes().addAll(c.getAttributes());
57             for(AttributeInfo at : sc.getAllAttributes()) {
58                 c.addInAllAttribute(at);
59             }
60             c.getAllMethods().addAll(c.getMethods());
61             for(MethodInfo m : sc.getAllMethods()) {
62                 c.addInAllMethod(m);
63             }
64             c.getAllActions().addAll(c.getActions());
65             for(MethodInfo ac : sc.getAllActions()) {
66                 c.addInAllAction(ac);
67             }
68         }
69     }
70
71     public void print(ClassInfo c) {
83 }
```

Fonte: Elaborado pelo autor

Figura 42 – Código da classe *ModelVisitor*, que visita elementos mapeados com anotações de modelo

```
 1  package org.jb.visitor;
 2
 3⊕ import java.util.*;
26
27  public class ModelVisitor implements Visitor {
28      DataDictionary dictionary = null;
29
30⊖     public ModelVisitor(DataDictionary dictionary, Messager messager) {
31          this.dictionary = dictionary;
32          //this.messager = messager;
33      }
34
35⊖     public ModelVisitor(Messager messager) {
36          this.dictionary = new DataDictionary();
37          //this.messager = messager;
38      }
39
40⊖     public DataDictionary getDictionary() {
41          return dictionary;
42      }
43
44⊖     public void doSomething(Element element) {
45          if(element.getKind() == ElementKind.CLASS)
46              doSomething((TypeElement)element);
47          if(element.getKind() == ElementKind.ENUM)
48              doSomething((TypeElement)element);
49          if(element.getKind() == ElementKind.FIELD)
50              doSomething((VariableElement)element);
51          if(element.getKind() == ElementKind.METHOD)
52              doSomething((ExecutableElement)element);
53      }
54
55⊕     public void doSomething(TypeElement element) {
132
133⊕    public void doSomething(VariableElement element) {
185
186⊕    public void doSomething(ExecutableElement element) {
300
301⊖    public void buildResources(Messager messager, Filer filer, Element element) {
302         if(dictionary != null && dictionary.getTotalClasses() > 0) {
303             Collection<ClassInfo> classList = dictionary.getClassValues();
304             ClassInfo []array = new ClassInfo[classList.size()];
305             array = classList.toArray(array);
306             try {
307                 ProjectBuilder pb = new ProjectBuilder(dictionary, messager, filer, element);
308                 pb.build();
309             } catch (Exception e) {
310                 e.printStackTrace();
311             }
312         }
313     }
314 }
```

Fonte: Elaborado pelo autor

Figura 43 – Código da classe *PersistenceVisitor*, que visita elementos mapeados com anotações de persistência

```
 1  package org.jb.visitor;
 2
 3⊕ import java.util.*;
18
19  public class PersistenceVisitor implements Visitor {
20      DataDictionary dictionary = null;
21
22⊖     public PersistenceVisitor(DataDictionary dictionary, Messager messager) {
23          this.dictionary = dictionary;
24      }
25
26⊖     public PersistenceVisitor(Messager messager) {
27          this.dictionary = new DataDictionary();
28      }
29
30⊖     public DataDictionary getDictionary() {
31          return dictionary;
32      }
33
34⊖     public void doSomething(Element element) {
35          if(element.getKind() == ElementKind.CLASS)
36              doSomething((TypeElement)element);
37          if(element.getKind() == ElementKind.ENUM)
38              doSomething((TypeElement)element);
39          if(element.getKind() == ElementKind.FIELD)
40              doSomething((VariableElement)element);
41          if(element.getKind() == ElementKind.METHOD)
42              doSomething((ExecutableElement)element);
43      }
44
45⊕     public void doSomething(TypeElement element) {
116
117⊕    public void doSomething(VariableElement element) {
384
385⊕    public void doSomething(ExecutableElement element) {
388
389⊖    public void buildResources(Messager messager, Filer filer, Element element) {
390         if(dictionary != null && dictionary.getTotalTables() > 0) {
391
392             Collection<TableInfo> tableList = dictionary.getTableValues();
393
394             TableInfo []array = new TableInfo[tableList.size()];
395             array = tableList.toArray(array);
396         }
397     }
398 }
```

Fonte: Elaborado pelo autor

Figura 44 – Código da classe *ProjectBuilder*, que realiza a criações de todos os arquivos do projeto Android

```
 1  package org.jb.builder;
 2
 3  import java.util.ArrayList;
66
67  public class ProjectBuilder {
68      DataDictionary dictionary;
69      Messager messager;
70      Filer filer;
71      Element element;
72
73      List<ProjectGenerator> listProjectGenerator;
74      List<Generator> listGenerator;
75      List<MethodGenerator> listMethodGenerator;
76      List<EntityGenerator> listEntityGenerator;
77      List<EnumGenerator> listEnumGenerator;
78
79      public ProjectBuilder(DataDictionary dictionary, Messager messager, Filer filer, Element element) {
86
87      public void prepareBuild() {
176
177     public void build() {
178         for(int i = 0; i < listGenerator.size(); i++) {
179             Generator g = listGenerator.get(i);
180             g.source();
181             g.genarate();
182         }
183         listGenerator.clear();
184
185         for(int i = 0; i < listProjectGenerator.size(); i++) {
186             ProjectGenerator pg = listProjectGenerator.get(i);
187             pg.source();
188             pg.genarate();
189         }
190         listProjectGenerator.clear();
191
192         for(int i = 0; i < listEntityGenerator.size(); i++) {
193             EntityGenerator eg = listEntityGenerator.get(i);
194             eg.source();
195             eg.genarate();
196         }
197         listEntityGenerator.clear();
198
199         for(int i = 0; i < listMethodGenerator.size(); i++) {
200             MethodGenerator mg = listMethodGenerator.get(i);
201             mg.source();
202             mg.genarate();
203         }
204         listMethodGenerator.clear();
205
206         for(int i = 0; i < listEnumGenerator.size(); i++) {
207             EnumGenerator eg = listEnumGenerator.get(i);
208             eg.source();
209             eg.genarate();
210         }
211         listEntityGenerator.clear();
212     }
213 }
```

Fonte: Elaborado pelo autor

Figura 45 – Código da classe *ListActivityLayoutEntityGenerator*, responsável pela criação arquivo de *layout* para listagem de objetos

```
package org.jb.generator;

import java.io.BufferedWriter;

public class ListActivityLayoutEntityGenerator extends EntityGenerator {

    public ListActivityLayoutEntityGenerator(ClassInfo c, DataDictionary dictionary) {

    public ListActivityLayoutEntityGenerator(ClassInfo classInfo,

    @Override
    public void source() {
        ActivityResourceName activityResourceName = new ActivityResourceName();
        FragmentResourceName fragmentResourceName = new FragmentResourceName();

        str.append("<?xml version=\"1.0\" encoding=\"utf-8\"?>\n");
        //TODO Ajustes de Layout
        str.append("<RelativeLayout xmlns:android=\"http://schemas.android.com/apk/res/android\"\n");

        str.append("    xmlns:tools=\"http://schemas.android.com/tools\"\n");

        //TODO Ajustes de Largura
        str.append("    android:layout_width=\"match_parent\"\n");
        str.append("    android:layout_height=\"match_parent\"\n");

        //TODO Ajustes de Margens
        str.append("    android:paddingBottom=\"@dimen/activity_vertical_margin\"\n");
        str.append("    android:paddingLeft=\"@dimen/activity_horizontal_margin\"\n");
        str.append("    android:paddingRight=\"@dimen/activity_horizontal_margin\"\n");
        str.append("    android:paddingTop=\"@dimen/activity_vertical_margin\"\n");

        //TODO ajustes de Contexto
        str.append("    tools:context=\"app.jb.generated." +
                activityResourceName.getClassName(c, Operation.LIST, dictionary) + "\" >\n");
        str.append("\n");

        str.append("    <TextView\n");
        str.append("        android:id=\"@+id/" +
                activityResourceName.getClassTitleWidgetResourceName(c, Operation.LIST, dictionary) + "\"\n");
        str.append("        android:layout_width=\"match_parent\"\n");
        str.append("        android:layout_height=\"wrap_content\"\n");
        str.append("        android:layout_marginTop=\"10dp\"\n");
        str.append("        android:layout_marginBottom=\"20dp\"\n");
        str.append("        android:layout_alignParentTop=\"true\"\n");
        str.append("        android:text=\"@string/" +
                activityResourceName.getClassTitleStringResourceName(c, Operation.LIST, dictionary) + "\"\n");
        str.append("        android:textStyle=\"bold\"\n");
        str.append("        android:textAppearance=\"?android:attr/textAppearanceMedium\" />\n");

        str.append("    <View\n");
        str.append("        android:id=\"@+id/" +
                activityResourceName.getFragmentWidgetLineResourceName(c, Operation.LIST, dictionary) + "\"\n");
        str.append("        android:layout_width=\"match_parent\"\n");
        str.append("        android:layout_height=\"1dp\"\n");
        str.append("        android:layout_marginTop=\"2dp\"\n");
        str.append("        android:layout_marginBottom=\"2dp\"\n");
        str.append("        android:background=\"@android:color/darker_gray\"\n");
        str.append("        android:layout_below=\"@id/" +
                activityResourceName.getClassTitleWidgetResourceName(c, Operation.LIST, dictionary) + "\" />\n");

        //TODO Transformar os Atributos em Fields
        str.append("    <fragment\n");
        str.append("        android:id=\"@+id/" +
                activityResourceName.getFragmentWidgetListResourceName(c, Operation.LIST, dictionary) + "\"\n");
        str.append("        android:name=\"app.jb.generated." +
                fragmentResourceName.getClassName(c, Operation.LIST, dictionary) + "\"\n");
        str.append("        android:layout_width=\"match_parent\"\n");
        str.append("        android:layout_height=\"match_parent\"\n");
        str.append("        android:orientation=\"vertical\"\n");
        str.append("        android:layout_below=\"@id/" +
                activityResourceName.getFragmentWidgetLineResourceName(c, Operation.LIST, dictionary) + "\">\n");
        str.append("    </fragment>\n");

        str.append("</RelativeLayout>\n");
    }

    @Override
    public void generate() {
        ActivityResourceName activityResourceName = new ActivityResourceName();
        FileObject file = null;
        try {
            file = filer.createResource(StandardLocation.SOURCE_OUTPUT , "", "../../res/layout/" +
                    activityResourceName.getClassLayoutResourceName(c, Operation.LIST, dictionary) + ".xml", element);
            BufferedWriter bw = new BufferedWriter(file.openWriter());
            bw.append(str.toString());
            bw.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

Fonte: Elaborado pelo autor