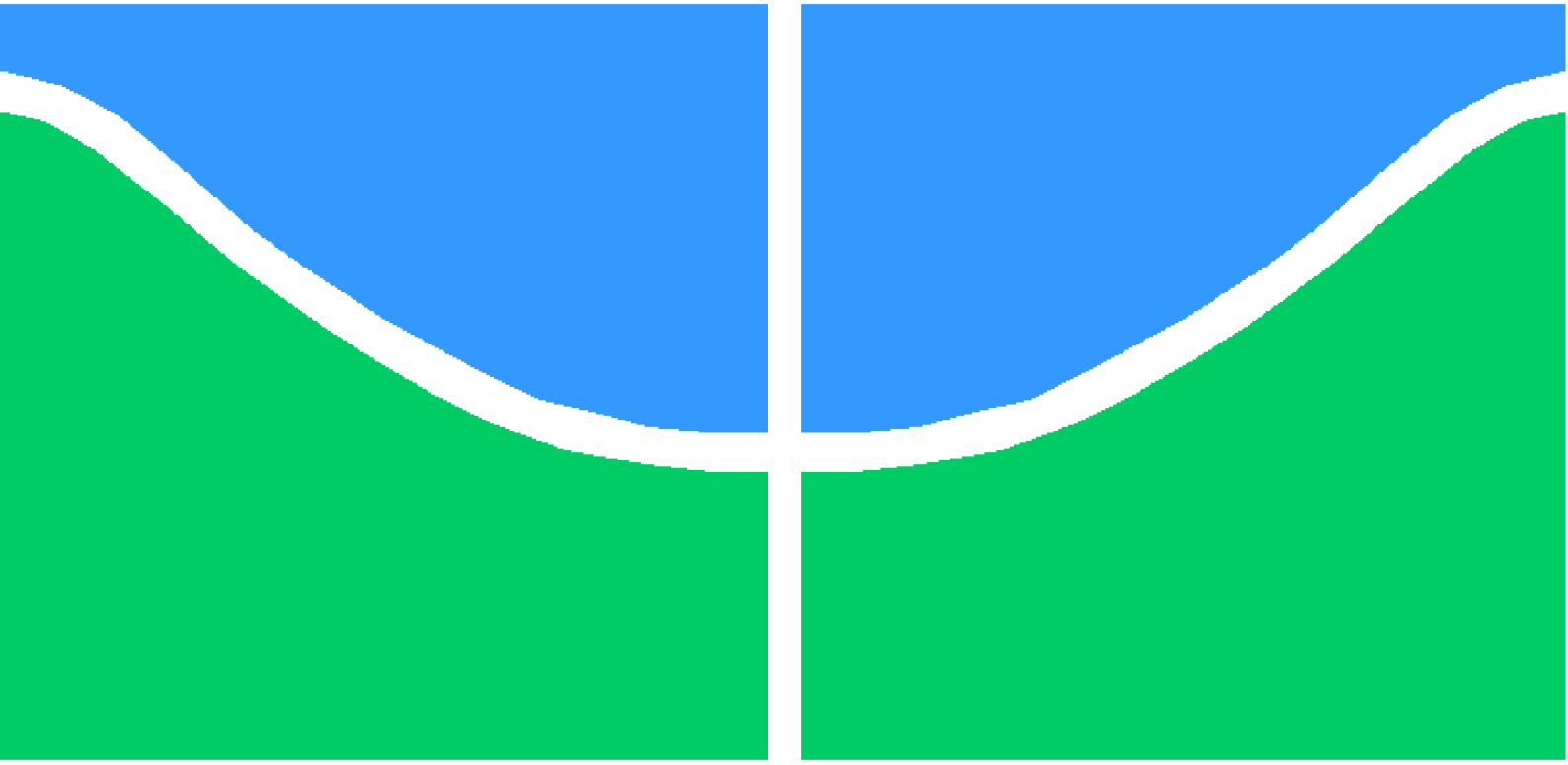


Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Geração de Código Dirigida a Modelo: Uma Abordagem Orientada à Meta

Autor: Rafael Akiyoshi Holsbach
Orientador: Profa. Dra. Milene Serrano

Brasília, DF
2018

Rafael Akiyoshi Holsbach

Geração de Código Dirigida a Modelo: Uma Abordagem Orientada à Meta

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Profa. Dra. Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2018

Rafael Akiyoshi Holsbach

Geração de Código Dirigida a Modelo: Uma Abordagem Orientada à Meta

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 03 de Julho de 2018 – Data da aprovação do trabalho:

Profa. Dra. Milene Serrano
Orientador

Prof. Dr. Maurício Serrano
Coorientador

Prof. Msc. Giovanni Almeida Santos
Convidado 1

Brasília, DF
2018

Este trabalho é dedicado às pessoas que
nunca cogitaram em desistir de seus sonhos.

Agradecimentos

Agradeço à minha mãe Regina Akiyoshi, heroína, que sempre me deu apoio e incentivo incondicional, nos momentos mais difíceis, de desânimo e cansaço. Agradeço aos meus amigos, Danilo Barros, Gustavo Sabino, Vitor Bertulucci, entre outros que me acompanharam e me ajudaram nessa jornada acadêmica. Agradeço à minha namorada, Milena Martins, por sempre estar ao meu lado, apoiando-me e fazendo da graduação um caminho menos árduo.

Meus agradecimentos também aos meus orientadores, Profa. Dra. Milene Serrano, e Prof. Dr. Maurício Serrano, pela paciência, pelo empenho dedicado à elaboração deste trabalho, e pelas excelentes orientações.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Resumo

A linha de base arquitetural de um software é considerada uma especificação de suma relevância, a qual orienta o desenvolvimento de um software em aderência aos requisitos acordados na Engenharia de Requisitos. Entretanto, a literatura da área da computação apresenta evidências de que os desenvolvedores frequentemente optam por iniciar a codificação sem ao menos iniciar uma especificação arquitetural prévia. Tal prática comumente compromete a manutenção evolutiva do software desenvolvido, tornando essa, principalmente, dispendiosa.

Considerando que tanto a especificação de uma linha de base de requisitos quanto a especificação de uma linha de base arquitetural são representadas por modelos, em diferentes níveis de abstração, os quais permitem, respectivamente, (i) modelar os requisitos elicitados junto aos interessados, e (ii) modelar os componentes e conectores arquiteturais, há forte associação da atividade de modelagem em ambos os casos.

Diante do exposto, e procurando colaborar nesse cenário, esse trabalho preocupou-se com o desenvolvimento de uma aplicação, chamada Model IT, atuando na geração semiautomática de código usando modelos como base.

Entretanto, para viabilizar o desenvolvimento de uma primeira versão dessa aplicação, foram estabelecidos: (i) um perfil de aplicação específico para geração de código, no caso, aplicações web; (ii) uma notação de modelagem específica, no caso, a notação i*; (iii) um padrão arquitetural específico, no caso, o Padrão *Model-View-Controller* (MVC), e (iv) uma linguagem de programação específica, no caso, orientada ao *framework* Django. A ideia é apoiar os desenvolvedores na construção de aplicações web, considerando a geração semiautomática de código em Django, orientando-se por modelos em i* (em diferentes níveis de abstração) e o Padrão Arquitetural MVC, e fazendo uso de heurísticas transformacionais bem como de princípios de *Model Driven Architecture* (MDA).

Adicionalmente, são apresentados detalhes da Model IT bem como os resultados obtidos com o uso desse suporte junto ao público alvo, considerando ciclos de pesquisa-ação, o que permitiu avaliar o uso do suporte e realizar evoluções no mesmo com base nos dados coletados em cada ciclo.

Palavras-chaves: Arquitetura, Arquitetura Dirigida a Modelo, i*, Model View Controller, Geração de código.

Abstract

The architectural baseline of software is considered a highly relevant specification, which guides the development of software in compliance with the requirements agreed upon in Requirements Engineering. However, the computer literature presents evidence that developers often choose to start coding without at least starting a prior architectural specification. Such practice commonly compromises the evolutionary maintenance of the developed software, making it especially costly.

Considering that both the specification of a baseline of requirements and the specification of an architectural baseline are represented by models at different levels of abstraction, which allow (i) to model the requirements elicited with stakeholders, and (ii) to model the architectural components and connectors, there is a strong association of the modeling activity in both cases.

In view of the above, and seeking to collaborate in this scenario, this work was concerned with the development of an application, called Model IT, working in the semiautomatic generation of code using models as a basis.

However, to enable the development of a first version of this application, it was established: (i) a specific application profile for code generation, in this case, web applications; (ii) a specific modeling notation, in this case the notation i *; (iii) a specific architectural standard, in this case the Model View-Controller (MVC), and (iv) a specific programming language, in this case, oriented to the Django framework. The idea is to support developers in building web applications, considering the semiautomatic generation of code in Django, focusing on i * models (at different levels of abstraction) and the MVC Architectural Standard, and making use of transformational heuristics as well as principles of Model Driven Architecture (MDA).

In addition, Model IT details are presented as well as the results obtained with the use of this support with the target audience, considering action-action cycles, which allowed to evaluate the use of the support and to perform evolutions in the same based on the data collected in each cycle.

Key-words: Architecture, Model Driven Architecture, i*, Model View Controller, Code generation

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MDA	Model Driven Architecture
MVC	Model-View-Controller
MVT	Model-View-Template
UML	Unified modeling language
TCC	Trabalho de Conclusão de Curso
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
SD	Strategic Dependency Model
SR	Strategic Rationale Model
CIM	Computer Independent Model
PIM	Platform Independent Model
PSM	Platform Specific Model
SADT	Structured analysis and design technique
SPA	Single-Page-Application

Sumário

1 Introdução

Neste capítulo, será apresentado informações iniciais necessárias para o entendimento do trabalho, dividido em contextualização (Seção 1.1), as justificativas (Seção 1.3), a questão de pesquisa (Seção 1.2), os objetivos geral (Seção 1.4.1) e específicos (Seção 1.4.2) bem como a organização dessa monografia em capítulos (Seção 1.5).

1.1 Contextualização

Arquitetura de software pode ser entendida como um nível de *design*, conhecido também como nível estratégico, o qual se preocupa com questões que vão além dos algoritmos e dados. Trata-se de um conjunto de componentes de software, suas propriedades externas, e seus relacionamentos com outros componentes e/ou produtos de software. Segundo Taylor (2010), Arquitetura de Software é uma disciplina de foco primário no escopo da Engenharia de Software, dada a sua importância como linha de base no desenvolvimento de qualquer software. Essa linha de base é estabelecida como um conjunto de decisões relevantes, tomadas em tempo de *design*, e que visa a construção de um software de qualidade.

Existe uma correlação entre a Engenharia de Requisitos, que segundo Pressman (2000), fornece meios para entender e analisar as necessidades e viabilidades do software, e a Arquitetura de Software. Esta correlação dá-se via, principalmente, atividade de modelagem (SOMMERVILLE, 2011). Trata-se do desenvolvimento de modelos abstratos que representam uma visão ou perspectiva do desenvolvimento do software. Esses modelos são utilizados durante a Engenharia de Requisitos, ajudando na extração dos requisitos. Já em *design*, esses modelos são utilizados para detalhar a arquitetura adotada, facilitando a compreensão da mesma por parte dos Engenheiros de Software. Essa abrangência, reforça a importância das modelagens no contexto da Engenharia de Software (SOMMERVILLE, 2011).

Atualmente, há várias maneiras de elaborar tais modelagens, sendo a notação **UML**(*Unified Model Language*) uma das mais conhecidas e utilizadas para a modelagens arquiteturais pequenas (PRESSMAN, 2000). Trata-se de uma notação padrão, a qual é adotada pela comunidade de software para modelagem de diferentes artefatos, representando diferentes visões do software. Inclusive, é a notação utilizada pela OMG (2017a)(*Object Management Group*), uma organização internacional que aprova padrões abertos para aplicações orientadas a objetos.

Segundo Richards (2015), é relativamente comum os desenvolvedores iniciarem

a codificação sem ao menos iniciar uma especificação arquitetural prévia, que satisfaça o perfil do software a ser desenvolvido. Com isso, são causados diversos problemas no software, desde a reutilização até a manutenção.

O presente Trabalho de Conclusão de Curso procurou contribuir com a geração semi-automática de código usando modelos como base, para ajudar os desenvolvedores a não deixarem de atuar uma etapa fundamental do ciclo de vida do desenvolvimento de um software. Portanto, através da especificação e/ou transformação de modelos, em diferentes níveis de abstração, é possível a geração do código. Essa geração não é plena. A intenção é gerar um código preliminar, uma espécie de *template* de código, procurando melhorar a produtividade do Engenheiro de Software. Entretanto, gerar código, mesmo que preliminar, para qualquer tipo de software, considerando diferentes arquiteturas bem como qualquer porte de produto de software, não é uma tarefa fácil. Portanto, foi escolhido focar em um padrão arquitetural, tipicamente utilizado para o desenvolvimento de aplicações Web.

Nesse escopo, de geração de código/arquitetura dirigido(a) a modelos, pode-se estabelecer uma associação desse trabalho com o *framework* arquitetural denominado MDA (*Model-Driven Architecture*) (MELLOR et al., 2004). Este framework permite a construção de modelos, sem o conhecimento prévio de outros modelos já elaborados para o sistema. Posteriormente, permite combiná-los, visando aprimorar o sistema. Tal proposta possibilita tornar a aplicação independente de tecnologia, com arquitetura interoperável (MELLOR et al., 2004). Os modelos são elaborados em diferentes níveis de abstração. Portanto, tem-se especificação em alto nível de abstração e especificação em baixo nível de abstração, bem próxima de código. Tais modelos são especificados independentemente de tecnologias e/ou linguagens de programação. Sendo possível, portanto, implementar o software em diferentes linguagens, a partir dessas especificações.

Com base na literatura (JACYNTHO; SCHWABE; ROSSI, 2002); (PRAJAPATI; DABHI, 2009), foi possível identificar o padrão arquitetural MVC (*Model-View-Controller*) como sendo um dos mais adequados para o desenvolvimento de aplicações Web. Dessa forma, esse padrão arquitetural será utilizado para orientar a definição da linha de base arquitetural da abordagem proposta nesse trabalho.

Ao investigar os temas foco para elaboração da proposta do presente trabalho, percebeu-se maior atuação da comunidade de software usando o conceito de MDA com a notação de modelagem UML (VIEIRA; REIS, 2015). Entretanto, existem abordagens de modelagem mais emergentes (ALMISNED; KEPPENS, 2010; YRJöNEN; MERILINNA, 2008). Ainda de acordo com a mesma investigação, não foi possível encontrar o uso dos princípios do *framework* MDA orientando-se por essas abordagens mais emergentes, tal como a notação i* (DALPIAZ; FRANCH; HORKOFF, 2016).

O i* é um *framework* conceitual, o qual acorda uma notação para modelagem

de artefatos orientados à intencionalidade ou à meta. Tais modelos são diferenciados em relação às modelagens tradicionais, pois se preocupam com os porquês, o como, entre outros detalhes. Dessa forma, são modelos que permitem manter os rastros quanto às decisões em tempo de *design*, permitindo anotar alternativas (representadas como tarefas ou meios) para atingir objetivos específicos (representados como metas). Adicionalmente, permitem especificar requisitos não funcionais (representados como metas flexíveis) bem como os impactos da realização de cada tarefa em relação às metas flexíveis estabelecidas. Percebe-se, diante do exposto, que se trata de uma notação mais abrangente, mais completa. Maiores detalhes quanto a essa notação serão cobertos no capítulo de 2: Referencial Teórico, apresentado mais adiante na monografia.

Procurando atuar em um *gap* tecnológico, esse Trabalho de Conclusão de Curso procurou contribuir com a Geração de Código dirigida a Modelos Intencionais, ou Orientados à Meta. Usou como linha de base arquitetural o padrão MVC, e focou no desenvolvimento do esqueleto de código de aplicações Django. A intenção foi prover código - uma versão preliminar do mesmo - a partir de modelos Orientados à Meta, os quais serão especificados em diferentes níveis de abstração. Sendo assim, tem-se a aplicação dos princípios e boas práticas de MDA.

1.2 Questão de Pesquisa

Este trabalho pretendeu investigar a seguinte questão de pesquisa: Há possibilidade de facilitar/auxiliar o trabalho de arquitetos de software, proporcionando a geração parcial de código de aplicações Web, a partir de modelos Orientandos à Meta em diferentes níveis de abstração, e orientando-se de acordo com o padrão arquitetural MVC? Em outras palavras: Há possibilidade de facilitar/auxiliar o trabalho de arquitetos de software, utilizando-se de princípios e boas práticas do framework MDA; entretanto, considerando o tripé: modelos intencionais, aplicações Web de pequeno e médio portes, e o padrão arquitetural MVC?

1.3 Justificativa

É inegável que software é um artefato muito caro. Segundo a Gartner (2016), em 2017 foram previstos U$ 357 bilhões de gastos com *software* no mundo, o que corresponde a um aumento de 7,7% em comparação ao ano anterior, que teve o gasto de U$ 333 bilhões. Contudo, mesmo com tamanho investimento, existem muitas chances de falha. Segundo ARCIDIACONO (2017), de acordo com uma publicação feita pela International Data Corporation em 2009, 25% dos projetos de Software falham totalmente, 50% exigem algum retrabalho, e de 20% a 25% não apresentam retorno de investimento .

Tendo em vista tal situação, melhorias na produção de produtos de software são desejadas. O desenvolvimento de aplicações mais confiáveis, seguras, e que funcionem adequadamente está diretamente ligado com a sua arquitetura (SPINELLIS; GOUSIOS, 2009). Portanto, torna-se pertinente o apoio ao desenvolvimento de software, orientando-se por uma linha de base arquitetural interoperável, como proposto no framework MDA.

Entre os princípios e boas práticas de MDA, destaca-se o uso de modelos; os quais são, inicialmente, desenhados em alto nível de abstração, e sistematicamente/evolutivamente transformados em modelos cada vez menos abstratos. Por fim, têm-se modelos bem técnicos, específicos, os quais são mapeados mais facilmente para código. Com o intuito de apoiar o trabalho do arquiteto de software bem como de explorar aspectos ainda pouco investigados pela comunidade, procurou-se desenvolver uma abordagem que permite gerar códigos parciais de aplicações Web, orientando-se pelo padrão arquitetural MVC; aplicando conceitos de MDA, e fazendo uso de modelos Orientados à Meta ou Intencionais.

1.4 Objetivos

O principal intuito desse trabalho foi alcançar os objetivos geral e específicos apresentados, respectivamente, em 1.4.1 e 1.4.2

1.4.1 Objetivo Geral

Gerar parte do código, semi-automaticamente, de aplicações Django, orientando-se por uma linha de base arquitetural bem como fazendo uso de princípios de MDA e modelagem intencional (no caso, na notação i*).

1.4.2 Objetivos Específicos

Visando atingir o objetivo geral, foram estabelecidos alguns objetivos específicos, conforme colocado a seguir:

1. Investigar os temas chave para o escopo do Trabalho. Particularmente, serão aprofundados os estudos sobre Arquitetura de Software, MDA, modelagem i*, e padrão arquitetural MVC. Nesse caso, foi utilizada a pesquisa bibliográfica como base para condução desses estudos.

2. Levantamento de boas práticas da Engenharia de Software a serem empregadas na geração de código, tais como técnicas de programação (MARTIN, 2008) e padrões de projeto (GAMMA et al., 1995).

3. Desenvolvimento de uma aplicação que auxilie arquitetos e/ou desenvolvedores de software no desenvolvimento de aplicações Web utilizando o *framework* Django, orientando-se pelo padrão arquitetural MVC. Tal aplicação procurou automatizar parte do processo de geração de código.

4. Geração parcial do código a partir de modelos i*.

5. Coleta e avaliação de resultados preliminares.

1.5 Organização do Documento

Este documento apresenta-se dividido em capítulos, sendo eles:

- **Capítulo 2 Referencial Teórico:** Apresenta as áreas e conceitos chave para o desenvolvimento do trabalho.
- **Capítulo 3 Suporte Tecnológico:** Apresenta as tecnologias que conferiram suporte tanto ao desenvolvimento do trabalho quanto da pesquisa.
- **Capítulo 4 Metodologia:** Apresenta os processos metodológicos que guiaram a pesquisa, bem como o desenvolvimento da aplicação Model IT.
- **Capítulo 5 Model IT:** Apresenta a aplicação final desenvolvida.
- **Capítulo 6 Resultados Obtidos:** Apresenta os resultados dos ciclos de pesquisa-ação realizados.
- **Capítulo 7 Considerações Finais:** Apresenta as conclusões do trabalho.

2 Referencial Teórico

Neste capítulo, serão abordadas as principais referências teóricas que embasaram o presente Trabalho de Conclusão de Curso. Organizado em seções, o capítulo contextualizará sobre Arquitetura de Software na seção 2.1, seguido pelo padrão arquitetural MVC, comumente utilizado para o desenvolvimento de aplicações Web. A seção 2.2 abordará um *framework* arquitetural denominado MDA. Na seção 2.3, será descrita a atividade de modelagem de software, seguida da modelagem orientada à meta. Por fim, serão apresentadas as considerações finais do capítulo.

2.1 Arquitetura de Software

Conforme já colocado no Capítulo 1, a Arquitetura de Software, preocupa-se, principalmente, com as associações entre os componentes de software, sendo um aspecto relevante no que compreende o desenvolvimento de um produto de software, e é também um fator decisivo para o resultado final de qualquer sistema (SPINELLIS; GOUSIOS, 2009).

Entretanto, por diversas razões, é comum os desenvolvedores codificarem sem especificar uma arquitetura condizente com o perfil de software a ser desenvolvido. Assim, criam-se módulos de código que costumam não viabilizar a reutilização de software bem como não são manuteníveis (RICHARDS, 2015).

Para ajudar os desenvolvedores e arquitetos de software, existem os padrões arquiteturais. Tais padrões de arquitetura ajudam a pré-definir uma série de características básicas do sistema, bem como o comportamento entre os módulos de uma aplicação (RICHARDS, 2015). Adicionalmente, por serem padrões arquiteturais, permitem auxiliar o desenvolvimento usando a *baseline* de requisitos, estabelecida na Engenharia de Requisitos, orientando-se por diferentes artefatos de projeto, os quais modelam - sob diferentes perspectivas - os componentes de software a serem implementados.

Um desses padrões arquiteturais, comumente utilizado no desenvolvimento de aplicações Web, é o MVC (*Model-View-Controller*). Esse é apresentado na próxima seção, e foi utilizado como base nesse Trabalho de Conclusão de Curso.

2.1.1 Padrão Arquitetural Model-View-Controller (MVC)

O padrão arquitetural *Model-View-Controller* (MVC) é um padrão amplamente utilizado e aplicado para sistemas de software interativos (KALELKAR; CHURI; KALELKAR, 2014). O MVC, como apresentado na Figura 1, está modularizado em três grupos de responsabilidades, denominados de *Model*, *View* e *Controller*. Os componentes de um

software devem se comunicar uns com os outros de forma protocolada. Caso contrário, as responsabilidades podem não ser mapeadas adequadamente.

O componente *Model* é responsável por encapsular os dados do sistema, ou seja, é independente do comportamento do restante do sistema (KALELKAR; CHURI; KALELKAR, 2014). No componente *Model* é descrita a camada de dados do sistema, e ela possui a responsabilidade sobre estes dados. Dessa forma, colabora apenas com a camada *Controller*, tornando-a uma intermediadora entre a *Model* e a *View*.

O componente *View* deve assegurar que, independente do estado o qual o componente *Model* esteja, sua apresentação ao usuário deve ser fidedigna a ele (PRAJAPATI; DABHI, 2009). Em outras palavras, o componente *View* apresenta ao usuário o estado atual do componente *Model*. Tal comunicação, entre *View* e *Model*, é intermediada via *Controller*.

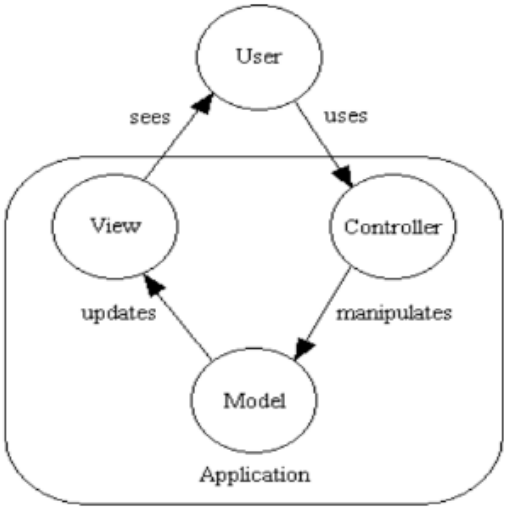


Figura 1 – Arquitetura MVC (KALELKAR; CHURI; KALELKAR, 2014)

Por fim, o componente *Controller* é o responsável por possuir o controle dos dados que serão perpassados do componente *Model* ao componente *View*. Também deve possuir um comportamento dinâmico, ou seja, como ele é responsável pelo controle dos componentes *View*, se uma *Model* for modificada, isso deverá impactar o componente *View* (KALELKAR; CHURI; KALELKAR, 2014), como mostra a Figura 2.

Existem *frameworks* conceituais que acordam soluções dirigidas por boas práticas e padrões para condução do desenvolvimento de software, orientando-se pelo tópico Arquitetura de Software, conforme apresentado na seção 2.2.

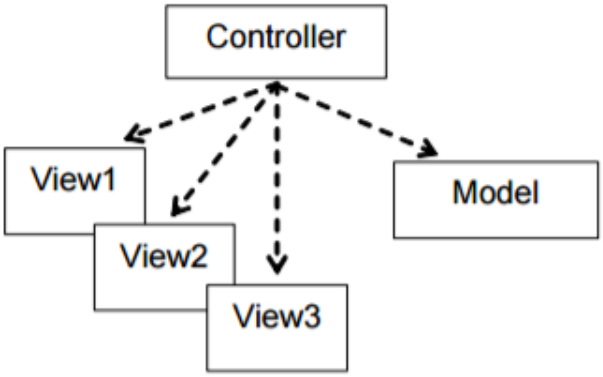


Figura 2 – Exemplo Controller (LITVIN; LITVIN, 2007)

2.2 Arquitetura Dirigida por Modelos (MDA)

Trata-se de um *framework* arquitetural que se baseia em modelos e transformações de modelos. Permite a construção do software, ganho de produtividade, interoperabilidade, integração e, principalmente, portabilidade (VIEIRA; REIS, 2015).

A MDA (do inglês, *Model Driven Architecture*) baseia-se em especificações que começam em alto nível de abstração. Esse nível vai sendo reduzido gradualmente, até especificações/modelos cada vez mais técnicos, ou seja, próximos de código. Adicionalmente, adota a proposta de representar ideias em uma especificação globalizada/única, sendo esse princípio conhecido como interoperabilidade em tempo de projeto (MELLOR et al., 2004)

Outro princípio chave de MDA é especificar uma solução que seja independente de plataforma e/ou linguagem de programação. Essa solução será concretizada em um produto de software usando como base modelos em diferentes nível de abstração, como se passasse por etapas, viabilizadas por transformações no nível de modelagem.

Em uma primeira etapa, um modelo de nível mais elevado de abstração pode ser visto como livre de computação, conhecido em MDA como CIM (*Computation Independent Model*. Esse modelo é pouco utilizado na comunidade de software, justamente pelo seu alto nível de abstração. Depois, em uma segunda etapa (para alguns autores, a primeira etapa de fato), tem-se um modelo de abstração intermediário, independente de plataforma, conhecido em MDA como PIM (*Platform Independent Model*).

Em uma terceira etapa, tem-se o modelo no nível mais baixo de abstração, visto como específico de plataforma, sendo conhecido em MDA como PSM (*Platform Specific Model*). Pode-se ter mais de um modelo especifico de plataforma. Por fim, em uma etapa de fechamento, tem-se a geração de código a partir desses modelos específicos de plataforma.

Nesse Trabalho de Conclusão de Curso, foram utilizados alguns desses princípios e boas práticas de MDA, em uma versão adaptada desse *framework* conceitual, visando o desenvolvimento de aplicações Web, no padrão arquitetural MVC. Para tanto, dada a importância da atividade de modelagem, cabe a definição do termo Modelo, apresentada na seção 2.2.1.

2.2.1 Modelos

Segundo o MDA GUIDE 2.0 (2014), o objetivo principal da arquitetura MDA é poder derivar valor dos modelos produzidos e, dessa forma, obter as informações necessárias para lidar com a complexidade e a interdependência de sistemas complexos.

Stephen J. Mellor (2004), trata os modelos como sendo um aglomerado de elementos que juntos descrevem uma realidade seja física, abstrata ou hipotética, e que modelos bem formulados podem ser utilizados como uma forma de comunicação.

Todo modelo pode ser definido através de um metamodelo. Esse, por sua vez, consiste em um modelo usado para descrever, representar, especificar modelos, orientando-se por uma linguagem de modelagem, com notação específica. Modelos podem ter relacionamentos semânticos com outros modelos de diferentes níveis abstracionais. Para isso, é definido um mapeamento, normalmente, com heurísticas que permitem entender, associar, converter as abstrações de um modelo nas abstrações de outro modelo.

Dificilmente, dadas as particularidades de cada modelo, é possível obter um mapeamento um para um entre as abstrações dos modelos envolvidos. Quando se deseja estabelecer um mapeamento e/ou correspondência entre as abstrações, é recorrente o uso de marcas ou modelos de marcação, conforme ilustrado na Figura 3 , que representa um metamodelo transformacional de modelos. Tal figura representa um diagrama que mostra como um modelo é transformado em outro.

Diante do exposto, e dada a necessidade de mapeamento entre modelos, seguem detalhes de como se dá a transformação entre modelos, orientando-se especificamente pelos princípios de MDA. Lembrando que uma visão mais generalista já foi acordada anteriormente. Entretanto, na seção 2.2.2, a intenção é apresentar uma visão mais focada nos aspectos transformacionais envolvendo modelos em diferentes níveis de abstração.

2.2.2 Transformação de Modelos

Os modelos em MDA respeitam um ciclo de vida transformacional, ou seja, eles se modificam a cada mapeamento realizado, e são classificados em três camadas, como mostrado na Figura 4 . Segundo Vieira (2015), as camadas são: *Computation Independent Model* (CIM), *Platform Independent Model*(PIM) e, por fim *Platform Specific Model*

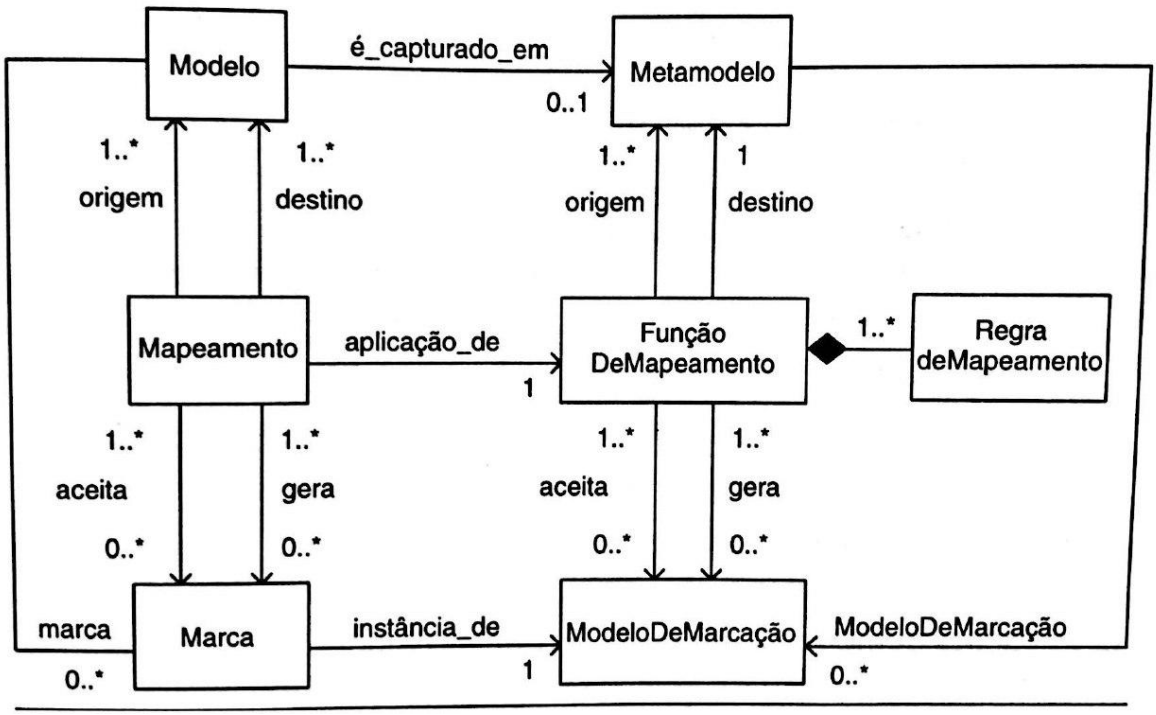


Figura 3 – Marcas e modelos de marcação (MELLOR et al., 2004)

(PSM). O modelo CIM, é um modelo alto nível, a nível de negócio. As transformações abaixam o nível abstracional deste, até chegar ao PSM, e depois ao código.

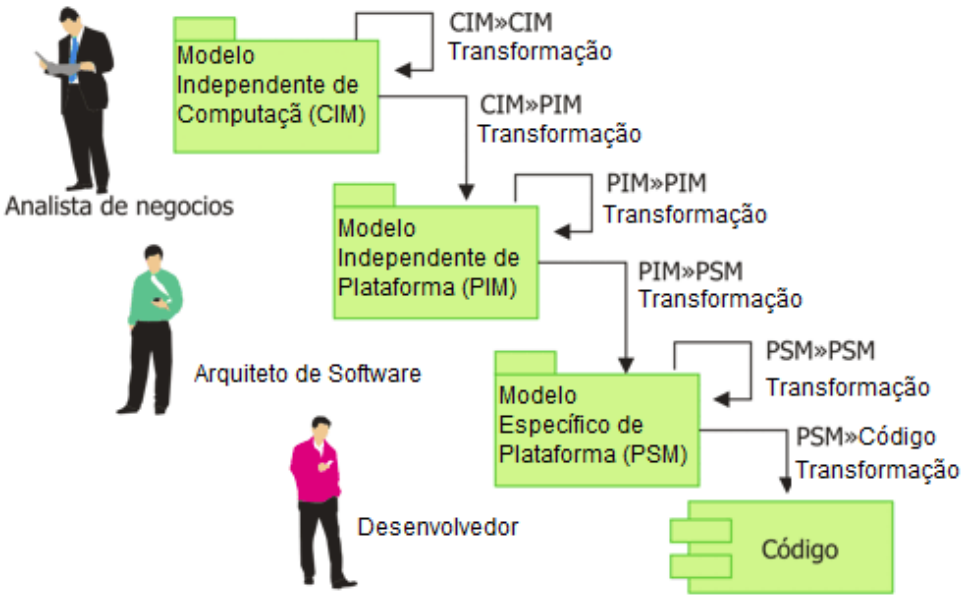


Figura 4 – Ciclo de Vida Traduzido de (VIEIRA; REIS, 2015)

2.2.2.1 *Computation Independent Model* (CIM)

Trata-se de um modelo que possui como foco primário expressar os requisitos do sistema, ou seja, é um modelo que deve ser mais próximo ao mundo real, sendo independente de tecnologia, plataforma ou mesmo linguagem de programação. Além disso, seu objetivo é ser claro para que, através dos mapeamentos realizados, possam ser gerados o PIM e o PSM, para a construção do software em questão (VIEIRA; REIS, 2015).

2.2.2.2 *Platform Independent Model*(PIM)

Como já mencionado, um PIM descreve as características de domínio como as classes e seus atributos (VIEIRA; REIS, 2015). Porém, como um PIM pode representar um ou mais modelos de plataforma (PSM), logo, não é necessário explicitar todos os detalhes. Por exemplo, não há necessidade de especificar como lidar especificamente com segurança ou persistência de dados. Pode-se optar por deixar tal responsabilidade para os modelos mais específicos de plataforma. (MELLOR et al., 2004).

2.2.2.3 *Platform Specific Model* (PSM)

Esse modelo é especificado em um nível mais baixo de abstração, permitindo detalhar o que será desenvolvido orientando-se por uma determinada tecnologia. Contudo, novamente, nem tudo precisa ser detalhado. É possível, portanto, restringir algumas particularidades sobre a implementação, permitindo que esse modelo mantenha-se abstrato para que seja reutilizado mais facilmente em diferentes plataformas e/ou linguagens de programação (VIEIRA; REIS, 2015).

Nesse Trabalho de Conclusão de Curso, usou-se de transformações entre modelos no intuito de facilitar o desenvolvimento de aplicações Web no padrão arquitetural MVC. Mas, existe uma particularidade extra na escolha dos modelos base para condução desse processo de desenvolvimento. Como foram observadas muitas iniciativas usando MDA e a notação UML (OMG, 2017b), tais como:(ZARRAS, 2006) e (MAZóN, 2008); optou-se por contribuir usando alguns princípios de MDA, mas orientando-se por modelos mais emergentes, associados à Modelagem Intencional ou Modelagem Orientada à Meta (ABID et al., 2009).

2.3 Modelagem Orientada à Meta

Modelar orientando-se à meta significa preocupar-se com os motivos existentes por trás de um determinado requisito (CARES; GUTIéRREZ, 2009). Em outras palavras, a modelagem orientada à meta preocupa-se em representar não apenas o como, o quando e o onde de um sistema ou organização, mas também o "porquê"de cada um desses questionamentos. Segundo Cares 2009, existem duas linguagens de modelagem que estão em

evidência, que são KAOS e o *framework* i*. Neste trabalho, foi utilizado o *framework* i*, e o mesmo encontra-se descrito na seção 2.3.1.

2.3.1 *Framework* i*

A linguagem de modelagem *i** (YU, 1997) foi desenvolvida nos anos noventa e surgiu com o objetivo de ser capaz de exprimir características organizacionais como a intencionalidade (por que?), sociais (quem?), e estratégicas (como?); ou seja, surgiu com o objetivo de ser uma linguagem de modelagem orientada à meta e a atores (DALPIAZ; FRANCH; HORKOFF, 2016).

Segundo Yu (1997), o *framework* prevê dois nível de modelagens, representados por modelos diferentes. Um primeiro modelo, chamado modelo de Dependência Estratégica (SD - *Strategic Dependency Model*), o qual consiste em descrever as relações de dependências entre os vários atores existentes em um ambiente organizacional. Um segundo modelo, chamado modelo de Racionalidade Estratégica (SR - *Strategic Rationale Model*), o qual é usado para expressar interesses e preocupações dos atores interessados.

2.3.1.1 Modelo de Dependência Estratégica (SD)

O modelo de Dependência Estratégica (SD) é uma representação gráfica, na qual cada ator é representado por um círculo. A relação de dependência entre dois atores é representada pela ligação entre eles. Na ligação entre os atores, possui uma forma geométrica chamada de *dependum* Figura 5, que representa o objetivo foco do ator chamado de *depender*.

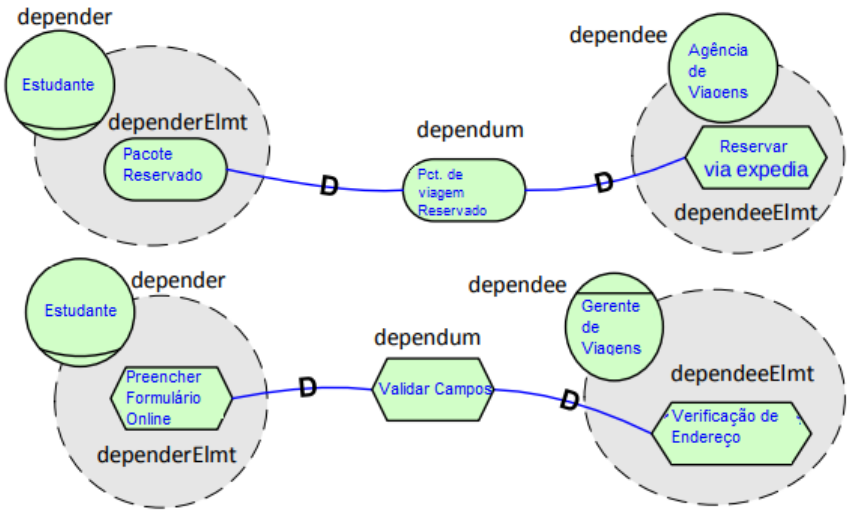


Figura 5 – Exemplo de Modelo de Dependência Estratégica. Traduzido de(DALPIAZ; FRANCH; HORKOFF, 2016)

Tal objetivo só será cumprido graças a ajuda do outro ator chamado *dependee*, justificando-se assim a dependência entre os atores. Dentro da fronteira dos atores, existem os elementos que auxiliam os atores a chegarem em seus objetivos e/ou seus objetivos e tarefas pessoais. Por exemplo, o exemplo 1 da Figura 5 é a representação de dependência estratégica do seguinte contexto: O estudante tem o objetivo de reservar um pacote. Para tal, depende da Agência de Viagens para realizar a reserva do pacote de viagem, realizando-a através da expedia.(YU, 1999).

2.3.1.2 Modelo de Racionalidade Estratégica (SR)

O modelo de racionalidade estratégica é uma representação gráfica que exprime o raciocínio dos atores sobre suas relações (mostradas pelo modelo SD). Os nós do modelo podem representar metas, tarefas, recursos e metas flexíveis (*softgoals*), e o objetivo do modelo SR é deixar explícito como cada ator lidará com seus respectivos nós de acordo com a dependência existente com outros atores.

Segundo Yu 1999, um modelo SR pode fornecer análises sobre capacidade de trabalho e viabilidade. Pode apoiar a criação de questões, identificação e exploração de alternativas, bem como o reconhecimento de problemas correlacionados e a resolução de tais problemas.

Ao observar a Figura 6, pode-se perceber que esta representa o modelo de racionalidade estratégica do exemplo de modelo de dependência estratégica da Figura 5.

2.4 Resumo do Capítulo

A utilização de tais referências para a construção do presente Trabalho de Conclusão de Curso apoiou algumas das principais atividades no desenvolvimento de um software, sendo elas modelagem do software, arquitetura de software e o código.

Cada uma dessas atividades utilizou como base a literatura prescrita neste capítulo, ou seja, a modelagem do software contou com o *framework* de modelagem i*, orientada a meta. A arquitetura do software teve o apoio do *framework* arquitetural MDA e também do padrão arquitetural MVC. A geração do código foi apoiada também em princípios MDA, com foco na transformação de modelos.

A Figura 7 exprime tal mapeamento, sendo as atividades representadas na cor preta, as que foram orientadas pelo referencial teórico.

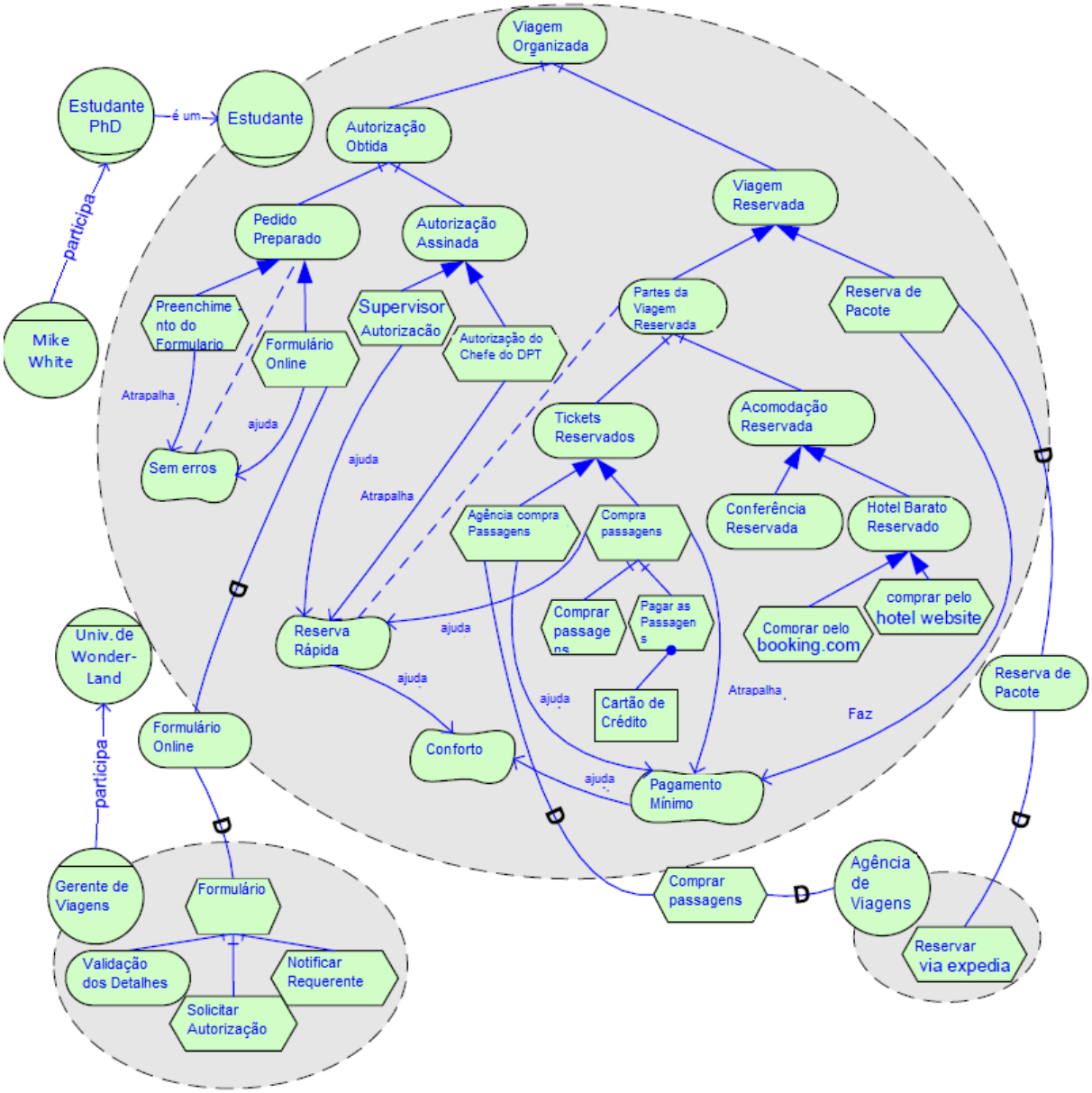


Figura 6 – Exemplo de Modelo de Racionalidade Estratégica. Traduzido de(DALPIAZ; FRANCH; HORKOFF, 2016)

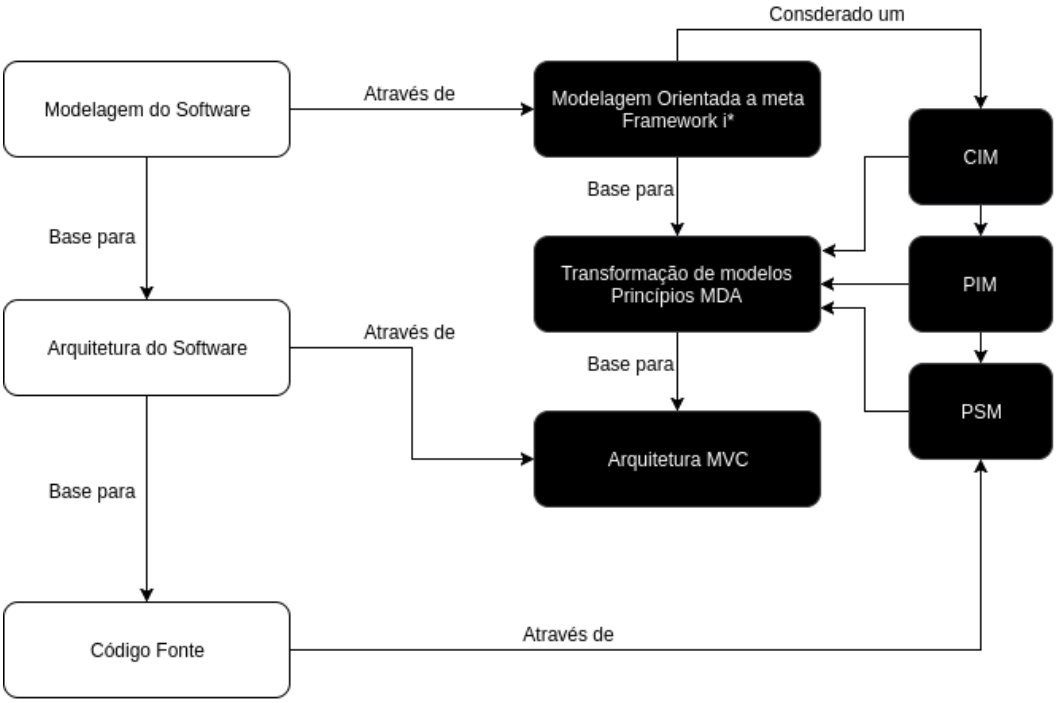


Figura 7 – Mapeamento das atividades

3 Suporte Tecnológico

Neste capítulo, serão apresentadas as principais ferramentas e tecnologias que deram suporte ao desenvolvimento desse trabalho. O capítulo foi dividido em três sub seções, sendo a primeira, Seção 3.1, sobre o desenvolvimento da aplicação; a segunda, Seção 3.2, sobre a engenharia de software, e a terceira, Seção 3.3 sobre escrita e condução da pesquisa.

3.1 Desenvolvimento da Aplicação

Esta seção contemplará as tecnologias que foram utilizadas na construção do código da aplicação.

3.1.1 Javascript ECMAScript 6

Segundo a (MOZILLA, 2017), JavaScript é uma linguagem de programação que permite implementações mais complexas em páginas web, ou seja, com JavaScript é possível à uma aplicação web ir além do que apenas exibir dados. É possível também criar conteúdos dinâmicos, realizar o controle de multimídias, criar imagens animadas e gráficos. Esta foi a principal linguagem utilizada na construção da aplicação.

3.1.1.1 GO.js

Segundo o site oficial da GO.js (GO.JS, (acessado em 17 de setembro de 2017)), trata-se de uma biblioteca de JavaScript puro, com diversos recursos para se implementar diagramas interativos nos navegadores e plataformas atuais. Portanto, trata-se de uma biblioteca facilita a construção de diagramas de JavaScript.

A versão utilizada foi a LTS, que atualmente é 1.7.26, e foi utilizada na construção do módulo de modelagem da aplicação.

3.1.1.2 Vue.js 2.0

O Vue é um *framework* progressivo que auxilia na construção de *interfaces* de usuário. Porém, diferente dos outros *frameworks* monolíticos, sendo projetado para lidar com questões de componetização, e pensado para ser focado exclusivamente na camada visual. Adicionalmente, Vue é capaz de projetar SPA (*Single-Page-Applications*) quando combinado com ferramentas adequadas (VUE.JS, 2017).

Foi utilizado o Vue.js para a manipulação do cliente da aplicação, acoplando em si o módulo de modelagem utilizando Go.js.

3.1.1.3 NODE.js

NODE é uma biblioteca JavaScript que foi projetada para criar aplicações de rede escaláveis, sendo orientada por eventos assíncronos (NODE.JS, 2017). Trata-se, portanto, de uma biblioteca JavaScript capaz de criar aplicações *web* assíncronas. A versão utilizada foi a LTS, que atualmente é 6.11.3, e foi utilizada para lidar com o servidor da aplicação.

3.1.2 Python 3

Python é uma linguagem de programação interpretada e orientada a objetos. Possui interface para bibliotecas e chamadas de sistema, sendo extensível às linguagens C e C++. Por fim, cabe ressaltar que é uma linguagem portátil, rodando em Unix, no Mac e no Windows 2000 e posterior (PYTHON, 2017). Esta linguagem foi a utilizada no esboço do código gerado. Em outras palavras, o código gerado é python.

3.1.2.1 Django

Trata-se de um *framework* escrito em Python para a construção de aplicações *web*, que incentiva o desenvolvimento rápido. O Django cuida de muitos problemas comuns no desenvolvimento para a Web, para que se possa concentrar no domínio da aplicação. É gratuito e de código aberto (DJANGO, 2017). A versão utilizada foi a LTS, que atualmente é 1.11.5, e foi utilizada para adequar a arquitetura proposta à linguagem Python.

3.1.2.2 MongoDB

RethinkDB (MONGODB, 2017) é um banco de dados baseado em JSON, e não relacional. Possui grande escalabilidade, performance e conta com uma comunidade bem ativa. A versão utilizada foi LTS 3.6, e foi empregado para a persistência das informações das modelagens realizadas utilizando o módulo construído em GO.js.

3.2 Engenharia de Software

Esta seção apresentará os suportes tecnológicos associados mais especificamente à Engenharia de Software. A mesma foi dividida em três sub seções, sendo elas: Gerenciamento de Projetos, Gerenciamento de Desenvolvimento e Gerenciamento de Configuração.

3.2.1 Gerenciamento de Projeto

Esta subseção diz respeito sobre as metodologias e ferramentas adotadas para gerenciamento do projeto, no que tange o controle das atividades realizadas bem como o gerenciamento do desenvolvimento e de configurações da aplicação Model IT.

3.2.1.1 Trello

(TRELLO, 2017) é uma ferramenta *online* e gratuita para o gerenciamento de projetos e atividades através de cartões, utilizando a função *drag and drop* entre listas. Com Trello, é possível compartilhar arquivos e *feedbacks de atividades*.

Com a ajuda do Trello, as atividades executadas bem como suas descrições e prazos foram gerenciados neste trabalho.

3.2.2 Gerenciamento de Desenvolvimento

Para a construção do código da aplicação, foi utilizado o editor de texto Visual Studio Code, e o sistema operacional Ubuntu 16.04.

3.2.2.1 Visual Studio Code 1.16.1

Visual Studio Code (VSCODE, 2017) é um editor de texto ou código fonte da Microsoft(MICROSOFT, 2017) que possui ferramentas poderosas que auxiliam no desenvolvimento de software, como por exemplo conclusão e depuração de código *IntelliSense*(Sugestões inteligentes de auto preenchimento de algum parâmetro ou atributo no código). Além disso, o editor suporta diversas linguagens de programação, e possui um acervo grande de *plugins*, visando tornar seu trabalho mais eficiente.

3.2.2.2 Ubuntu

(UBUNTU, 2017) é uma distribuição Linux totalmente gratuita e de código aberto. Este sistema operacional, patrocinado pela Canonical(CANONICAL, 2017), foi desenvolvido utilizando o kernel linux em seu núcleo.

3.2.3 Gerenciamento de Configuração

Para a realização do gerenciamento de configuração, foi utilizadas as seguintes ferramentas:

3.2.3.1 Git 2.7.4

Projetado pela *Software freedom Conservacy* (CONSERVANCY, 2017), o Git é um sistema de controle de versionamento livre e de código aberto sob a licença *GNU General Public License versão 2.0*. Foi projetado para dar suporte a projetos de qualquer dimensão com eficácia e robustez.

3.2.3.2 GitLab

GitLab é uma aplicação *online* de código aberto para persistência de projetos, juntamente com o controle de versão Git. Possui uma wiki integrada, e permite gerenciamento de projetos privados gratuitamente (GITLAB, 2017).

3.3 Escrita e Condução da Pesquisa

Para ajudar no desenvolvimento do Trabalho de Conclusão de Curso, bem como nas pesquisas realizadas, foram utilizadas as seguintes ferramentas:

3.3.0.1 LaTeX

Trata-se de uma tecnologia para preparação de documentos, focando na qualidade de apresentação dos mesmos. Muito utilizado para textos técnicos e científicos, LaTeX (LATEX, 2017) formata o texto, permitindo que seu usuário se preocupe apenas com o conteúdo que foi transmitido. LaTeX é um software livre, distribuído sob a Licença Pública do Projeto LaTeX.

A versão utilizada foi a LaTeX2e.

3.3.0.2 Zotero

Criado pela *Roy Rosenzweig Center for History and New Media* (RRCHNM, 2017), Zotero é uma ferramenta gratuita, utilizada para facilitar o gerenciamento de suas fontes de pesquisa, podendo auxiliar tanto nas citações quanto no compartilhamento das fontes.

Permite ainda detectar o conteúdo do navegador, adicionando-o na biblioteca pessoal (HISTORY; MEDIA, 2017).

A versão utilizada foi a versão 5.0.

3.4 Resumo do Capítulo

Para contextualizar a utilização das tecnologias mencionadas, a Tabela 1 acorda cada tecnologia, apresentando nome, descrição e versão utilizada.

Tabela 1 – Tecnologias utilizadas

Nome	**Descrição**	**Versão**
JavaScript	Linguagem de programação para web	EcmaScript 6
Go.js	Biblioteca para construir diagramas	1.7.26
Vue.js	Framework progressivo de interface	2.0
NODE.js	Biblioteca JavaScript para aplicações assíncronas na web	6.11.3
Python	Lnguagem de programação interpretada e orientada a objetos	3
Django	framework Python para criação de aplicações em MVC	1.11.5
MongoDB	Banco de dados baseado em JSON para persistência de informações	3.6
Trello	Aplicação de gerenciamento de projetos	-
Visual Studio Code	Editor de textos ou código fonte	1.16.1
Ubuntu	Sistema operacional Linux	16.04
Git	Sistema de controle de versionamento	2.7.4
GitLab	Aplicação online de persistência de projetos em Git	-
Docker	Solução de modularização de software em containers	17.06.2
LaTeX	Tecnologia para qualidade de documentos científicos	LaTeX2e
Zotero	Gerenciador de referências e bibliografia	5.0

4 Metodologia

Este capítulo apresenta a metodologia que orientou o desenvolvimento do presente trabalho. O capítulo encontra-se dividido em seções: seção 4.1, com o detalhamento da metodologia que foi adotada, sendo essa classificada de acordo com os critérios de abordagem, natureza, objetivos e procedimentos; a seção 4.2, com foco na descrição do fluxo de atividades que guiou o desenvolvimento do trabalho como um todo; e a seção 4.3, a qual apresenta o cronograma para cumprimento dos prazos.

4.1 Classificação da pesquisa

Segundo Gerhardt (2009), uma pesquisa científica, que é o resultado de uma busca minuciosa para determinado problema, pode ser classificada em vários tipos, como mosta a Figura 8:

Quanto à Abordagem	
Qualitativa	Quantitativa

Quanto à Natureza	
Básica	Aplicada

Quanto aos Objetivos		
Exploratória	Descritiva	Explicativa

Quanto aos Procedimentos					
Experimental	Bibliográfica	Documental	de Campo	Ex-Post-Facto	de Levantamento
com Survey	Estudo de Caso	Participante	Pesquisa-Ação	Etnográfica	Etnometodológica

Figura 8 – Tipos de Pesquisa

4.1.1 Quanto à Abordagem

A pesquisa quanto à abordagem qualitativa possui enfoque no aprofundamento e na compreensão dos resultados. Já uma abordagem quantitativa, além de quantificar os resultados, a visão do pesquisador é externa (GERHARDT; SILVEIRA, 2009). Dessa forma, a pesquisa classifica-se como qualitativa, pois além da visão do pesquisador ser interna, ainda possui resultados não métricos que precisam de um maior enfoque na interpretação.

4.1.2 Quanto à Natureza

Segundo Gerhardt (2009), uma pesquisa de natureza básica tem como objetivo gerar novos conhecimentos que serão interessantes para a evolução daquela área de conheci-

mento, sem aplicação prática. Já uma pesquisa aplicada, prevê a geração de conhecimento para aplicações práticas específicas. Diante do exposto, o presente trabalho enquadra-se mais como uma pesquisa de natureza aplicada.

4.1.3 Quanto aos Objetivos

De acordo com Gil (1991), uma pesquisa exploratória visa proporcionar maior conhecimento do problema, afim de aprimorar ideias, através, por exemplo, de um levantamento bibliográfico. Portanto, o atual Trabalho de Conclusão de Curso contou com uma pesquisa exploratória, uma vez que foi necessário ter uma maior familiaridade com o problema, criar hipóteses, além de ter sido necessário um levantamento bibliográfico.

4.1.4 Quanto aos Procedimentos

A pesquisa, em termos de procedimentos, pode ser classificada como: Pesquisa Bibliográfica. Essa pode ser entendida, de acordo com Gil (1991), como pertinente quando há necessidade de realizar buscas por informações, consultando livros e artigos científicos. Adicionalmente, como o autor esteve participando ativamente, ao longo das atividades que envolverem coleta de resultados (principalmente), percebeu-se a necessidade de ciclos iterativos de Pesquisa-Ação (GIL, 1991).

4.2 Fluxo das Atividades

Com base na escolha da classificação de pesquisa, e levando em consideração as atividades necessárias para a realização do trabalho, foi construído um fluxo de atividades apresentado na Figura 9. Adicionalmente, será apresentada uma breve descrição para cada atividade ilustrada no fluxo. Tal processo orientou o desenvolvimento do trabalho como um todo.

Definir Tema: Esta atividade, já realizada, teve por objetivo escolher um tema, junto aos orientadores e com base em *gaps* tecnológicos da área. Diante das pesquisas realizadas, escolheu-se trabalhar com geração de código a partir de modelos orientados à meta.

Realizar Levantamento Bibliográfico: Esta atividade, já realizada, teve por objetivo obter um conhecimento inicial dos autores bem como de publicações pertinentes aos tópicos de interesse do tema dessa monografia.

Elaborar Proposta Inicial: Esta atividade, já realizada, teve por objetivo elaborar uma proposta que embasasse e justificasse a necessidade de geração de código a partir de modelos orientados à meta. Conforme já acordado antes, optou-se por desenvolver uma aplicação que permita modelar usando a notação i* bem como gerar um código

preliminar, utilizando-se de princípios do *framework* MDA e orientando-se pelo padrão arquitetural MVC.

Definir Suporte Tecnológico: Esta atividade, já realizada, teve por objetivo levantar as tecnologias para apoiar tanto o desenvolvimento da aplicação, quanto as necessidades de gerenciamento e pesquisa desse trabalho.

Definir Referencial Teórico: Esta atividade, já realizada, teve por objetivo levantar as principais fontes conceituais para embasar o presente trabalho.

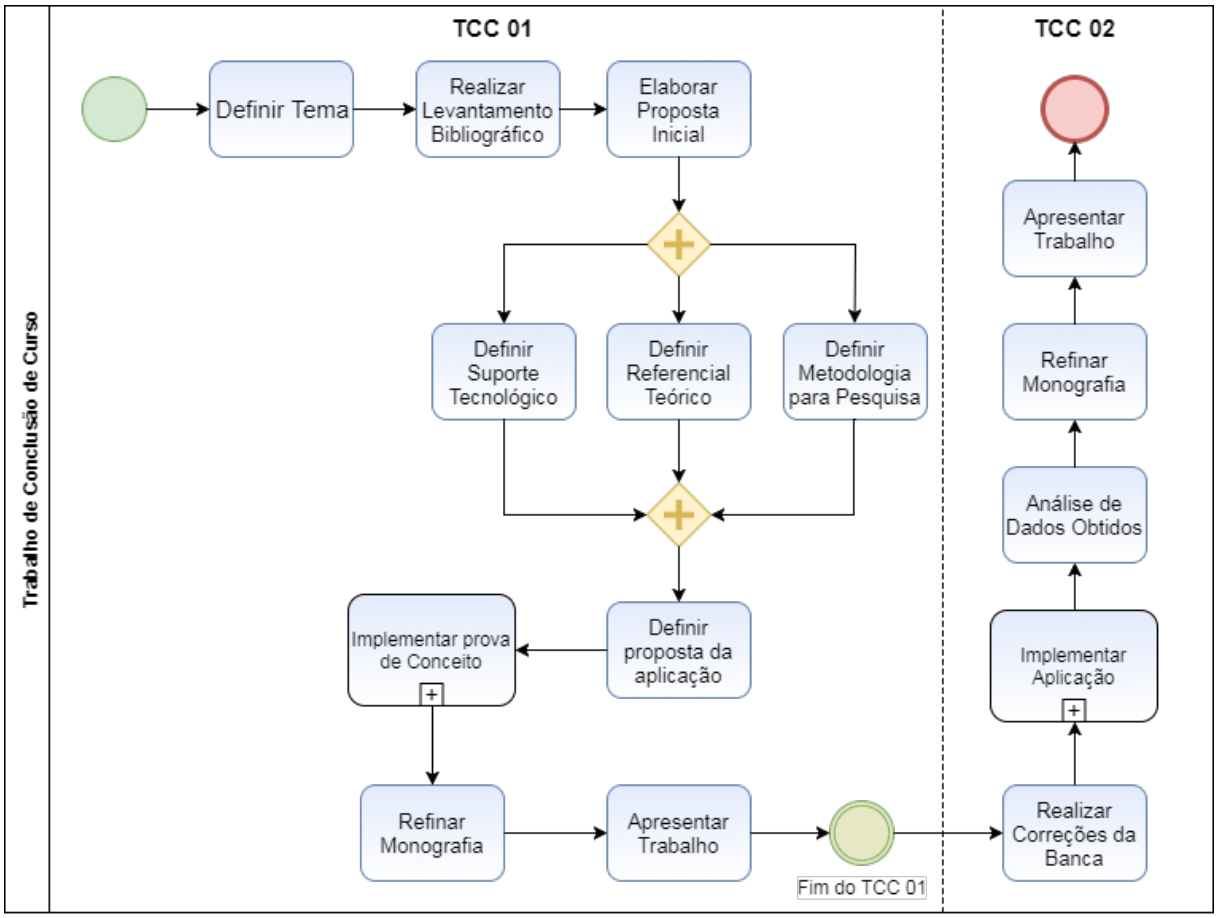


Figura 9 – Fluxo de Atividades do Trabalho

Definir Metodologia de Pesquisa: Esta atividade, já realizada, teve por objetivo definir a metodologia para guiar a pesquisa, permitindo classificar essa última quanto à abordagem, à natureza, aos objetivos, e aos procedimentos. Adicionalmente, ao conhecer melhor a pesquisa, e como resultado dessa atividade, foi possível estabelecer um fluxo de atividades bem definido.

Definir Proposta da Aplicação: Esta atividade, já realizada, teve por objetivo delimitar o escopo da proposta, estabelecendo com maior detalhamento os principais objetivos desse trabalho, em seu âmbito geral - objetivo geral - e específico - objetivos

específicos.

Implementar prova de Conceito: Esta atividade, já realizada, teve por objetivo implementar parte da proposta, desde a primeira parte do trabalho, permitindo avaliar a viabilidade do projeto que se esperava realizar, em sua plenitude, na segunda parte do trabalho. Tal atividade orientou-se pelo fluxo apresentado na seção 4.2.1.

Refinar Monografia: Esta atividade, já realizada, teve por objetivo realizar ciclos de revisão, junto aos orientadores, visando aprimorar a monografia.

Apresentar Trabalho: Esta atividade visou apresentar todas as atividades realizadas na primeira parte do trabalho, para os membros da banca, com o intuito de coletar outras impressões, as quais serviram de insumos relevantes para aprimoramento do trabalho.

Realizar Correções da Banca: Esta atividade, já realizada, objetivou revisar o trabalho, com base nas impressões coletadas junto aos membros da banca.

Implementar Aplicação: Esta atividade, já realizada, teve o objetivo de finalizar a implementação da aplicação, a qual foi iniciada na prova de conceito bem como representa o principal produto de software obtido ao final desse trabalho. Tal atividade orientou-se pelo fluxo apresentado na seção 4.2.1.

Análise de Dados Obtidos: Esta atividade, já realizada, visou analisar a aplicação, ou seja, se essa atendeu ao esperado (lê-se proposto); além de levantar se os objetivos foram alcançados.

Refinar Monografia: Esta atividade, já realizada, previu realizar novos ciclos de revisão, junto aos orientadores, visando aprimorar a monografia.

Apresentar Trabalho: Esta atividade visa apresentar o resultado final do trabalho aos membros da banca; novamente, com o intuito de coletar as impressões desses membros. Havendo necessidade, será realizada mais uma atividade para refinamento da monografia.

4.2.1 Atividades de Desenvolvimento

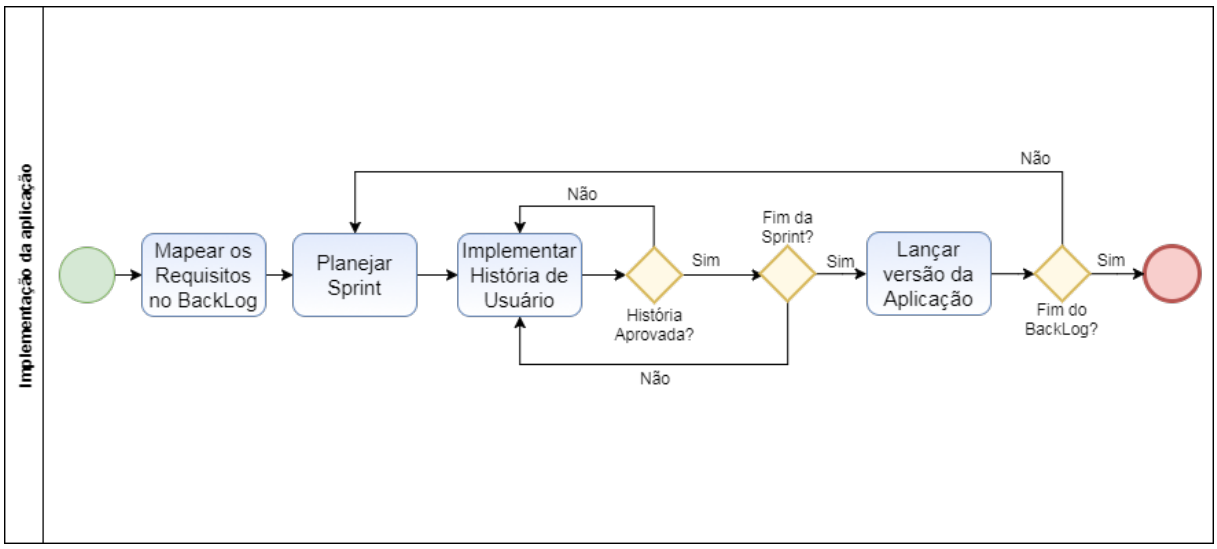


Figura 10 – Fluxo do desenvolvimento

Para a realização das atividades referentes ao desenvolvimento da aplicação, foram conduzidas práticas previstas na metodologia Scrum (SCRUM-GUIDE, 2017). Dentre essas, podemos destacar o gerenciamento de um *backlog*, *sprints* de quinze dias, e revisão da *sprint*. A Figura 10 representa o processo que foi utilizado no desenvolvimento, levando em conta tais práticas, adaptadas da metodologia Scrum.

4.2.2 Pesquisa-Ação

Segundo Gil (1991), uma pesquisa-ação contém certas atividades que não são realizadas linearmente no tempo, devido a sua natureza flexível bem como a influência do envolvimento dos pesquisadores e grupos interessados. Tais atividades estão apresentadas na Figura 11, sendo as atividades em azul as mais relevantes quanto aos resultados obtidos, tendo suas explicações na sub-seção 4.2.3. Tal seção procura apresentar como foi realizada a análise de resultados no escopo do presente trabalho, utilizando a pesquisa-ação para tal.

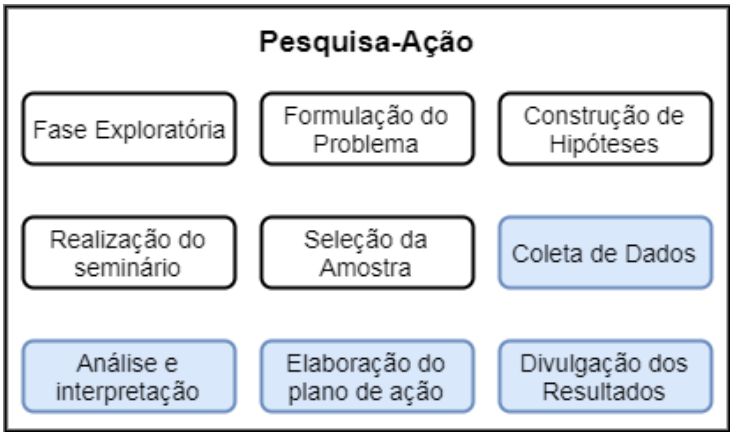


Figura 11 – Atividades de uma Pesquisa-Ação

4.2.3 Análise dos Resultados

As atividades a seguir estão de acordo com Gil (1991).

Como as atividades destacadas na Figura 11 acontecem em ciclos não definidos, foi previsto que, quando necessário, dentro do prazo da monografia, ocorressem ciclos de **coleta de dados**, **interpretação desses dados** e, por fim, a **divulgação de tais resultados**. Caso haja pontos que possam ser melhorados, um **plano de ação** seria estabelecido. Ocorreram dois ciclos de pesquisa-ação ao longo do trabalho, e estes estão expostos no Capítulo 6, de Resultados Obtidos.

Estes ciclos ocorreram para garantir que o trabalho proposto fluísse corretamente, afim de alcançar os objetivos estabelecidos no Capítulo 1.

Coleta de Dados: Esta atividade foi realizada com questionários após os usuários fazerem o uso da aplicação.

Análise e Interpretação dos Dados: Esta atividade consistiu apenas em análise e generalização dos dados para verificar se a aplicação realmente cumpria com seu objetivo ou não. Foi possível utilizando os dados obtidos através dos questionários com os utilizadores da aplicação.

Elaboração do Plano de ação: Esta atividade foi necessária para solucionar problemas encontrados, dentro do prazo e escopo desse trabalho.

Divulgação dos Resultados: Dado que as atividades anteriores ocorreram em ciclos, com o trio Coleta, Análise e Ação, a atividade de Divulgação de Resultados pode ser entendida, para o escopo desse trabalho, como os resultados obtidos ao final de cada ciclo envolvendo: Coleta de Dados, Análise e Interpretação dos Dados e Elaboração do Plano de Ação, e estão disponíveis no Capítulo 6, de Resultados Obtidos.

4.3 Cronograma

Os cronogramas a seguir explicitam as atividades executadas com seus respectivos prazos. Dessa forma, tanto a execução da primeira parte do TCC, quanto a execução da segunda parte desse trabalho, ficam mais bem gerenciadas, utilizando a Tabela 2 e a Tabela 3, respectivamente.

Para fins de uma visualização mais adequada, foi considerado com a letra **P**, a data prevista, e a letra **E**, para a data Executada.

Tabela 2 – Atividades do trabalho de conclusão de curso (Parte I)

Atividades	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Definir Tema	P-E			
Realizar Levantamento Bibliográfico	P-E	P-E		
Elaborar Proposta Inicial	P-E			
Definir Suporte Tecnológico		P-E		
Definir Referencial Teórico		P-E	E	
Definir Metodologia de Pesquisa		P	P-E	
Definir Proposta da Aplicação		P-E	P-E	
Implementar prova de Conceito	E	E	P-E	
Refinar Monografia				P-E
Apresentar Trabalho				P-E

O cronograma do Trabalho de conclusão de curso (Parte I) foi executado com sucesso.

Tabela 3 – Atividades do trabalho de conclusão de curso (Parte II)

Atividades	**Janeiro**	**Fevereiro**	**Março**	**Abril**	**Maio**	**Junho**	**Julho**
Realizar Correções da Banca	P-E						
Implementar Aplicação		P-E	P-E	P-E	P-E		
Análise de Dados Obtidos				P	P-E	E	
Refinar Monografia						P-E	
Apresentar Trabalho							P-E

O cronograma do Trabalho de conclusão de curso (Parte II) também foi executado com sucesso, salvo alguns atrasos.

4.4 Resumo do Capítulo

Este capítulo apresentou como o trabalho foi desenvolvido. Delimitou atividades e seus prazos, bem como a classificação da pesquisa que foi realizada. Também explicitou o fluxo de atividades seguido para o desenvolvimento da aplicação, baseado em atividades ddo Srum (adaptadas).

A Figura 12 destaca que há uma macro metodologia, a qual orientou toda a condução do TCC; uma metodologia que orientou as atividades de desenvolvimento (adaptação do Scrum), e uma metodologia que orientou as atividades de análise de resultados (Pesquisa-Ação).

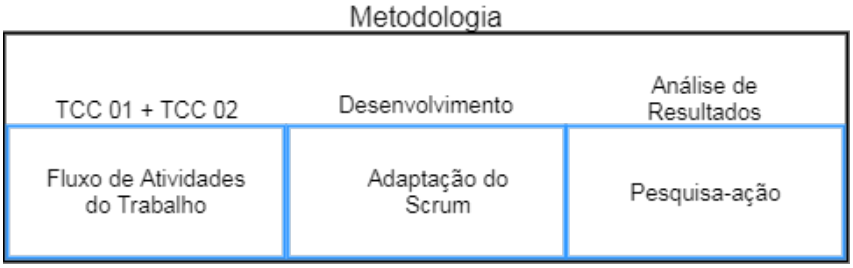


Figura 12 – Resumo das Metodologias Adotadas no Trabalho de Conclusão de Curso

5 Model IT

Este capítulo apresenta o produto de software (uma aplicação Web chamada Model IT), que foi desenvolvido neste Trabalho de Conclusão de Curso. O capítulo procura acordar detalhes quanto ao funcionamento da aplicação, com base em heurísticas transformacionais e uma arquitetura específica.

O capítulo está dividido em três seções, sendo essas: seção 5.1 - Mapeamentos Heurísticos, na qual serão exploradas as regras transformacionas adotadas: Seção 5.2 - Arquitetura Model IT, na qual será explicada a comunicação entre os componentes arquiteturais da aplicação, em três níveis de abstração; seção 5.3 - Aplicação Model IT, na qual será explorado o produto de software final desse Trabalho de Conclusão de Curso, e seção 5.4 - Resumo do Capítulo, na qual um resumo dos principais aspectos discutidos ao longo do capítulo será apresentado.

5.1 Mapeamentos Heurísticos

Nesse Trabalho de Conclusão de Curso, conforme abordado no capítulo 2, Referencial Teórico, procurou-se estudar o uso dos princípios e conceitos de Arquitetura Dirigida a Modelos (MDA). Esse *framework* arquitetural baseia-se em modelos e transformações de modelos, partindo de modelos de alto nível de abstração para modelos de baixo nível de abstração. Nesse sentido, procurando acordar um modelo em alto nível de abstração (chamado, nos referenciais teóricos de MDA, como modelo independente de computação), optou-se por um modelo na notação i*. Detalhes desse notação, foram tratados na seção 2.3, Modelagem Orientada à Meta.

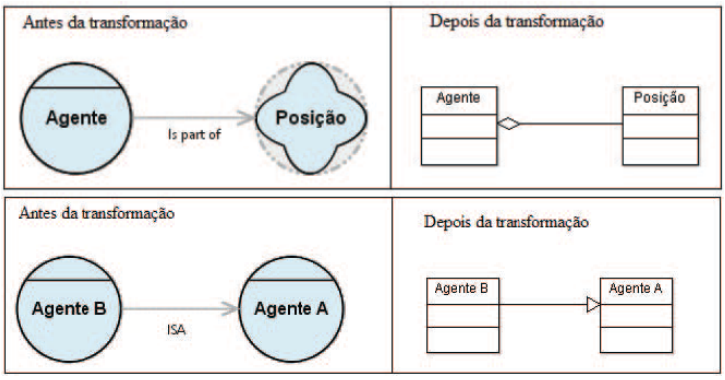


Figura 13 – Exemplo de Transformação de Modelos. Adaptado de (MELO et al., 2015).

Visando transformar os modelos na notação i* (independente da computação) para modelos mais próximos à computação (ou seja, em mais baixo nível de abstração), utilizou-

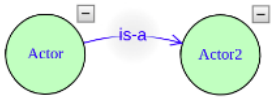
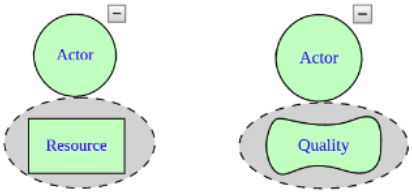
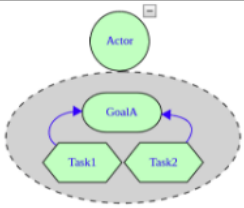
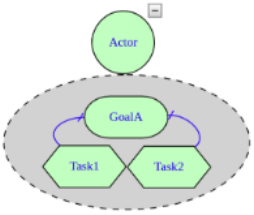
Modelagem i*	Esperado em Django
	Classe modelo, com o mesmo nome do nó.
	Duas classes modelo, sendo "Actor2" uma generalização de "Actor"
	Nos dois casos é esperado uma classe modelo (classe Actor) com um atributo dentro desta classe, com o mesmo nome da tarefa ou objetivo maleável.
	É esperado uma classe modelo, na qual possuirá dois métodos. De alguma forma, qualquer um dos métodos deve atingir o objetivo.
	É esperado uma classe modelo, na qual possuirá dois métodos. De alguma forma, apenas os dois métodos juntos atingirão o objetivo.

Figura 14 – Heurísticas Transformacionais

se como base o trabalho realizado por Melo (2015). Tal trabalho exprime a formalização de um mapeamento entre i* e diagrama de classes em UML, que é um tipo de modelo específico de plataforma (PSM). No que diz respeito à MDA, um novo modelo é gerado quando aplicado um mapeamento, e um mapeamento é gerado aplicando-se Funções de Mapeamento, como apresentado na Figura 3 da seção 2.2.2. Portanto, o trabalho citado visou fornecer um mapeamento adequado à transformação de modelos. Um exemplo pode ser observado na Figura 13.

Com base nas transformações do trabalho citado, foi proposta uma abordagem heurística específica para este trabalho, no qual, utilizando-se de transformação de modelos, um mapeamento entre modelagem intencional em i* e código Django foram gerado. Vale ressaltar que o presente trabalho propõe uma visão preliminar envolvendo MDA com modelos i* e código em DJango. Portanto, as heurísticas transformacionais focam nas principais entidades e relacionamentos do i*, sendo: ator, herança entre atores, recursos, critérios de qualidade, decomposições em *AND* e decomposições em *OR*. Tal mapeamento

pode ser encontrado na Figura 14.

5.2 Arquitetura Model IT

Neste Trabalho de Conclusão de Curso, foi desenvolvida uma aplicação *web*, a qual foi intitulada Model IT. O nome é oriundo de uma conotação da língua inglesa, o que seria mais ou menos "modele aquilo". A principal funcionalidade da aplicação é gerar um esboço de código Django a partir de modelos orientados à meta, no caso, com base na notação i*. A própria aplicação confere suporte para modelagem em i*.

A seguir, será apresentada a arquitetura do software desenvolvida para este produto. Optou-se por apresentar a arquitetura em três diferentes níveis, começando com uma visão mais generalizada da aplicação (nível 1), e conferindo visões cada vez mais específicas e detalhadas nos níveis posteriores (níveis 2 e 3).

5.2.1 Nível 1

Com base na notação SADT (LAKHOUA; ANNABI,), a Figura 15 procura ilustrar as entradas, os controles, os mecanismos e as saídas, considerando a aplicação (ou seja, o Model IT) como uma caixa preta.

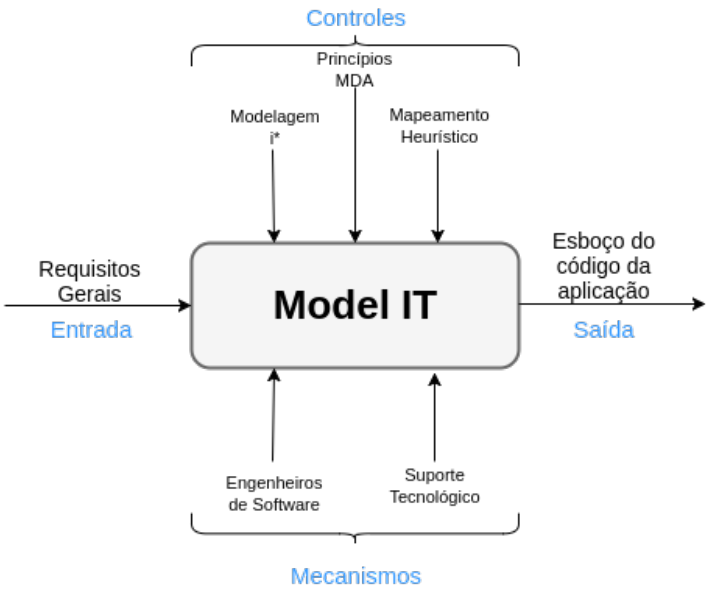


Figura 15 – Aplicação como Caixa Preta

Em termos de entradas, têm-se os requisitos gerais, elicitados pelo(s) interessado(s) em construir sua(s) aplicação(ões) concreta(s) utilizando-se do suporte Model IT. Como mecanismos, podem ser destacados todos os suportes tecnológicos mencionados no Capítulo 3 dessa monografia, os quais são insumos para viabilizar o suporte Model IT, bem

como os próprios Engenheiros de Software, os quais representam quaisquer interessados em construir suas aplicações concretas utilizando-se do suporte Model IT. Os principais controles são os modelos orientados à meta, desenhados na notação i*; os princípios de MDA, e os mapeamentos heurísticos propostos por (MELO et al., 2015). Essas heurísticas foram apresentadas anteriormente, nesse capítulo. Por fim, como saída, é gerado o esboço do código da aplicação concreta, desejado pelo(s) interessado(s).

5.2.2 Nível 2

Expandindo-se a Figura 15, é possível conhecer um pouco mais o suporte Model IT, principalmente, em termos arquiteturais. Tal detalhamento é ilustrado na Figura 16. A arquitetura do suporte Model IT aproxima-se do estilo arquitetural Cliente-Servidor. O cliente, que foi desenvolvido em Vue.js e mantido no *Web Store* Heroku, comunica-se com o servidor, desenvolvido em Node.js e mantido no *Web Store* Heroku.

A comunicação entre os componentes cliente e servidor foi realizada utilizando o protocolo HTTP, com o qual foi possível "transmitir"modelos i* do cliente para o servidor. Adicionalmente, o esboço do código é transmitido pelo caminho inverso.

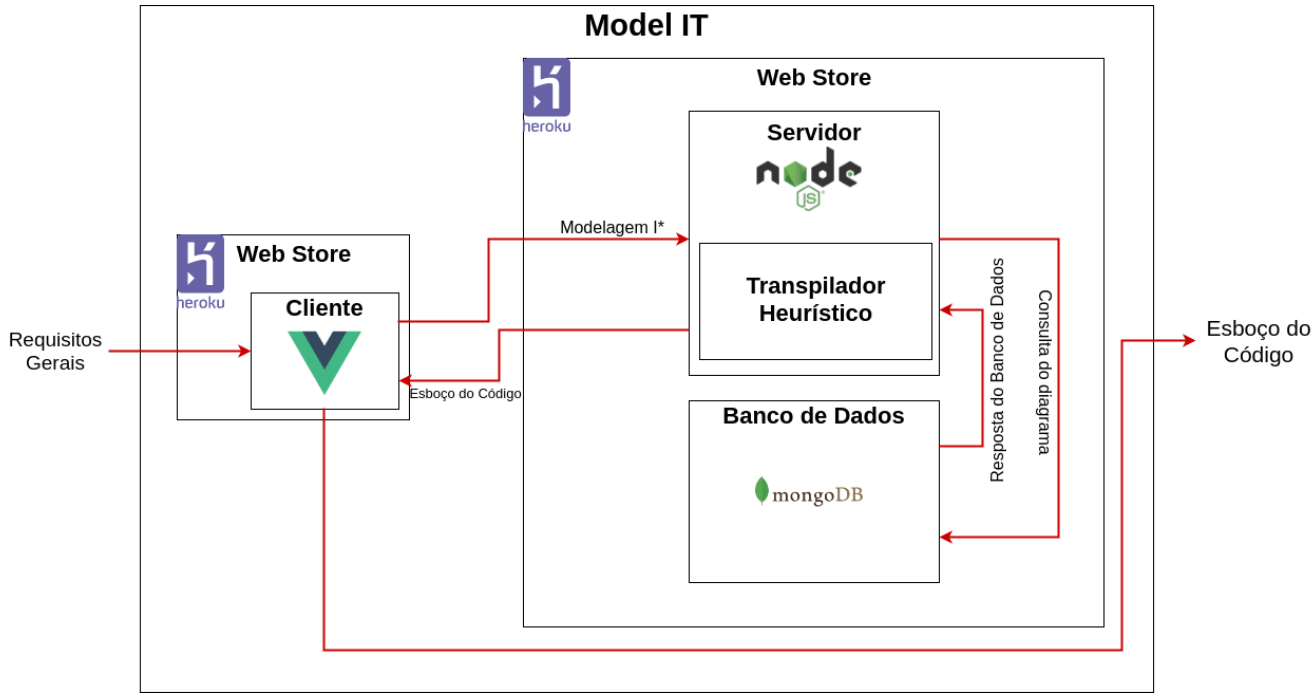


Figura 16 – Visão Arquitetural da Aplicação

5.2.3 Nível 3

Como já mencionado, o nível 3 trata-se da visão funcional da aplicação. Logo, ao expandir cada componente arquitetural da Figura 16, pode-se conhecer as principais

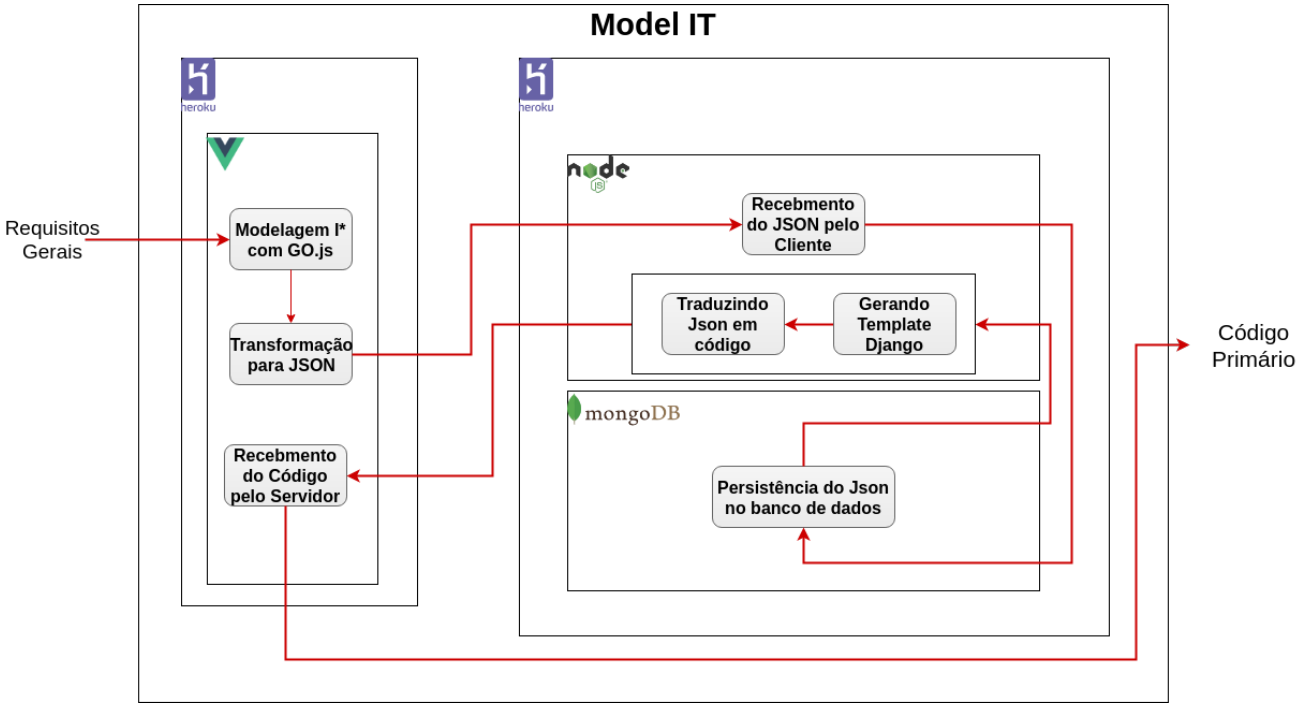


Figura 17 – Visão Funcional da Aplicação

funcionalidades desses componentes. Esse detalhamento está contemplado na Figura 17 e, logo em seguida, a explicação de cada função.

- **Cliente Vue.js:**
  - **Modelagem i* com GO.js:** O utilizador da aplicação Model IT pode modelar seus requisitos gerais utilizando a notação i*. Para isso, o Model IT conta com a biblioteca GO.js, que é capaz de criar modelagens interativas com HTML Canvas.
  - **Transformação para JSON:** Com a modelagem i*, o cliente vue.js serializa tal modelagem em JSON para que seja possível facilitar a leitura das informações presentes na modelagem por parte do servidor. Esse processo é automatizado pelo Model IT.

- **Servidor Node.js:**
  - **Recebimento do JSON pelo Cliente:** Após a transformação da modelagem em JSON, o servidor é capaz de receber tal JSON, através do protocolo HTTP, e em seguida, criar uma comparação das informações desse JSON com as heurísticas de mapeamento do (MELO et al., 2015).

– **Persistência:** Com o Json da modelagem em mãos, a primeira tarefa do servidor é de registrar essa modelagem no banco de dados, para que o cliente possa recuperá-la.

– **Gerando *template* Djano:** O módulo Transpilador do servidor é o responsável por esta tarefa. Ele cria um *template* básico e em branco de uma aplicação em Django, para que o próximo passo seja executado adequadamente.

O código do transpilador está disponível no Apêndice B.

– **Traduzindo JSON em código:** Após ter o *template* Django gerado, e com o Json da modelagem em mãos, a tradução é iniciada. Os arquivos **models.py** e **views.py** do *template* são abertos, e o código é gerado dentro destes arquivos.

5.3 Aplicação Model IT

Nessa seção, a aplicação Model IT será apresentada de forma mais detalhada, em particular considerando a utilização dos principais componentes e um exemplo de código gerado.

5.3.1 Utilização dos Componentes

Tomando como referência a Figura 18, têm-se que:

- **Item 1)** Menu de navegação para trocar de telas dentro da aplicação. "Meus Diagramas"fará uma requisição ao servidor, pedindo ao banco de dados todos os diagramas persistidos de um usuário, que será exibido na tela.
- **Item 2)** Título do diagrama apresentado. Ao clicar é possível renomear o diagrama, porém só é possível salvá-lo se não houver outro diagrama de mesmo nome em sua conta.
- **Item 3)** Menu de apoio ao diagrama. Com ele é possível salvar o diagrama, exportar para imagem(apenas o diagrama), e fazer *download* do código.
- **Item 4)** Paleta de nós. Este espaço é destinado aos nós, nos quais é possível clicar e arrastar para o item *5* para iniciar a modelagem.
- **Item 5)** Área de modelagem. Este é o espaço destinado para realizar a modelagem. Para aumentar a área, é possível minimizar a outra área (de código), simplesmente clicando no botão na extremidade da área de modelagem. Vale ressaltar que qualquer modificação nos nós presentes na área de modelagem resultará na atualização do código exibido no item **7**.

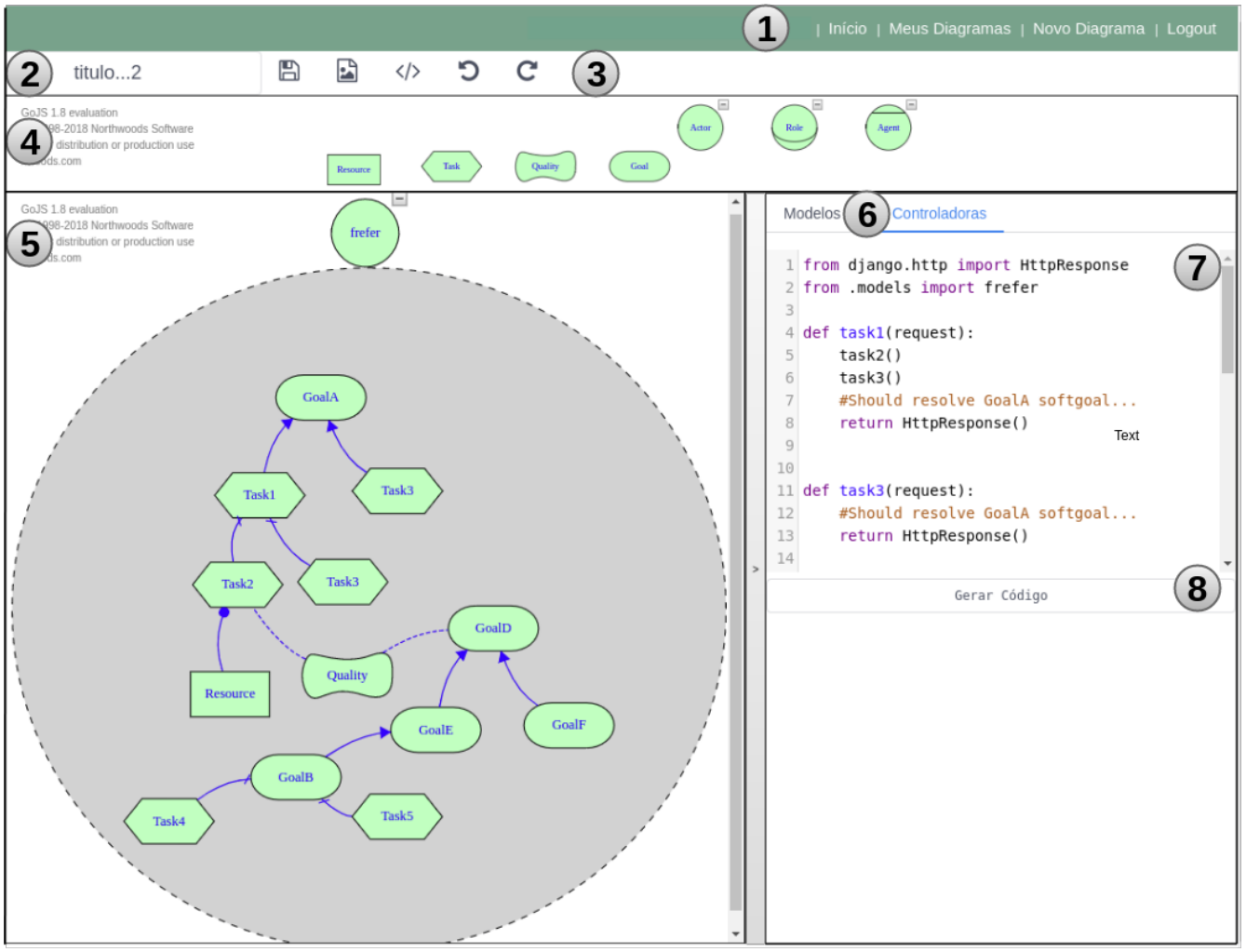


Figura 18 – Visão Funcional da Aplicação

- **Item 6)** Este é um simples menu para trocar o código exibido entre **models.py** e **views.py**. Uma vez que o Django utiliza o já citado MVT, a aba "Controladoras"representa o arquivo **views.py** do *template* Django.
- **Item 7)** Esta é a área em que o código gerado será exibido após ser gerado pelo servidor.
- **Item 8)** Este também é um lugar, no qual é possível fazer o *download* do código.

5.3.2 Exemplo de código gerado

A seguir, tem-se o código gerado pela modelagem demonstrada na Figura 18. Vale ressaltar que este código gerado está dividido em modelos e controladoras. A seguir, tem-se o código gerado para as modelos.

```
1 from django.db import models
2
```

```
3  class frefer(models.Model):
4      Resource = models.CharField(max_length=100) #coming from resource
5      Quality = models.CharField(max_length=100) #coming from quality
```

Listing 5.1 – Código das modelos

A seguir, o código gerado para as controladoras. Pode-se observar que, como já mencionado, o código gerado não se trata de um código final e sim de um código preliminar. Porém, é possível visualizar que as principais abstrações modelacionais do i*, como por exemplo ligações "AND"e "OR", foram atendidas a nível de código.

Além disso, definiu-se que todos os objetivos seriam o retorno das funções. Porém, como se trata de um código preliminar, apenas um comentário especificando o que o método deve retornar foi implementado.

```
1  from django.http import HttpResponse
2  from .models import frefer
3
4  def task1(request):
5      task2()
6      task3()
7      #Should resolve GoalA softgoal...
8      return HttpResponse()
9
10
11 def task3(request):
12     #Should resolve GoalA softgoal...
13     return HttpResponse()
14
15
16 def task4(request):
17     return HttpResponse()
18
19
20 def task5(request):
21     return HttpResponse()
22
23
24 def task2(request):
25     return HttpResponse()
26
27
28 def task3(request):
29     return HttpResponse()
30
31
32 def metataskgoalb(request):
33     task5()
```

```
34      task4()
35      #Should resolve GoalB softgoal...
36      return HttpResponse()
```

Listing 5.2 – Código gerado para as controladoras

Pode-se observar que, como já mencionado, o código gerado não se trata de um código final e sim de um código preliminar. Porém, é possível visualizar que as principais abstrações modelacionais do i*, como por exemplo ligações "AND"e "OR", foram atendidas a nível de código.

Considerando como exemplo o *GoalB* na modelagem, e tomando como base as heurísticas da Figura 14, tem-se que, de alguma forma, ambas as *tasks* devem ocorrer para que o objetivo seja atingido. Pode-se observar que foi criada uma *metatask* para englobar as duas *tasks*, e assim conseguir atingir o objetivo ao final da *metatask*.

No capítulo 6, Resultados Obtidos, será conferida uma visão mais aprofundada quanto aos códigos, expondo-se os resultados dos ciclos de pequisa-ação, com suas devidas validações do código gerado a partir da modelagem.

5.4 Resumo do Capítulo

Nesse capítulo, foi descrito o produto de software desenvolvido nesse Trabalho de Conclusão de Curso. Trata-se de uma aplicação Web, orientada pelo padrão arquitetural MVC, e que procura aplicar os princípios e os conceitos de MDA, utilizando-se de transformações de modelos e heurísticas transformacionais específicas, adaptadas de Mello (2015). Essa aplicação parte de modelos independentes de computação, no caso, especificados na notação i*, permitindo obter código em DJango.

A aplicação atende a um fluxo que condiz com o estilo arquitetural cliente-servidor em duas vias. A primeira, sendo a passagem do Json gerado (com base na modelagem) do cliente ao servidor. A segunda, sendo a passagem do código como retorno do servidor ao cliente.

Um visão geral sobre o fluxo é ilustrada na Figura 19. Vale ressaltar que a parte *core* do produto de software desenvolvido consiste nas transformações de modelos. Essa parte é representada na figura com CIM, PIM e PSM. Outros detalhes quanto as entradas, os controles, os mecanismos e as saídas foram apresentados ao longo do capítulo, na notação SADT.

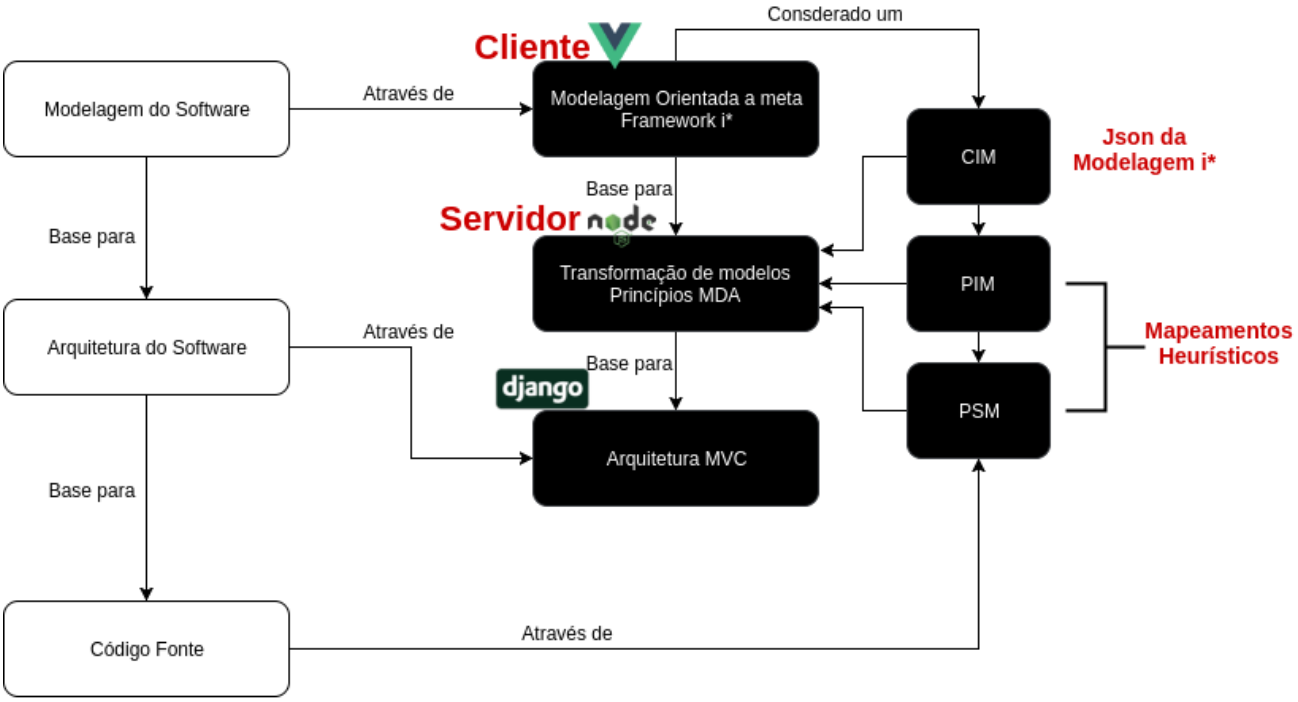


Figura 19 – Visão Funcional da Aplicação

6 Resultados Obtidos

Visando coletar as primeiras impressões da aplicação Web desenvolvida nesse Trabalho de Conclusão de Curso, foi aplicada uma modalidade de análise de resultados conhecida como Pesquisa-Ação. Detalhes dessa modalidade foram tratados na seção 4.2.2. Basicamente, foram realizados dois ciclos de Pesquisa-Ação, os quais são apresentados nesse capítulo, nas seções de 6.1 a 6.6.

6.1 Atividades da Pesquisa-Ação Realizadas

As principais atividades envolvidas na Pesquisa-Ação são Coleta de Dados, Análise e Interpretação, Elaboração do Plano de Ação e Divulgação dos Resultados, e foram desenvolvidas baseando-se no processo da Figura 20.

Em Coleta de Dados, dados da aplicação, referentes ao seu objetivo, são coletados. Tais dados servirão de insumo para a atividade seguinte. Em Análise e Interpretação, é feito um estudo utilizando os dados já coletados, para analisar se os resultados estão ou não de acordo com os objetivos propostos no início da monografia. Em Elaboração do Plano de Ação, é realizado um planejamento para, se necessário, contornar a situação desfavorável para o cumprimento dos objetivos do trabalho. Em Divulgação dos Resultados, são divulgados os resultados do cumprimento do plano de ação pré estabelecido.

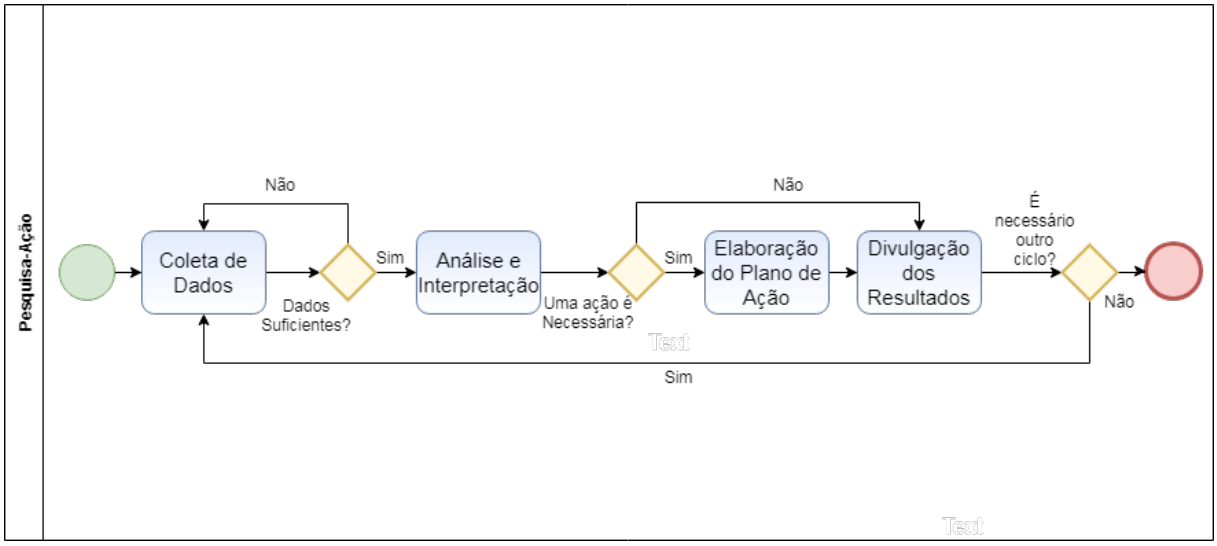


Figura 20 – Processo de Pesquisa-ação aplicado

6.2 Primeiro Ciclo

Para Coleta de Dados do primeiro ciclo, foi realizado um questionário, disponível no apêndice A, com estudantes de graduação do curso de Engenharia de Software da Universidade de Brasília. Na Tabela 4, segue um descritivo sobre o perfil desse público alvo investigado, organizado em: Período Cursado, Conhecimento, Quantidade de Participantes e Pré-Requisitos Realizados. Vale ressaltar que todos os usuários neste ciclo utilizaram a aplicação para implementar modelos de aplicações reais.

Tabela 4 – Informações do público alvo

Período Cursado	Entre o terceiro e o sexto período.
Conhecimento	Usuários já possuíam conhecimento prévio de i*
Total de Participantes	14 (Quatorze) estudantes
Pré-Requisitos Realizados	Treinamento de 30 minutos com a aplicação Model IT.

Nas próximas seções, serão descritas como as atividades principais da Pesquisa-Ação foram realizadas ao longo desse ciclo.

6.2.1 Coleta de Dados, Análise e Interpretação

Essa seção está focada nas atividades de Coleta de Dados, Análise e Interpretação. Visando apresentar como estas atividades foram realizadas ao longo do primeiro ciclo, foram consideradas as perguntas do questionário. Para cada pergunta, será apresentada uma figura ilustrando a coleta de dados, com base em um aspecto específico, o qual a pergunta procura acordar. Depois, em linguagem natural, e com base no gráfico acordado para cada pergunta, são providas análises e interpretações.

A primeira pergunta "Você considera a Arquitetura um ponto importante para a qualidade de um produto de software?", exibida como uma imagem na Figura 21, permitiu avaliar o perfil do usuário utilizador da aplicação. Como a Model IT é um produto de software desenvolvido com base em transformações heurísticas e arquitetura dirigida a modelos, é de fundamental importância que os clientes (i.e utilizadores) dessa aplicação tenham conhecimento técnico sobre a importância da Arquitetura de Software. Portanto, com 85% dos investigados reconhecendo a importância a arquitetura para a qualidade de um produto de software, acredita-se que o público alvo da aplicação foi validado para fins de coleta das primeiras impressões.

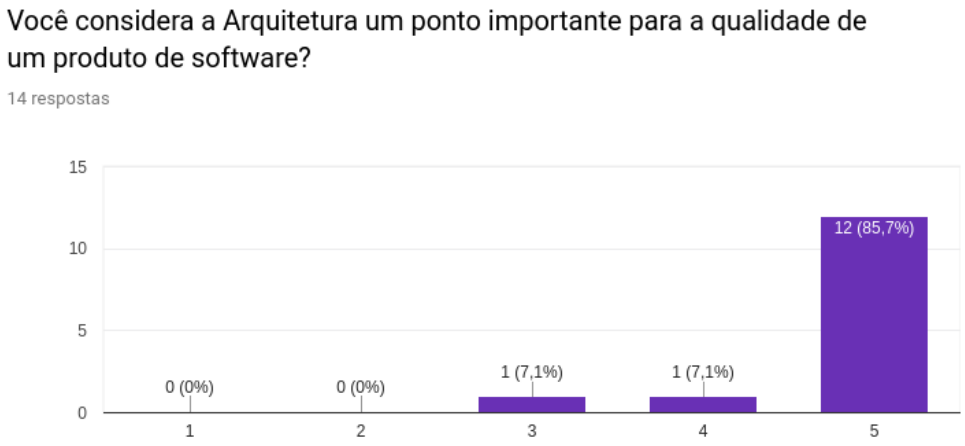


Figura 21 – Pergunta 01

A segunda pergunta "Você acha que a aplicação Model IT é útil?", exibida como uma imagem na Figura 22, permitiu avaliar o impacto da mesma em seus utilizadores. Como a aplicação possui o objetivo de auxiliar, é fundamental que seus utilizadores achem-na útil. Com mais de 70% dos utilizadores dizendo que a aplicação é extremamente útil, acredita-se que a utilidade da aplicação foi parcialmente validada, faltando ainda, a validação específica da utilidade em termos de geração de código Django.

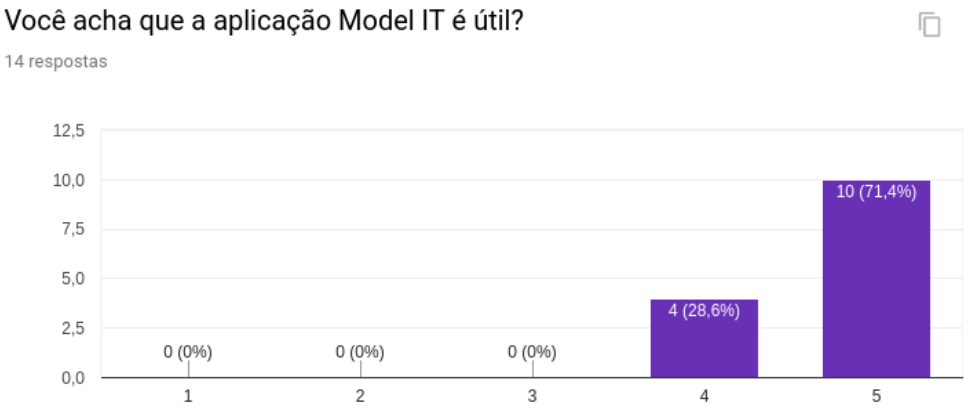


Figura 22 – Pergunta 02

A terceira pergunta "Visto que o objetivo da aplicação é a geração parcial de código Django através de modelos i*, você acha que a aplicação cumpre com o proposto?", exibida como uma imagem na Figura 23, é a pergunta mais importante deste ciclo de pesquisa-ação, pois ela valida grande parte dos objetivos propostos neste trabalho. Como o objetivo principal da aplicação é a geração de código Django, fica clara a importância desta pergunta. Com cerca de 64% dos votos máximos, acredita-se que o objetivo geral do presente trabalho foi alcançado com sucesso.

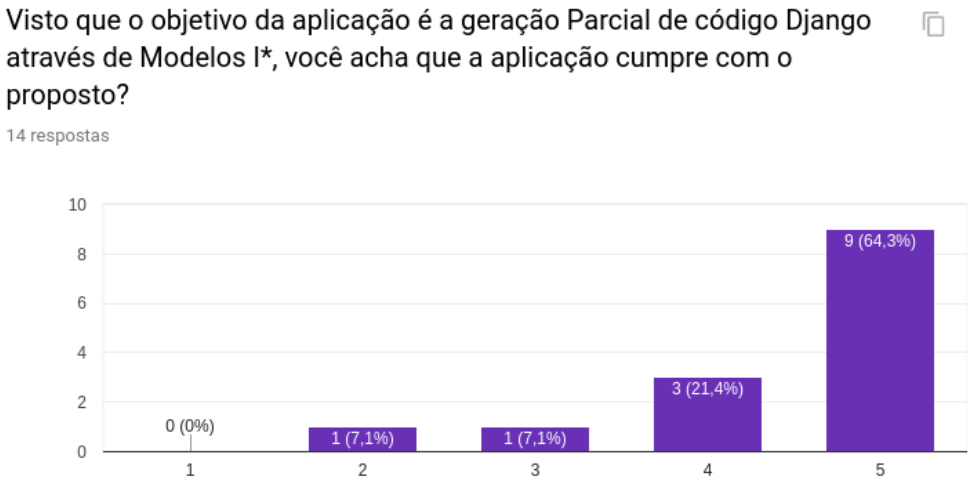


Figura 23 – Pergunta 03

A quarta pergunta "Você acha que a aplicação consegue lhe poupar tempo, visto que gera uma porção de código?", exibida como uma imagem na Figura 24, permitiu avaliar a confiabilidade da pergunta anterior, pois não há como a aplicação cumprir com seu propósito, caso a mesma não consiga poupar tempo dos desenvolvedores. Portanto, é fundamental que os utilizadores consigam afirmar que a aplicação lhes poupa tempo. Com 70% dos investigados reconhecendo que há tempo sendo poupado com sua utilização, acredita-se que a aplicação teve êxito em seu propósito.

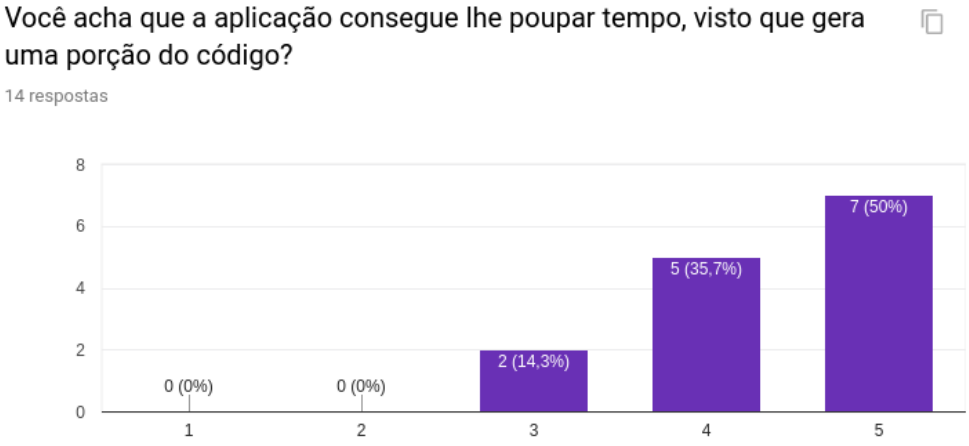


Figura 24 – Pergunta 04

A quinta pergunta "Você faria o uso da Aplicação Model IT em seus projetos?", exibida como imagem na Figura 25, serviu para averiguar o impacto da utilização da aplicação, nos seus utilizadores. Uma vez que o utilizador adota a aplicação para uso pessoal em seus projetos, pode-se afirmar um resultado positivo da aplicação no que tange ao objetivo de prover apoio ao desenvolvedor na construção de aplicações web, utilizando o *framework* Django e orientando-se pelo padrão arquitetural MVC. Com mais de 92% dos votos, conclui-se que a aplicação foi bem sucedida nesse aspecto.

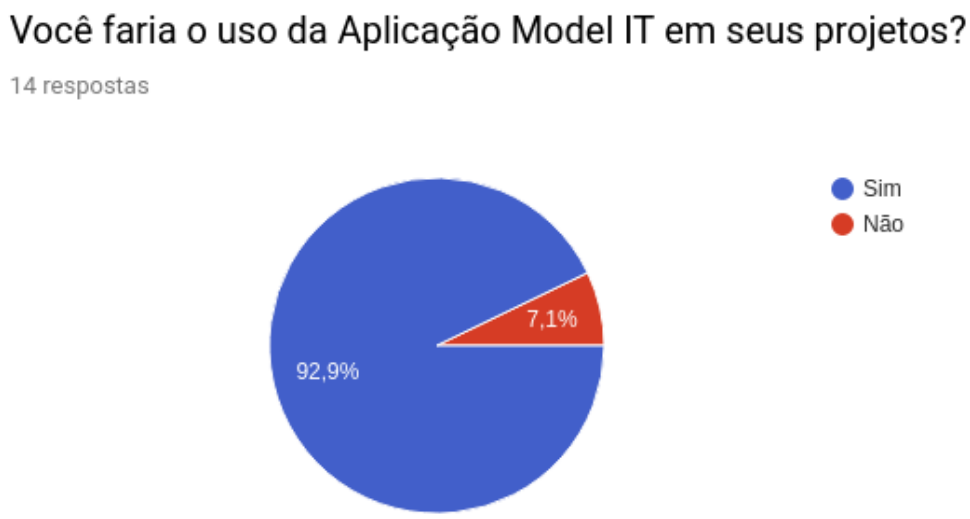


Figura 25 – Pergunta 05

Ao final das perguntas citadas anteriormente, os utilizadores tinham a possibilidade de expor críticas, sugestões ou mesmo erros referentes à aplicação. Todos os 14 usuários exprimiram suas opiniões, e tais informações estão expostas a seguir, utilizando suas próprias palavras.

1. "A área para composição do modelo é muito pequena. As setas bugam e é difícil movimentar todo o modelo de uma vez."
2. "Adicionar fronteiras"
3. "Implementar ctrl+z"
4. "Colocar legenda para as ferramentas e links. Seria interessante também colocar uma aba de informações ou um tutorial para explicar modelagem i* para que não lembra ou não sabe."
5. "Colocar algo útil na página principal, para me salvar um clique. Melhorar e aumentar o header. Versionamento dos diagramas."
6. "Mais de uma pessoa editar um arquivo. Exportar diagrama para imagem. AUMENTAR O BOTÃO PARA VER O CÓDIGO."
7. "Muito bom. Seria interessante também subir um código e gerar o diagrama. Editar diagrama de forma colaborativa. Gerar controle dos objetos por camadas (layers). Gerar imagem .png. Salvar versões do diagrama. Criar modo de apresentação."
8. "Algumas melhorias de usabilidade no geral. Auto-save para não perder os diagramas. E uma breve descrição de cada "ferramenta"para eu saber o que elas fazem e como se relacionam. Fiquei confuso quando fui pegar uma "entidade"e ela se ligou com as outras na barra de ferramenta."

9. "Alguns pontos relacionados à usabilidade como: tamanho dos botões que poderiam ser maiores, estarem mais em evidência. Setas que permitem as ligações entre goals, tasks, resources, poderiam possuir legendas para que o usuário possa selecionar e utilizar a mais adequada para cada ligação. O tamanho para a escrita de caracteres dentro de cada componente, às vezes é ultrapassado e não comporta o tamanho da escrita."

10. "varios usuários mexendo em um mesmo diagrama ao mesmo tempo, exportar para arquivo ou imagem"

11. "poderia ter um tutorial"

12. "Está muito bom"

13. "Nada"

14. "Usabilidade"

6.2.2 Divulgação do Plano de Ação

Com a análise disposta na subseção anterior, foi possível notar que algumas ações precisavam ser tomadas para corrigir defeitos e prover melhorias propostas pelos usuários. Porém, vale ressaltar que, prioritariamente, não será abordado o tópico de usabilidade, critério de qualidade esse que correspondeu a grande parte das observações acordadas pelos utilizadores. A não priorização dessa questão baseia-se no fato desse Trabalho de Conclusão de Curso propor desenvolver uma primeira versão dessa aplicação *online*, a qual envolve outras preocupações - *a priori* - mais fundamentais para validar a ideia, tais como:

(i) Ter um ambiente, necessariamente interno à aplicação, que permita ao utilizador elaborar modelos na notação i*, sejam esses SDs ou SRs, sem depender de suportes de terceiros; e

(ii) Possibilitar a geração semi-automatizada de código, partindo de transformações de modelos, orientando-se por heurísticas e pelo padrão arquitetural MVC. Ambos, heurísticas e arquitetura MVC, vistos, complementarmente, como boas práticas para as problemáticas envolvidas nesse contexto.

O plano de ação feito para contornar tais itens foi, prioritariamente, focado em melhorias que impactavam mais diretamente os desafios chave para geração da primeira versão da aplicação Model IT. Dada que a evolução de um software é uma atividade constante ao longo do ciclo de vida do mesmo, adicionalmente, ajustou-se o plano de ação de acordo com o *slot* de tempo disponível para a atividade de Análise de Resultados, sem

prejuízo às demais atividades intrínsecas ao desenvolvimento e fechamento do Trabalho de Conclusão de Curso.

Diante do exposto, os itens apresentados na Tabela 5, foram priorizados e realizados.

Tabela 5 – Plano de ação do primeiro ciclo

Item	Plano de Ação
1	Considerado como trabalho futuro.
2	Implementar melhoria.
3	Implementar melhoria.
4	Considerado como trabalho futuro.
5	Considerado como trabalho futuro.
6	Implementar melhoria.
7	Considerado como trabalho futuro.
8	Considerado como trabalho futuro.
9	Considerado como trabalho futuro.
10	Considerado como trabalho futuro.
11	Considerado como trabalho futuro.
12	Não se enquadra.
13	Não se enquadra.
14	Não se enquadra.

A implementação das melhorias propostas foi realizada em uma *Sprint*, afim de finalizar tais demandas e iniciar um novo ciclo de pesquisa-ação.

6.2.3 Divulgação dos Resultados

Com a implementação proposta no plano de ação, apenas modificações de usabilidade foram feitas, o que não impacta nos objetivos propostos neste trabalho. Portanto, os resultados referentes aos objetivos da aplicação permaneceram como já mencionados.

A Figura 26 mostra a aplicação em seu estado anterior à ação proposta. Após as melhorias propostas, a aplicação se tornou àquela apresentada no capítulo anterior, na Figura 18.

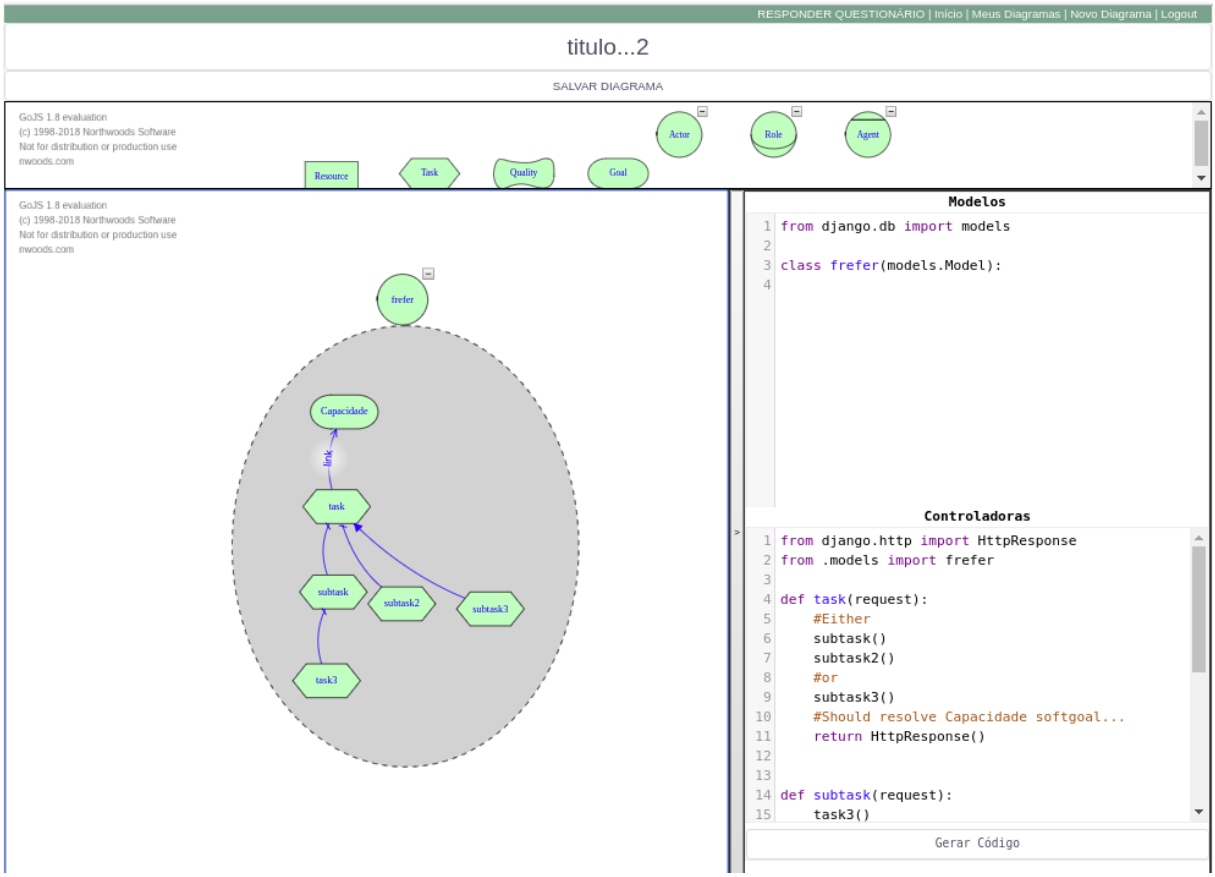


Figura 26 – Versão anterior da aplicação

6.3 Segundo Ciclo

Com base no primeiro ciclo de pesquisa-ação, foi possível validar os objetivos do trabalho como, por exemplo, confirmar a pertinência da aplicação Model IT no apoio aos desenvolvedores que se enquadram no perfil de usuários da aplicação. Entretanto, dado que os usuários focaram suas observações de melhorias focadas no critério de qualidade usabilidade, não sendo esse o principal foco de atenção nessa primeira versão da aplicação Model IT, por razões já mencionadas anteriormente, optou-se por uma nova abordagem de análise de resultados para o segundo ciclo de pesquisa-ação. A intenção foi, principalmente, permitir avaliações e evoluções em aspectos vistos como mais relevantes para essa primeira versão da aplicação Model IT, com destaque para a avaliação das heurísticas transformacionais bem como para o cumprimento do padrão arquitetural MVC.

Para a condução deste ciclo, o público alvo que contribuiu para a condução da pesquisa foram: (i) dois especialistas em i*; e (ii) o autor com papel de desempenhar uma Observação Participativa (WHYTE, 2005).

6.3.1 Coleta de Dados e Análise

Para o segundo ciclo, optou-se por fazer uma análise das heurísticas, levando em consideração o mapeamento disponibilizado na Figura 14. Esta análise deu-se implementando o modelo proposto e analisando o código gerado.

6.3.2 Heurística 1

A heurística 1, prevê, em código Django, uma classe de mesmo nome do nó modelado. Tal heurística teve sua validação, tanto pelos dois especialistas quanto pelo autor. O código da modelo pode ser verificado em Listing 6.1.

```
1 from django.db import models
2
3 class Actor(models.Model):
```

Listing 6.1 – Código da Heurística 1

6.3.3 Heurística 2

A heurística 2 prevê, em código Django, duas classes com os mesmo nomes do nó. Porém, há uma relação *is-a* entre os nós, seria esperado uma herança. Tal heurística não foi atingida nesse primeiro momento, como mostra o código das modelos em Listing 6.2.

```
1 from django.db import models
2
3 class Actor(models.Model):
4
5 class Actor2(models.Model):
```

Listing 6.2 – Código da Heurística 2

6.3.4 Heurística 3

A heurística 3 prevê, em código Django, além da classe com mesmo nome do nó, que tanto *Resource* quanto *Quality* virem atributos desta classe se, e somente se, estiverem dentro de sua fronteira. Tal Heurística não foi plenamente concluída, pois houve falhas na geração do atributo de *Resource*. Já o atributo de *Quality* foi gerado com sucesso, como mostra o código em Listing 6.3 e Listing 6.4.

```
1 from django.db import models
2
3 class Actor(models.Model):
```

Listing 6.3 – Código das modelos da Heurística 3

```
1 from django.db import models
2
3 class Actor(models.Model):
4     Quality = models.CharField(max_length=100)
```

Listing 6.4 – Código das modelos da heurística 3

6.3.5 Heurística 4

A heurística 4 prevê, em código Django, dois métodos que, de alguma forma, qualquer um deles cumpra com o objetivo. Tal heurística foi não foi cumprida pois, apesar de os métodos terem sido gerados, não há como definir tais métodos, sendo assim, não há como saber quais métodos deveriam cumprir o objetivo, conforme ilustrado em Listing 6.5.

```
1 from django.http import HttpResponse
2 from .models import Actor
3
4 def task1(request):
5     return HttpResponse()
6
7 def task2(request):
8     return HttpResponse()
```

Listing 6.5 – Código das controladoras da heurística 4

6.3.6 Heurística 5

A heurística 5 prevê, em código Django, dois métodos que, de alguma forma, apenas a execução de ambos cumpra com o objetivo. Tal heurística não foi cumprida plenamente, pois, apesar de os métodos terem sido gerados, não há como definir tais métodos, sendo assim, não há como garantir que os dois métodos sejam executados, conforme ilustrado em Listing 6.6.

```
1 from django.http import HttpResponse
2 from .models import Actor
3
4 def task1(request):
5     return HttpResponse()
6
7 def task2(request):
8     return HttpResponse()
```

Listing 6.6 – Código das controladoras da heurística 5

6.3.7 Divulgação do Plano de Ação

O plano de ação gerado pela análise dos dados obtidos está disponível na Tabela 6.

Tabela 6 – Plano de Ação ciclo dois

Heurística	**Ação**
1	Nenhuma. Heurística Validada
2	Gerar herança entre classes
3	Implementar atributo Resource
4	Implementar melhoria no 'OR' para afirmar objetivo
5	Implementar melhoria no 'AND' para afirmar objetivo

Tal plano de ação foi proposto e desenvolvido em uma *Sprint*, para que o ciclo três da pesquisa-ação ocorresse.

6.3.8 Divulgação dos Resultados

Como a heurística 1 foi validada, ela não mais será avaliada nos ciclos, tomando-a como analisada e confirmada tanto pelos especialistas quanto pelo autor.

Para fins de melhor condução deste capítulo, a divulgação dos três foram tratadas em conjunto, e e ambas estão presentes no ciclo quatro.

6.4 Terceiro Ciclo

Ainda seguindo o proposto no segundo ciclo da pesquisa-ação, o público alvo que contribuiu para a condução da pesquisa foram: (i) dois especialistas em i*; e (ii) o autor com papel de desempenhar uma observação participativa (WHYTE, 2005).

6.4.1 Coleta de Dados e Análise

Como já foi validada a heurística 1, neste ciclo foram analisadas apenas as heurísticas 2,3,4 e 5.

6.4.2 Heurística 2

A heurística 2, como já mencionado, prevê, em código Django, duas classes com os mesmos nomes do nó. Tal heurística foi validada, conforme mostra o código em Listing 6.7.

```
1 from django.db import models
2
3 class Actor(Actor2):
```

```
4
5 class Actor2(models.Model):
```

Listing 6.7 – Código da Heurística 2

6.4.3 Heurística 3

A heurística 3, como já mencionado, prevê, em código Django, além da classe com mesmo nome do nó, que tanto *Resource* quanto *Quality* virem atributos desta classe se, e somente se, estiverem dentro de sua fronteira. Tal heurística foi validada junto aos especialistas e com o autor. O resultado pode ser encontrado no código em Listing 6.8 e Listing 6.9.

```
1 from django.db import models
2
3 class Actor(models.Model):
4     Resource = models.CharField(max_length=100) #coming from resource
```

Listing 6.8 – Código da Heurística 3

ł

```
1 from django.db import models
2
3 class Actor(models.Model):
4     Quality = models.CharField(max_length=100) #coming from resource
```

Listing 6.9 – Código da Heurística 3

6.4.4 Heurística 4

A heurística 4, como já mencionado, prevê, em código Django, dois métodos que, de alguma forma, qualquer um deles cumpra com o objetivo. Neste caso, pode-se observar que, para abranger a questão do Objetivo, o mesmo foi tratado como o retorno de uma função. Como não há a possibilidade da geração total do código, uma vez que o escopo deste trabalho é apenas a geração de um template de código do mesmo, o objetivo foi tratado como um comentário ao final do corpo do método. Tanto o método *task1* quanto o método *task2* retornam este objetivo em seus métodos, cumprindo, assim, a heurística, conforme apresentado em Listing 6.10. ł

```
1 from django.http import HttpResponse
2 from .models import Actor
3
4 def task1(request):
5     #Should resolve GoalA softgoal...
6     return HttpResponse()
```

```
7
8  def task2(request):
9      #Should resolve GoalA softgoal...
10     return HttpResponse()
```

Listing 6.10 – Código da Heurística 4

6.4.5 Heurística 5

A heurística 5, como já mencionado, prevê, em código Django, dois métodos que, de alguma forma, apenas a execução de ambos cumpra com o objetivo. Contudo, a heurística continua sendo não cumprida, pois ao final da *Sprint* do ciclo três, não foi possível finalizar a sua implementação, conforme apresentado em Listing 6.11. ł

```
1  from django.http import HttpResponse
2  from .models import Actor
3
4  def task1(request):
5      return HttpResponse()
6
7  def task2(request):
8      return HttpResponse()
```

Listing 6.11 – Código da Heurística 5

6.4.6 Divulgação do Plano de Ação

O plano de ação gerado pela análise dos dados obtidos está disponível na Tabela 7.

Tabela 7 – Plano de Ação ciclo três

Heurística	**Ação**
1	Nenhuma. Heurística Validada no segundo ciclo
2	Nenhuma. Heurística Validada no terceiro ciclo
3	Nenhuma. Heurística Validada no terceiro ciclo
4	Nenhuma. Heurística Validada no terceiro ciclo
5	Implementar melhoria no 'AND' para afirmar objetivo

Tal plano de ação foi proposto e desenvolvido em uma *Sprint*, para que o ciclo quatro da pesquisa-ação ocorresse.

6.5 Quarto Ciclo

Ainda seguindo o proposto do terceiro ciclo da pesquisa-ação, o público alvo que contribuiu para a condução da pesquisa foram: (i) dois especialistas em i*; e (ii) o autor com papel de desempenhar uma observação participativa (WHYTE, 2005).

6.5.1 Coleta de Dados e Análise

Como as heurísticas 1,2,3 e 4 já foram validadas, apenas a heurística 5 foi analisada neste ciclo.

6.5.2 Heurística 5

A heurística 5, como já mencionado, prevê, em código Django, dois métodos que, de alguma forma, apenas a execução de ambos cumpra com o objetivo. Neste caso, como o objetivo foi tratado como um comentário de um retorno de um método, e como ambos os métodos *task1* e *task2* precisam ser realizados para atingir o objetivo, foi criada uma *metatask* para englobar os outros dois métodos, e assim retornar o objetivo após a realização obrigatória dos outros dois métodos.

Com isso, pode-se analisar a heurística e validar o código gerado, conforme apresentado em Listing 6.12. ł

```
1  from django.http import HttpResponse
2  from .models import Actor
3
4  def task1(request):
5      return HttpResponse()
6
7  def task2(request):
8      return HttpResponse()
9
10 def metataskgoala(request):
11     task1()
12     task2()
13     #Should resolve GoalA softgoal...
14     return HttpResponse()
```

Listing 6.12 – Código da Heurística 5

6.5.3 Divulgação do Plano de Ação

Conforme apresentado, e diante do fato das heurísticas terem sido cumpridas em sua totalidade, não houve a necessidade de um novo plano de ação.

6.5.4 Divulgação dos Resultados

Conforme mencionado anteriormente, nesta subseção será feita a divulgação dos resultados finais obtidos após os ciclos de pesquisa-ação realizados. Para tal, será demonstrado um fluxo de execução dentro da aplicação.

A aplicação Model IT está disponível para ser utilizada no seguinte endereço: modelit.herokuapp.com/

Seu código, também está disponível, através da licença GPL, e está dividido entre cliente e servidor.

O código da aplicação cliente se encontra em: <https://github.com/rafaelakiyoshi/modelit-front> Acesso em: 28 de Junho de 2018.

O código da aplicação servidor se encontra em: <https://github.com/rafaelakiyoshi/modelit-api> Acesso em: 28 de Junho de 2018.

Ao abrir a aplicação, a página da Figura 27 irá aparecer. Nesta página, pode-se realizar o *login*, caso já possua uma conta. Caso contrário, pode-se criar uma conta para iniciar o uso da aplicação. É necessário criar uma conta, pois os diagramas ficarão salvos nessa conta, para futuros acessos.

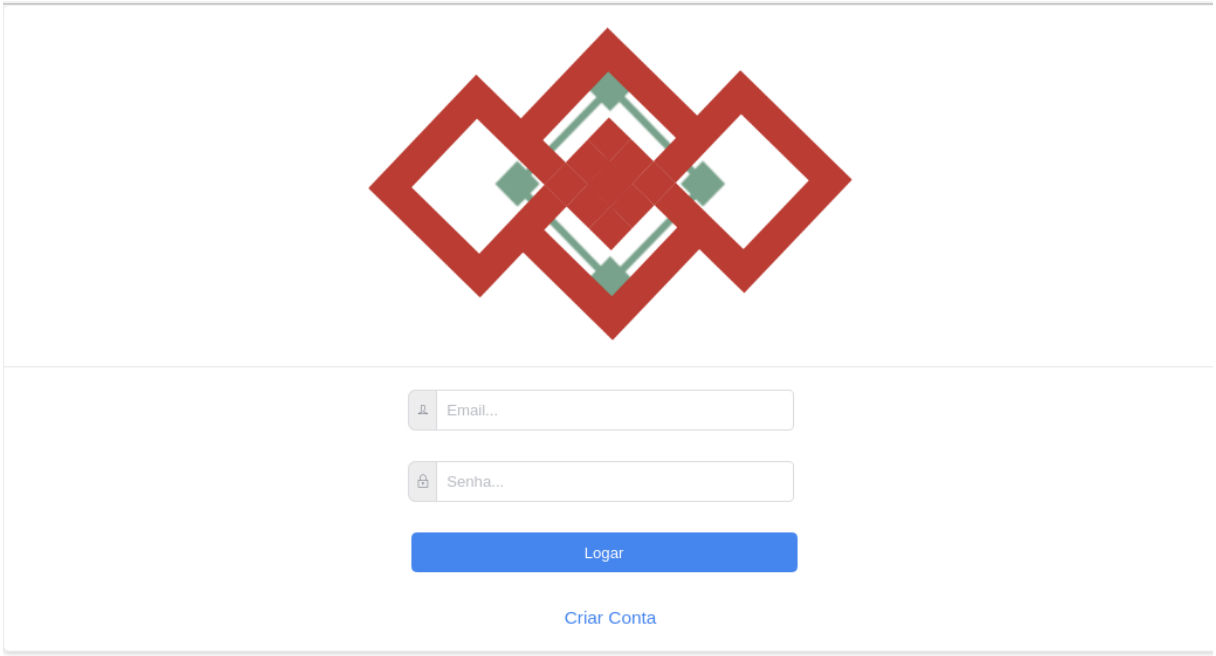


Figura 27 – Página inicial da Aplicação.

Após efetuado o *login*, a página que será exibida será a Figura 28. Nela, pode-se observar um *menu* no canto superior direito, no qual, como já mencionado no Capítulo 5, serve para navegar por toda a aplicação.

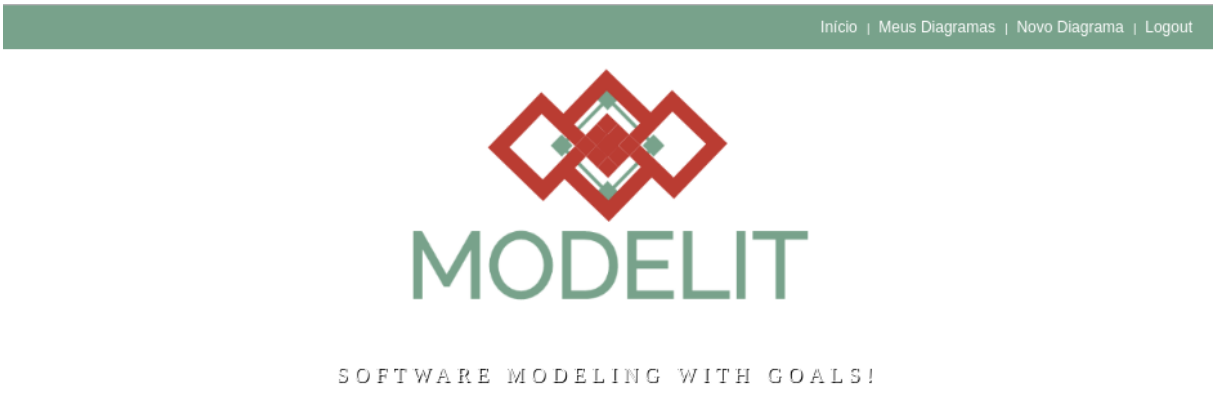


Figura 28 – *Dashboard* da Aplicação

Clicando em "Novo Diagrama", a página que será exibida é a da Figura 29. Como foi mostrado na Figura 18 do capítulo 5, esta é a página principal da aplicação. É nela que ocorre tanto a modelagem, do lado esquerdo, quanto a geração do código, do lado direito.

Após realizada a modelagem, o código será exibido no canto direito da tela. Tal código é dividido entre Modelos e Controladoras, como já mencionado no capítulo 5. O código pode ser baixado, clicando em "Baixar código", e um arquivo *zip*, chamado *archive*, será baixado, como mostra a Figura 30.

Ao abrir este arquivo *zip*, pode-se notar que o mesmo código gerado na aplicação foi, de fato, escrito nos arquivos *models.py*, que seriam as modelos do Django, e *views.py*, que seriam as controladoras. As Figuras 31 e 32 exibem essas colocações.

Dentro da aplicação, ao acessar "Meus Diagramas", é possível notar os diagramas salvos daquela conta específica. Tal diagrama pode ser acessado, o que retornará uma instância do diagrama selecionado na tela da Figura 29.

Vale ressaltar também, que os diagramas salvos deixam rastros, o que facilita o versionamento de diagramas. Tais colocações podem ser conferidas na Figura 33.

6.6 Resumo do Capítulo

Neste capítulo, foram apresentados os resultados obtidos ao longo desse Trabalho de Conclusão de Curso, através dos ciclos de pesquisa-ação realizados. Foram realizados

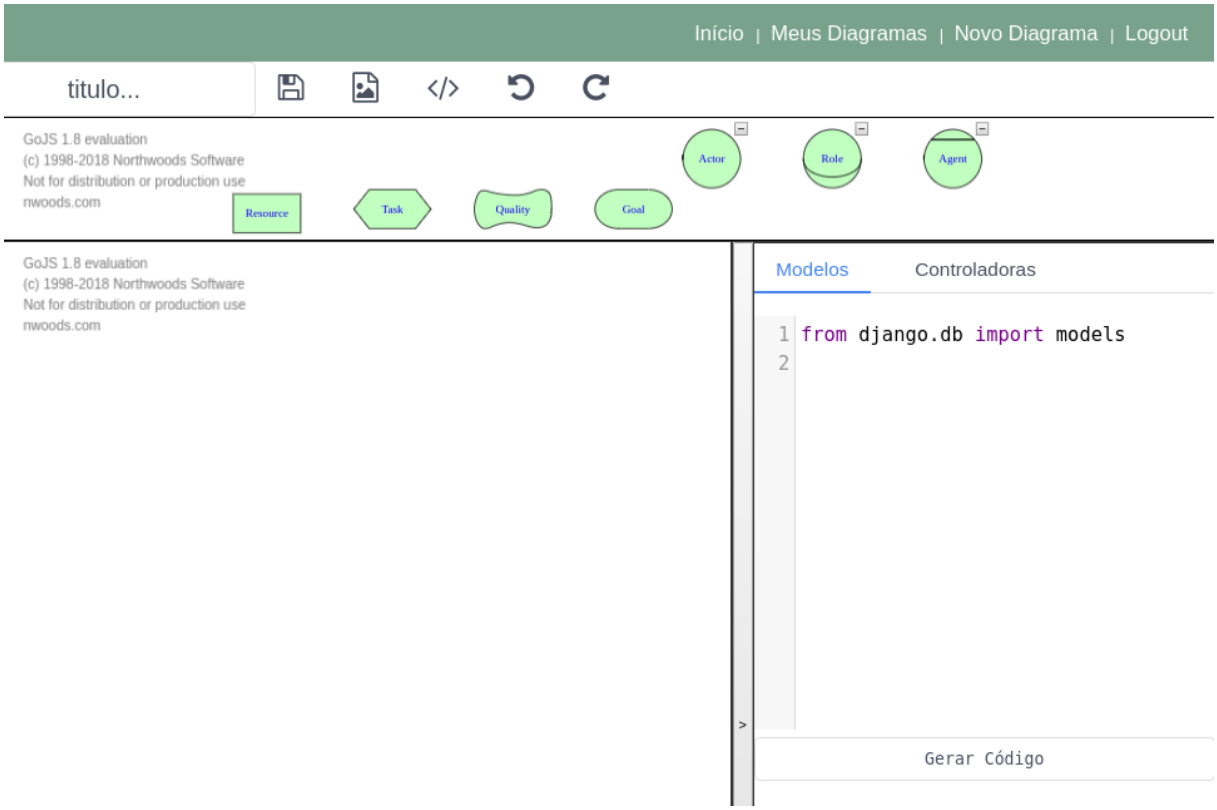


Figura 29 – Página de Modelagem da Aplicação

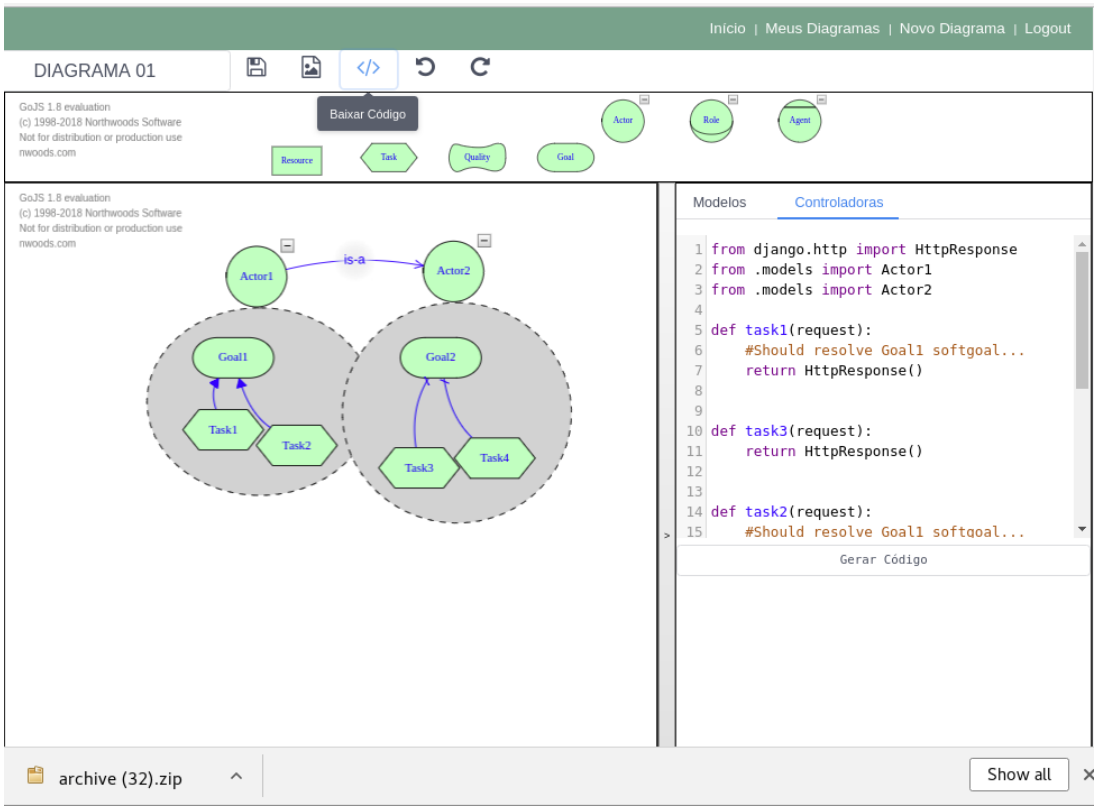


Figura 30 – Página de Modelagem com Código Baixado

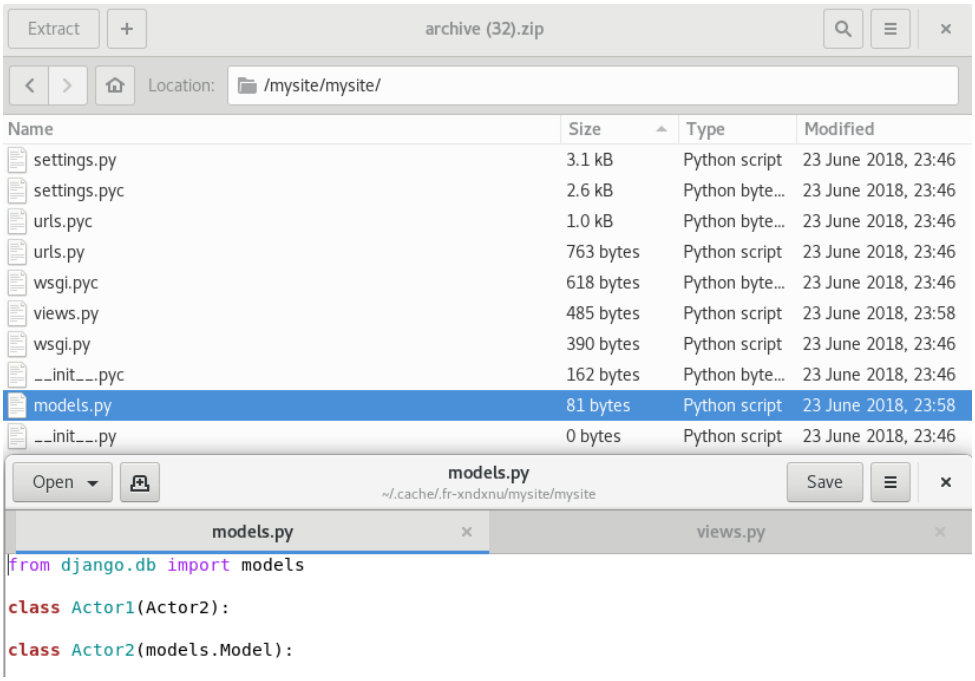


Figura 31 – Código das modelos Baixado

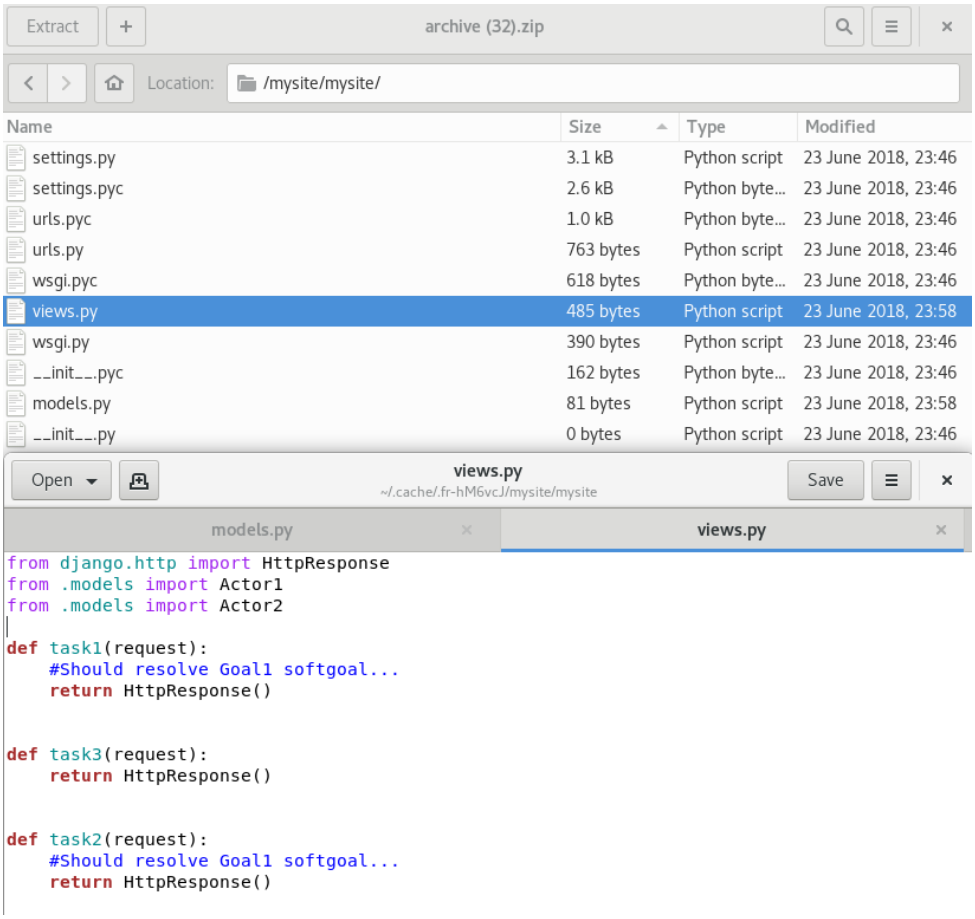


Figura 32 – Código das controladoras Baixado

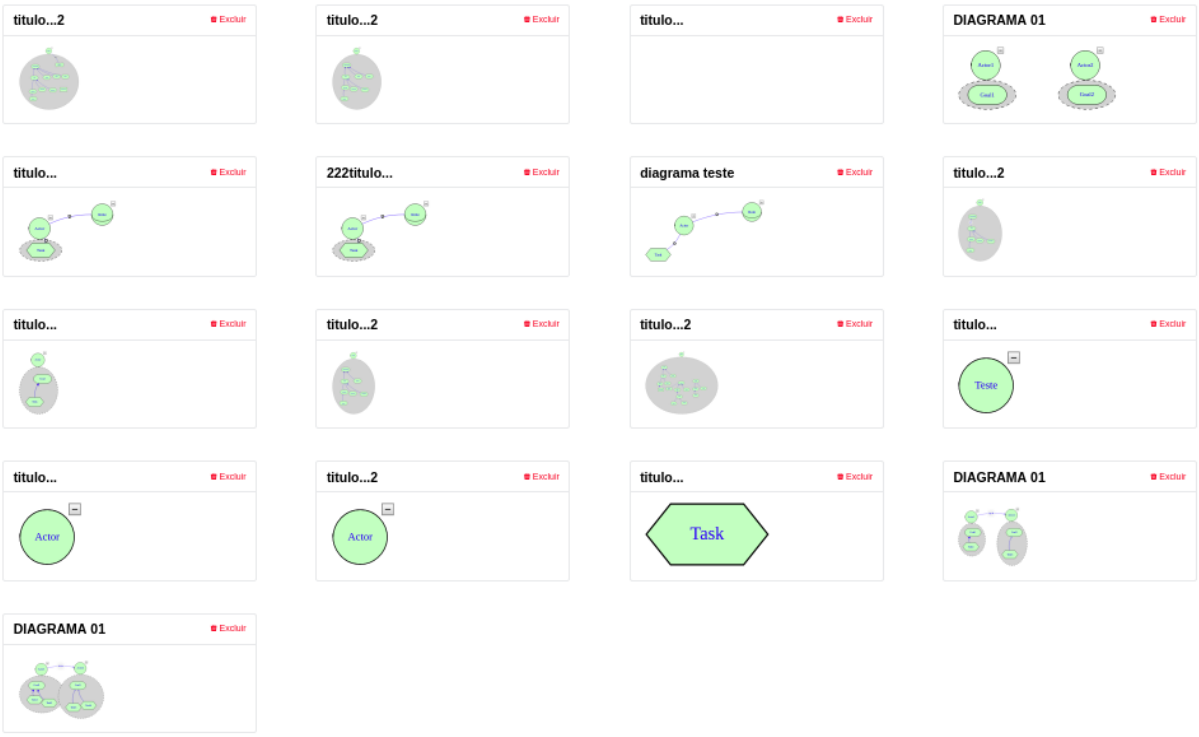


Figura 33 – Página de Meus Diagramas

um total de quatro ciclos de pesquisa-ação, porém nem todos tiveram a necessidade da realização de todas as atividades previstas e propostas no capítulo 4.

O primeiro ciclo, contou com quatorze estudantes de Engenharia de Software, entre o terceiro e o sexto período, como público alvo. Após algumas utilizações, os estudantes responderam a um questionário, afim de ajudar na avaliação da aplicação referente aos seus objetivos propostos. Tal avaliação serviu para validar os objetivos propostos, porém, não serviu para a validação das heurísticas propostas, uma vez que os estudantes não dominavam o *framework* utilizado, o i*. Por isso, foi proposta uma nova abordagem para os ciclos dois, três e quatro.

O segundo ciclo, contou com dois especialistas em i*, e o autor com observação participativa como público alvo. Tal ciclo teve como propósito validar as heurísticas transformacionais propostas para este trabalho, utilizando como referência o trabalho de Mello 2015. Este ciclo concluiu a validação apenas da primeira heurística, fazendo necessário um plano de ação para corrigir as heurísticas para o próximo ciclo.

O terceiro ciclo, seguiu a mesma abordagem do segundo ciclo, utilizando o mesmo público alvo. Devido ao fato da primeira heurística já ter sido validada, apenas as heurísticas 2,3,4 e 5 foram avaliadas neste ciclo. Concluiu-se que as heurísticas 2,3 e 4 foram validadas tanto pelos especialistas quanto pelo autor. Todavia, um plano de ação se fez necessário para a heurística 5.

O quarto ciclo seguiu a mesma abordagem do segundo ciclo, utilizando o mesmo público alvo. Devido ao fato de apenas a heurística 5 não ter sido validada, apenas esta foi considerada item de avaliação. Concluiu-se que a heurística 5 foi validada tanto pelos especialistas quanto pelo autor. Como nenhuma heurística deixou de ser cumprida, não houve um plano de ação.

A Figura 34 resume os principais aspectos tratados ao longo desse capítulo, sendo os quadros em verdes, os pontos validados; os quadros em vermelho, os pontos feridos; os quadros amarelos, o público alvo; e os quadros brancos, as atividades.

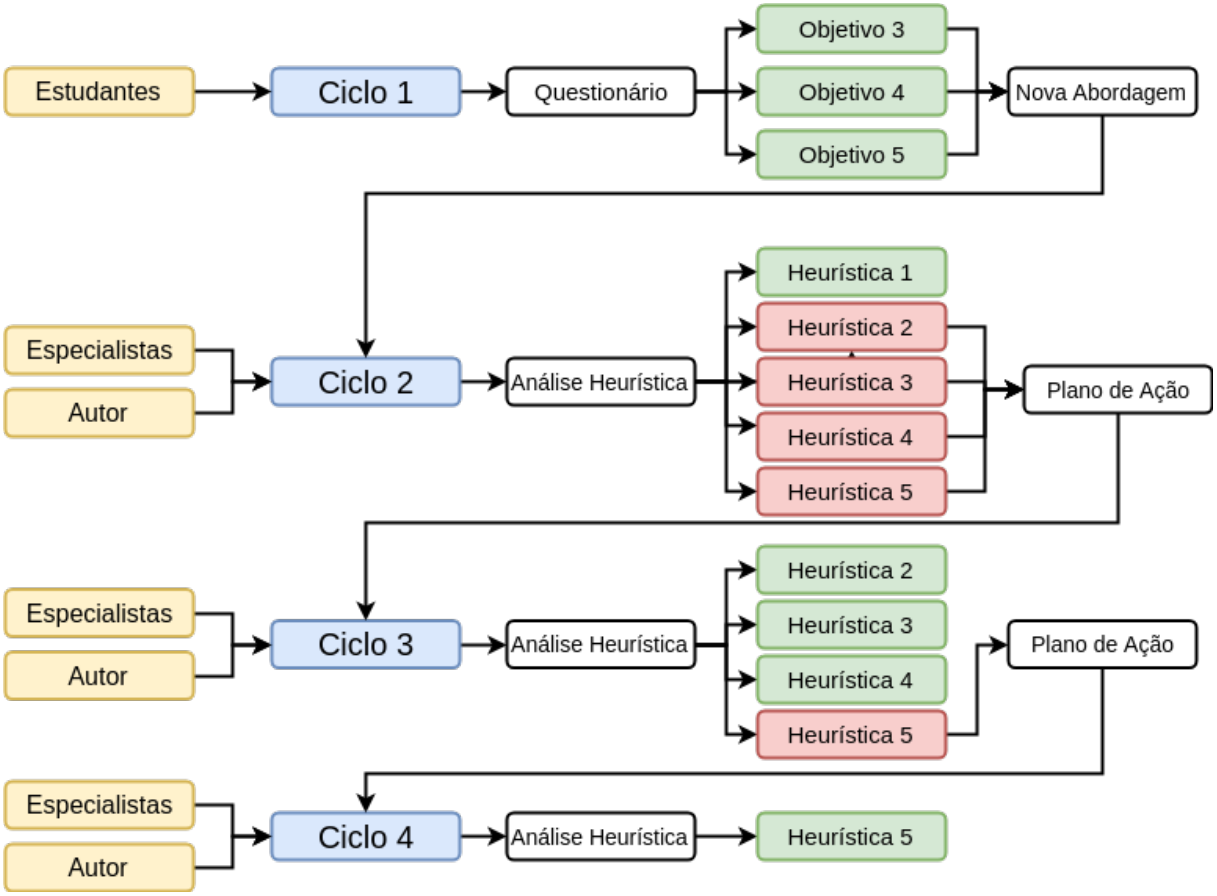


Figura 34 – Resumo dos ciclos de Pesquisa-Ação

7 Considerações Finais

Existe uma correlação entre Engenharia de Requisitos e Projeto de Software. Considerando que, em Projeto de Software, há a preocupação em estabelecer a linha de base arquitetural, condizente com os requisitos acordados, tem-se forte relação entre Requisitos de Software e Arquitetura de Software. Tal relação é evidenciada, principalmente, via a atividade de modelagem (SOMMERVILLE, 2011). Portanto, são desenvolvidos modelos abstratos que representam perspectivas do software. Esses modelos auxiliam na elicitação e na análise dos requisitos, se considerarmos a Engenharia de Requisitos. Em Projeto de Software, esses modelos auxiliam, dentre outros aspectos, na obtenção da linha de base arquitetural a ser adotada. Portanto, modelos, em diferentes níveis de abstração, são relevantes no desenvolvimento de produtos de software. Entretanto, há evidências na área, acordadas na literatura (conforme colocado na seção 1.1 dessa monografia), de que os desenvolvedores frequentemente optam por iniciar a codificação sem ao menos iniciar uma especificação arquitetural prévia. Tal prática comumente compromete a manutenção bem como a evolução do software desenvolvido.

A aplicação Model IT foi inspirada nesse cenário, como uma iniciativa com o intuito de contribuir para geração semiautomática de código usando modelos como base. Entretanto, para viabilizar o desenvolvimento de uma primeira versão dessa aplicação, foram estabelecidos: (i) um perfil de aplicação específico, no caso, aplicações web; (ii) uma notação de modelagem específica, no caso, a notação i*; (iii) um padrão arquitetural específico, no caso, o Padrão MVC, e (iv) uma linguagem de programação específica, no caso, orientada ao *framework* Django. A ideia é apoiar os desenvolvedores na construção de aplicações web, considerando a geração semiautomática de código em Django, orientando-se por modelos em i* e o Padrão Arquitetural MVC, e fazendo uso de heurísticas transformacionais bem como de princípios de MDA. Dessa forma, têm-se, a partir de transformações de modelos, em níveis de abstração diferentes, a geração de um código preliminar (i.e. *template* de código), em Django, para a aplicação web desejada.

Nesse capítulo, serão apresentadas as considerações finais sobre esse trabalho, procurando relembrar os objetivos Geral e Específicos acordados no Capítulo 1, Introdução, dessa monografia; evidenciando se os mesmos foram atingidos ou não, com justificativas, e retomando alguns aspectos e resultados documentados ao longo da monografia, em capítulos anteriores. Por fim, têm-se a apresentação dos trabalhos futuros, os quais procuram complementar essa iniciativa, tanto em suas competências quanto em suas fragilidades.

7.1 Objetivos Alcançados

7.1.1 Objetivos específicos

1. Investigar os temas chave para o escopo do Trabalho. Particularmente, serão aprofundados os estudos sobre Arquitetura de Software, MDA, modelagem i*, e padrão arquitetural MVC. Nesse caso, sendo utilizada a pesquisa bibli- ográfica como base para condução desses estudos.

   Tal objetivo foi alcançado ainda na primeira parte do trabalho.

2. Levantamento de boas práticas da Engenharia de Software a serem empregadas na geração de código, tais como técnicas de programação (MARTIN, 2008) e padrões de projeto (GAMMA et al., 1995).

   Tal objetivo foi atingido, mas não com padrões de projeto tradicionais e técnicas de programação tradicionais. Tal fato deu-se, principalmente, pois a notação utilizada para modelagem foi uma notação mais emergente, no caso i*, bem como a linguagem de programação utilizada foi uma linguagem orientada ao *framework* Django.

   No caso dos padrões de projeto, padrões estabelecidos via heurísticas transformacionais e em aderência aos referenciais conceituais do *framework* i* (orientado à meta) foram utilizados. No caso das técnicas de programação, boas práticas do Paradigma Orientado à Convenção e do Padrão Arquitetural MVC foram utilizadas.

3. Desenvolvimento de uma aplicação que auxilie arquitetos e/ou desenvolvedores de software no desenvolvimento de aplicações web utilizando o *framework* Django, orientando-se pelo padrão arquitetural MVC. Tal aplicação procurará automatizar parte do processo de geração de código.

   Tal objetivo foi atingido e validado durante os ciclos de pesquisa-ação.

4. Geração parcial do código a partir de modelos i*.

   Tal objetivo foi atingido e validado durante os ciclos de pesquisa-ação.

5. Coleta e avaliação de resultados preliminares.

   Tal objetivo foi atingido e validado durante os ciclos de pesquisa-ação.

A Tabela 8 apresenta os objetivos - identificados por números; o status final de cada um e, por fim, algumas observações extras.

7.1.2 Objetivo Geral

O objetivo geral do trabalho, apresentado no capítulo 1 é Gerar parte do código, semi-automaticamente, de aplicações Django, orientando-se por uma baseline arquitetural

Tabela 8 – Objetivos Alcançados

Objetivo	**Status Final**	**Observações Extras**
1	100%	Realizado, predominantemente, via pesquisa bibliográfica
2	100%	Padrões estabelecidos via heurísticas transformacionais e Paradigma Orientado à Meta foram utilizados, bem como boas práticas do Paradigma Orientado à Convenção e do Padrão Arquitetural MVC foram utilizadas.
3	100%	Realizado usando como base uma adaptação do Scrum (Seção 4.2.1); heurísticas transformacionais, princípios de MDA e outros referenciais teóricos Capítulo 2, e os suportes tecnológicos acordados no Capítulo 3.
4	100%	Realizado, disponível no Capítulo 5
5	100%	Realizado, disponível no Capítulo 6

bem como fazendo uso de princípios de MDA e modelagem intencional (no caso, na notação i*). Tendo em vista que os objetivos específicos foram atingidos, pode-se concluir que o objetivo geral do presente trabalho foi atingido com sucesso.

7.2 Competências da Model IT

Por se tratar de uma aplicação autônoma, a Model IT possui alguns pontos de destaque, tais como a modelagem interna utilizando uma linguagem de modelagem emergente, o i*. Tal linguagem permite a modelagem de aspectos organizacionais. Outro ponto de destaque é a transformação de tais modelos em código. Essa transformação deu-se graças a boas práticas da Engenharia de Software, tais como: heurísticas transformacionais modelo-código acordadas por autores da área; princípios transformacionais e de interoperabilidade do *framework* MDA; Padrão Arquitetural MVC para o desenvolvimento de aplicações web, e geração de código orientada a convenções com base no *framework* Django.

7.3 Fragilidades da Model IT

Tais fragilidades foram percebidas ao longo da realização dos ciclos de pesquisa-ação. Questões de usabilidade foram algumas dessas fragilidades percebidas. Entretanto, como já justificado, por se tratar de uma iniciativa preliminar nesse contexto, com desafios mais relevantes que a própria usabilidade, recomendam-se estudos e esforços futuros para alinhamento dessas questões. Dessa forma, não foram vistos como prioritários nessa primeira iniciativa. Outra fragilidade percebida compreendeu certa inconsistência, devido à instabilidade da rede. Nesse caso, algumas ações no diagrama podem ser perdidas, caso

não tenham sido salvas, antes de ocorrerem algumas instabilidades de comunicação de rede entre cliente e servidor.

7.4 Trabalhos Futuros

Para trabalhos futuros, podem ser considerados os seguintes itens:

1. Aplicação de ciclos de usabilidade;

   Como já mencionado, a aplicação possui alguns defeitos no quesito de usabilidade. Portanto, seria relevante para este trabalho, a aplicação de ciclos de usabilidade, afim de melhorar este ponto. Um dos processos conhecidos para a aplicação deste é o processo de usabilidade (MAYHEW, 1999).

2. Implementação de colaboração *real-time*; Um ponto relevante para a utilização da aplicação é a implementação de uma colaboração na qual dois ou mais usuários pudessem modelar, simultaneamente, o mesmo modelo. Para tal, seria importante a implementação utilizando bibliotecas para isso, como por exemplo Socket.IO (SOCKETIO, 2018).

Referências

ABID, M. R. et al. A UML Profile for Goal-Oriented Modeling. In: *SDL Forum*. [S.l.]: Springer, 2009. p. 133–148. Citado na página 32.

ALMISNED, F.; KEPPENS, J. Requirements Analysis: Evaluating KAOS Models. *Journal of Software Engineering and Applications*, v. 03, n. 09, p. 869–874, 2010. ISSN 1945-3116, 1945-3124. Disponível em: <http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jsea.2010.39101>. Citado na página 22.

ARCIDIACONO, G. *Comparative research about high failure rate of IT projects and opportunities to improve*. [S.l.]: PM World Journal, 2017. ISBN 0201788918. Citado na página 23.

CANONICAL. *canonical*. [S.l.], 2017. Disponível em: <https://www.canonical.com/>. Citado na página 39.

CARES, C.; GUTIéRREZ, J. F. Towards a framework for improving goal-oriented requirement models quality. In: *12th Workshop on Requirements Engineering*. [s.n.], 2009. p. 3–14. Disponível em: <https://upcommons.upc.edu/handle/2117/12993>. Citado na página 32.

CONSERVANCY, S. F. *Git*. [S.l.], 2017. Disponível em: <https://git-scm.com/>. Citado na página 39.

DALPIAZ, F.; FRANCH, X.; HORKOFF, J. istar 2.0 language guide. *arXiv preprint arXiv:1605.07767*, 2016. Disponível em: <https://arxiv.org/abs/1605.07767>. Citado 4 vezes nas páginas 11, 22, 33 e 35.

DJANGO. *Django*. [S.l.], 2017. Disponível em: <https://www.djangoproject.com/>. Citado na página 38.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2. Citado 2 vezes nas páginas 24 e 82.

GARTNER, I. *Gartner Says Global IT Spending to Reach $3.5 Trillion in 2017*. 2016. Disponível em: <http://www.gartner.com/newsroom/id/3482917>. Citado na página 23.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de Pesquisa*. [S.l.]: Luciane Delani, 2009. Citado na página 43.

GIL, A. C. *Como Elaborar Projetos de Pesquisa*. [S.l.]: EDITORA ATLAS S.A., 1991. Citado 3 vezes nas páginas 44, 47 e 48.

GITLAB. *GitLab*. [S.l.], 2017. Disponível em: <https://about.gitlab.com/>. Citado na página 40.

GO.JS. *GO.js*. [S.l.], (acessado em 17 de setembro de 2017). Disponível em: <https://gojs.net/latest/index.html>. Citado na página 37.

HISTORY, R. R. C. for; MEDIA, N. *Zoterocker*. [S.l.], 2017. Disponível em: <https://www.zotero.org/>. Citado na página 40.

JACYNTHO, M. D.; SCHWABE, D.; ROSSI, G. A software architecture for structuring complex web applications. *J. Web Eng.*, v. 1, n. 1, p. 37–60, 2002. Disponível em: <https://www.researchgate.net/profile/Daniel_Schwabe/publication/220538185_A_Software_Architecture_for_Structuring_Complex_Web_Applications/links/0fcfd510be95fb3781000000.pdf>. Citado na página 22.

KALELKAR, M.; CHURI, P.; KALELKAR, D. Implementation of model-view-controller architecture pattern for business intelligence architecture. v. 102, n. 12, 2014. Disponível em: <http://search.proquest.com/openview/8c078dd4291e0b872586cac8cc4a5337/1?pq-origsite=gscholar&cbl=136216>. Citado 3 vezes nas páginas 11, 27 e 28.

LAKHOUA, M. N.; ANNABI, M. Approach of analysis of methods SA, SADT and SART. In: *Information and Communication Technologies, 2006. ICTTA'06. 2nd*. [S.l.]: IEEE. v. 1, p. 473–477. Citado na página 53.

LATEX. *latex*. [S.l.], 2017. Disponível em: <https://www.latex-project.org/>. Citado na página 40.

LITVIN, M.; LITVIN, G. Implementation of model-view-controller architecture pattern for business intelligence architecture. n. 12, 2007. Citado 2 vezes nas páginas 11 e 29.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008. ISBN 0132350882, 9780132350884. Citado 2 vezes nas páginas 24 e 82.

MAYHEW, D. J. *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design*. 1st. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN 1558605614, 9781558605619. Citado na página 84.

MAZóN, J. T. n. . J.-N. An mda approach for the development of data warehouses. *Decision Support Systems*, v. 45, n. 1, p. 41 – 58, 2008. Citado na página 32.

MELLOR, S. J. et al. *MDA Distilled*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN 0201788918. Citado 6 vezes nas páginas 11, 22, 29, 30, 31 e 32.

MELO, J. et al. Formalization of Mapping Rules from iStar to Class Diagram in UML. In: . IEEE, 2015. p. 71–79. ISBN 978-1-4673-9272-3. Disponível em: <http://ieeexplore.ieee.org/document/7328011/>. Citado 7 vezes nas páginas 11, 51, 52, 54, 55, 59 e 79.

MICROSOFT. *microsoft*. [S.l.], 2017. Disponível em: <https://www.microsoft.com/pt-br>. Citado na página 39.

MONGODB. *mongodb*. [S.l.], 2017. Disponível em: <https://www.mongodb.com>. Citado na página 38.

MOZILLA. *Mozilla*. [S.l.], 2017. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/O_que_e_JavaScript>. Citado na página 37.

NODE.JS. *NODE.js*. [S.l.], 2017. Disponível em: <https://nodejs.org/en/about/>. Citado na página 38.

OMG. *OMG*. [S.l.], 2017. Disponível em: <http://www.omg.org/>. Citado na página 21.

OMG. Omg unified modeling language tm (omg uml). 2017. Disponível em: <http://www.omg.org/spec/UML/2.5/PDF/>. Citado na página 32.

PRAJAPATI, H. B.; DABHI, V. K. High quality web-application development on java EE platform. In: . IEEE, 2009. p. 1664–1669. ISBN 978-1-4244-2927-1. Disponível em: <http://ieeexplore.ieee.org/document/4809267/>. Citado 2 vezes nas páginas 22 e 28.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. 5th ed. ed. [S.l.]: McGraw Hill, 2000. (McGraw-Hill series in computer science). ISBN 978-0-07-365578-9. Citado na página 21.

PYTHON. *General Python FAQ*. [S.l.], 2017. Disponível em: <https://docs.python.org/3/faq/general.html#what-is-python>. Citado na página 38.

RICHARDS, M. *Software Architecture Patterns*. United States of America: O'Reully, 2015. ISBN 978-0-596-51798-4. Citado 2 vezes nas páginas 21 e 27.

RRCHNM. *rrchnm*. [S.l.], 2017. Disponível em: <https://rrchnm.org/>. Citado na página 40.

SCRUM-GUIDE. *Scrum*. [S.l.], 2017. Disponível em: <http://www.scrumguides.org/scrum-guide.html>. Citado na página 47.

SIEGEL, J. M. Object management group model driven architecture (mda) mda guide rev. 2.0. v. 02, 2014. Disponível em: <http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf>. Citado na página 30.

SOCKETIO. *SocketIO*. [S.l.], 2018. Disponível em: <https://socket.io/>. Citado na página 84.

SOMMERVILLE, I. *Software engineering*. 9th ed. ed. [S.l.]: Pearson, 2011. OCLC: ocn462909026. ISBN 978-0-13-703515-1 978-0-13-705346-9. Citado 2 vezes nas páginas 21 e 81.

SPINELLIS, D.; GOUSIOS, G. *Beautiful architecture*. Sebastopol, Calif: O'Reully, 2009. ISBN 978-0-596-51798-4. Citado 2 vezes nas páginas 24 e 27.

TAYLOR, R. N.; MEDVIDOVIć, N.; DASHOFY, E. M. *Software architecture: foundations, theory, and practice*. [S.l.]: John Wiley, 2010. OCLC: ocn231670965. ISBN 978-0-470-16774-8. Citado na página 21.

TRELLO. *Trello*. [S.l.], 2017. Disponível em: <https://www.trello.com/>. Citado na página 39.

UBUNTU. *ubuntu*. [S.l.], 2017. Disponível em: <https://www.ubuntu.com/>. Citado na página 39.

VIEIRA, R. S.; REIS, U. ABORDAGEM DIRIGIDA AO DOMÍNIO APLICADO NA ARQUITETURA DE SISTEMAS WEB. *Revista de Sistemas e Computação-RSC*, v. 4, n. 2, 2015. Disponível em: <http://www.revistas.unifacs.br/index.php/rsc/article/view/3090>. Citado 6 vezes nas páginas 11, 22, 29, 30, 31 e 32.

VSCODE. *vscode*. [S.l.], 2017. Disponível em: <https://code.visualstudio.com/>. Citado na página 39.

VUE.JS. *Vue.js*. [S.l.], 2017. Disponível em: <https://br.vuejs.org/v2/guide/>. Citado na página 37.

WHYTE, W. F. Os dez mandamentos da observação participante. 2005. Disponível em: <http://www.scielo.br/pdf/rbcsoc/v22n63/a12v2263.pdf>. Citado 3 vezes nas páginas 68, 71 e 74.

YRJöNEN, A.; MERILINNA, J. *Extending the NFR Framework with Measurable NonFunctional Requirements*. Hoboken, N.J: Wiley-Interscience : IEEE Computer Society, 2008. OCLC: ocn154798454. ISBN 978-0-470-03666-2. Citado na página 22.

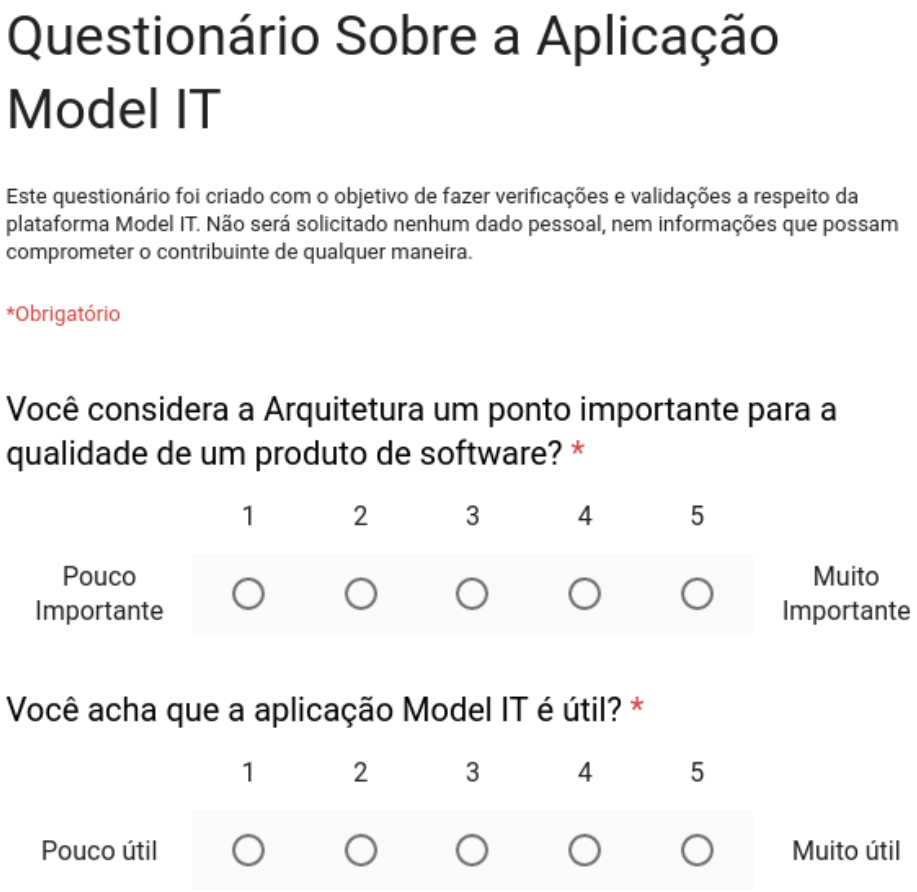
YU, E. Strategic modelling for enterprise integration. In: *Proceedings of the 14th world congress of the international federation of automatic control*. [s.n.], 1999. p. 5–9. Disponível em: <ftp://ftp.sys.utoronto.ca/pub/eric/eric/IFAC99.pdf>. Citado na página 34.

YU, E. S. Towards modelling and reasoning support for early-phase requirements engineering. In: *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*. IEEE, 1997. p. 226–235. Disponível em: <http://ieeexplore.ieee.org/abstract/document/566873/>. Citado na página 33.

ZARRAS, A. Applying model-driven architecture to achieve distribution transparencies. *Information and Software Technology*, v. 48, n. 7, p. 498 – 516, 2006. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584905000856>. Citado na página 32.

Apêndices

APÊNDICE A – Questionário de Avaliação da aplicação Model IT



Questionário Sobre a Aplicação Model IT

Este questionário foi criado com o objetivo de fazer verificações e validações a respeito da plataforma Model IT. Não será solicitado nenhum dado pessoal, nem informações que possam comprometer o contribuinte de qualquer maneira.

*Obrigatório

Você considera a Arquitetura um ponto importante para a qualidade de um produto de software? *

	1	2	3	4	5	
Pouco Importante	○	○	○	○	○	Muito Importante

Você acha que a aplicação Model IT é útil? *

	1	2	3	4	5	
Pouco útil	○	○	○	○	○	Muito útil

Figura 35 – Questionário Parte 1

Visto que o objetivo da aplicação é a geração Parcial de código Django através de Modelos I*, você acha que a aplicação cumpre com o proposto? *

	1	2	3	4	5	
Discordo Totalmente	○	○	○	○	○	Concordo Plenamente

Você acha que a aplicação consegue lhe poupar tempo, visto que gera uma porção do código? *

	1	2	3	4	5	
Poupa pouco tempo	○	○	○	○	○	Poupa Muito tempo

Você faria o uso da Aplicação Model IT em seus projetos? *

- ○ Sim
- ○ Não

O que você sugere que poderia ser diferente/melhor e/ou o que poderia ser acrescentado na aplicação Model IT? *

Sua resposta

Figura 36 – Questionário Parte 2

APÊNDICE B – Código do Transpilador

```
const fs = require('fs');

exports.codingControllers = async (diagram, writeFile=true, strings) => {
    var myDiagram = JSON.parse(diagram.json)
    myDiagram = parserLinkArray(myDiagram)
    var controller =
`from django.http import HttpResponse
`
    for(var model in Object.keys(strings)){
        controller = controller +
`from .models import ${Object.keys(strings)[model]}
`
    }
    var goals = {}
    for (var fragment in myDiagram.nodeDataArray) {
        var methods = ``
        var resolutions = ``
        if(myDiagram.nodeDataArray[fragment].category == 'goal'){
            for(var link in myDiagram.linkDataArray){
                if(myDiagram.linkDataArray[link].to == myDiagram.
                    nodeDataArray[fragment].key){
                    goals[`${myDiagram.linkDataArray[link].from}`] = []
                    goals[`${myDiagram.linkDataArray[link].from}`].push(
`    #Should resolve ${myDiagram.nodeDataArray[fragment].text} softgoal...
`)
                }
            }
        }

        var orTask = []
        var orTaskName = []
        var andTaskName = []
        orTaskName[`${myDiagram.nodeDataArray[fragment].text}`] = ``
        andTaskName[`${myDiagram.nodeDataArray[fragment].text}`] = ``
        if(myDiagram.nodeDataArray[fragment].category == 'task'){
            for(var link in myDiagram.linkDataArray){
                if(myDiagram.linkDataArray[link].to == myDiagram.
                    nodeDataArray[fragment].key && myDiagram.linkDataArray[
                    link].category == '-|>'){
                    orTask[`${myDiagram.nodeDataArray[fragment].text}`] =
                        true

```

```
                        orTaskName[`${myDiagram.nodeDataArray[fragment].text}`]
                            = orTaskName[`${myDiagram.nodeDataArray[fragment].
                            text}`] +
`
    #or
    ${findText(myDiagram.nodeDataArray ,myDiagram.linkDataArray[link].from)
        }()`.toLowerCase()
                } else if(myDiagram.linkDataArray[link].to == myDiagram.
                    nodeDataArray[fragment].key && myDiagram.linkDataArray[
                    link].category == '-|-'){

                    andTaskName[`${myDiagram.nodeDataArray[fragment].text
                        }`] = andTaskName[`${myDiagram.nodeDataArray[
                        fragment].text}`] +
`
    ${findText(myDiagram.nodeDataArray ,myDiagram.linkDataArray[link].from)
        }()`.toLowerCase()
                }
            }
        }


        if(myDiagram.nodeDataArray[fragment].category == 'task'){
            for(var link in myDiagram.linkDataArray){
                if(orTask[`${myDiagram.nodeDataArray[fragment].text}`]){
                    methods =
`
    #Either${andTaskName[`${myDiagram.nodeDataArray[fragment].text}`]}${
        orTaskName[`${myDiagram.nodeDataArray[fragment].text}`]}`
                } else{
                    methods = andTaskName[`${myDiagram.nodeDataArray[
                        fragment].text}`]
                }

            }
            if(goals[myDiagram.nodeDataArray[fragment].key]){
                resolutions = resolutions + goals[myDiagram.nodeDataArray[
                    fragment].key][0]
            }
            controller = controller +
`
def ${myDiagram.nodeDataArray[fragment].text.toLowerCase().replace(' ','_')
    }(request):${methods}
${resolutions}    return HttpResponse()

`
        }
```

```
74      }
75      if(writeFile){
76          classGenerator('views', controller)
77      }
78      return controller
79
80  }
81
82
83  exports.coding = async (diagram, writeFile=true) => {
84
85      var models =
86  `from django.db import models
87  `
88      var myDiagram = JSON.parse(diagram.json)
89      myDiagram = parserLinkArray(myDiagram)
90      console.log(myDiagram)
91      var strings = {}
92      var attr = catchAttr(myDiagram.nodeDataArray, myDiagram.linkDataArray)
93      for (var fragment in myDiagram.nodeDataArray) {
94          if(myDiagram.nodeDataArray[fragment].category == 'actor'){
95              strings[`${myDiagram.nodeDataArray[fragment].text}`] = `
96  class ${myDiagram.nodeDataArray[fragment].text}(models.Model):
97  `
98          }
99      }
100     for(var link in myDiagram.linkDataArray){
101         if(myDiagram.linkDataArray[link].text){
102             if(myDiagram.linkDataArray[link].category == '-->' && myDiagram
                    .linkDataArray[link].text.match(/is-a/i)){
103             strings[`${findText(myDiagram.nodeDataArray, myDiagram.
                    linkDataArray[link].from)}`] = `
104 class ${findText(myDiagram.nodeDataArray, myDiagram.linkDataArray[link].
    from)}(${findText(myDiagram.nodeDataArray, myDiagram.linkDataArray[link
    ].to)}):
105 `
106         }}
107     }
108     for (var fragment in myDiagram.nodeDataArray) {
109         if(myDiagram.nodeDataArray[fragment].category == 'actor'){
110             var stringAttr = ``
111             for(var att in attr[`${myDiagram.nodeDataArray[fragment].text
                    }`]){
112                 stringAttr = stringAttr +
113 `    ${attr[`${myDiagram.nodeDataArray[fragment].text}`][att]}
114 `
115             }
```

```
116             strings[`${myDiagram.nodeDataArray[fragment].text}`] = strings
                    [`${myDiagram.nodeDataArray[fragment].text}`] +
117 stringAttr
118         }
119     }
120     for(var model in strings){
121         models = models + strings[model]
122     }
123     if(writeFile){
124         var controllers = this.codingControllers(diagram, writeFile,
                strings)
125         classGenerator('models', models)
126     } else {
127         return [models, strings]
128     }
129 }
130
131
132 classGenerator = (file, classInfo) => {
133
134     fs.writeFile(__dirname + `/../../django-boilerplate/mysite/mysite/${
            file}.py`, classInfo, (err) => {
135         if (err) {
136           return console.log(err);
137         }
138       })
139 };
140
141 parserLinkArray = (diagram) => {
142     for(var iterator in diagram.linkDataArray){
143         if(diagram.linkDataArray[iterator].from.match(/task/)
144         && diagram.linkDataArray[iterator].to.match(/goal/)
145         && diagram.linkDataArray[iterator].category.match(/-\|-/)){
146             var goal = diagram.linkDataArray[iterator].to
147             var desc = `metatask${findText(diagram.nodeDataArray, diagram.
                    linkDataArray[iterator].to)}`
148             diagram.linkDataArray[iterator] = {
149                 from: diagram.linkDataArray[iterator].from,
150                 to: desc,
151                 category: diagram.linkDataArray[iterator].category
152             }
153             var pushing = true
154             for(var y in diagram.linkDataArray){
155                 if(diagram.linkDataArray[y].from == desc){
156                     pushing = false
157                 }
158             }
```

```
159             if(pushing){
160                 diagram.linkDataArray.push({
161                     from: desc,
162                     to: goal,
163                     category: ''
164                 })
165             }
166 
167 
168             var pushing = true
169             for(var x in diagram.nodeDataArray){
170                 if(diagram.nodeDataArray[x].key == desc){
171                     pushing = false
172                 }
173             }
174             if(pushing){
175                 diagram.nodeDataArray.push({
176                     category: 'task',
177                     key: desc,
178                     text: desc,
179                     loc: '367.4945578231293 41.62494331065761',
180                     group: 'actor'
181                 })
182             }
183         }
184     }
185     return diagram
186 }
187 findText = (iterator, finder) => {
188     for(var x in iterator){
189         if(iterator[x].key == finder){
190             return iterator[x].text
191         }
192     }
193 }
194 
195 findKey = (iterator, finder) => {
196     for(var x in iterator){
197         if(iterator[x].text == finder){
198             return iterator[x].key
199         }
200     }
201 }
202 
203 catchAttr = (nodeDataArray, linkDataArray) => {
204     var attr = {}
205     for (var fragment in nodeDataArray) {
```

```
206             attr[`${nodeDataArray[fragment].text}`] = []
207             for(var link in linkDataArray){
208                 if(nodeDataArray[fragment].key == linkDataArray[link].from){
209                     if(linkDataArray[link].to.match(/^quality.*/)) {
210                         attr[`${nodeDataArray[fragment].text}`].push(findText(
                                nodeDataArray, linkDataArray[link].to))
211
212                     }
213                 }
214             } if(nodeDataArray[fragment].group){
215                 if(nodeDataArray[fragment].category.match(/^quality.*/)) {
216                     var text = findText(nodeDataArray, nodeDataArray[fragment].
                            group)
217                     attr[`${text}`].push(`${nodeDataArray[fragment].text} =
                            models.CharField(max_length=100) #coming from ${
                            nodeDataArray[fragment].category}`)
218
219                 } else if(nodeDataArray[fragment].category.match(/^resource.*/)
                        ) {
220                     var text = findText(nodeDataArray, nodeDataArray[fragment].
                            group)
221                     attr[`${text}`].push(`${nodeDataArray[fragment].text} =
                            models.CharField(max_length=100) #coming from ${
                            nodeDataArray[fragment].category}`)
222
223                 }
224             }
225
226         }
227         return attr;
228     }
```

Listing B.1 – Código do transpilador