Universidade Federal do Maranhão

Centro de Ciência Exatas e Tecnologia

Curso de Ciência da Computação

Rodolfo Sobreira Alves

Apoiando a Análise de Requisitos Através de Técnicas de Processamento de Linguagem Natural

São Luís - MA

2020

Rodolfo Sobreira Alves

Apoiando a Análise de Requisitos Através de Técnicas de Processamento de Linguagem Natural

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Curso de Ciência da Computação

Universidade Federal do Maranhão

Orientador: Prof. Dr. Davi Viana dos Santos

São Luís - MA

2020

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Alves, Rodolfo Sobreira.
Apoiando a Análise de Requisitos Através de Técnicas de Processamento de Linguagem Natural / Rodolfo Sobreira Alves. - 2020.
46 f.

Coorientador(a): Luciano Reis Coutinho.
Orientador(a): Davi Viana dos Santos.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís - MA, 2020.

1. Análise de Requisitos. 2. Avaliação de Requisitos. 3. Engenharia de Requisitos. 4. Processamento de Linguagem Natural. I. Coutinho, Luciano Reis. II. Santos, Davi Viana dos. III. Título.

Rodolfo Sobreira Alves

Trabalho de Conclusão de Curso apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dr. Davi Viana dos Santos - Orientador
Universidade Federal do Maranhão - UFMA

Prof. Dr. Luciano Reis Coutinho - Coorientador
Universidade Federal do Maranhão - UFMA

Prof. Dr. Luis Jorge Enrique Rivero Cabrejos - Membro da Banca
Universidade Federal do Maranhão - UFMA

Prof. Dr. Sergio Souza Costa - Membro da Banca
Universidade Federal do Maranhão - UFMA

Agradecimentos

"A melhor maneira de prever o futuro
é inventá-lo"
- Alan Turing

Primeiramente gostaria de agradecer a Deus, por ter me dado saúde e força para superar as dificuldades. Sou grato pela minha família e pela minha namorada, pelo apoio e pelo suporte que me deram por todo esse período, principalmente para as mulheres da minha vida, minha mãe Elizabeth e minha namorada Amanda, por serem as minhas inspirações de garra e força de vontade. Gostaria também de agradecer aos meus amigos, que me apoiaram durante todos esses períodos da faculdade, principalmente ao meu amigo Jesse, que foi meu "duo" durante todos esses anos na UFMA, e é uma pessoa que se não fosse por ele, não estaria aqui escrevendo esse documento. Além disso, gostaria de agradecer ao meu orientador, professor Davi Viana, por todo o suporte oferecido, e pelo conhecimento adquirido por mim, e ao professor Francisco José, que me auxiliou e me deu oportunidades em um momento que estava muito desanimado com o curso. Deixo um agradecimento especial também a todos os membros do Laboratório de Sistemas Distribuídos Inteligentes, que foi minha segunda casa durante quase 3 anos, e lá onde tive contato com pessoas maravilhosas, como Anderson, Tércio, Marcelino, Danne, companheiros de pesquisa, sempre muito presentes e prestativos.

Esse trabalho foi apoiado em parte pela CAPES Código de Financiamento 001 e 88887.200532/2018-00 (PROCAD-Amazônia); pela Fundação de Amparo à Pesquisa do Estado do Maranhão - FAPEMA (UNIVERSAL-00745/19; e pelo INCT de Internet do Futuro para Cidades Inteligentes (CNPq 465446/ 2014-0, CAPES 88887.136422/2017-00, e FAPESP 14/50937-1 & 15/24485-9) e CNPq 311608/2017-5, 420907/2016-5, 312324/2015-4.

Rodolfo Sobreira Alves. Apoiando a Análise de Requisitos Através de Técnicas de Processamento de Linguagem Natural. Trabalho de Conclusão de Curso. Centro de Ciências Exatas e Tecnologias, Universidade Federal do Maranhão, São Luís, 2020.

Resumo

A fase de elicitação de requisitos é uma importante etapa para o desenvolvimento de um sistema, já que é nela onde serão estabelecidos todos os comportamentos do mesmo. Erros nos documentos gerados nesta etapa podem vir a prejudicar o restante do processo de desenvolvimento, por isso, uma análise desses documentos é de suma importância para o avanço correto por entre as etapas da concepção de um sistema. Erros como ambiguidade e incompletude podem gerar requisitos falhos e incompletos. A detecção manual destes requisitos é custosa, e tendencioso a falhas, ainda mais quando o sistema possui uma grande complexidade. A detecção automática desses erros se apresenta como uma abordagem promissora. O objetivo deste trabalho, é auxiliar o processo de análise de requisitos de maneira automática, focando na detecção de requisitos ambíguos e incompletos. Para isso foi gerada uma ferramenta que faz uso de técnicas de processamento de linguagem natural, para a detecção dos erros de ambiguidade e incompletude. Além disso, a ferramenta gerada é capaz de avaliar se o requisito está no contexto de cidades inteligentes e dar sugestões para incompletude de requisitos de cidades inteligentes. Para a comprovação dos resultados da ferramenta, foi realizada uma prova de conceito a fim de comprovar a eficácia da ferramenta proposta.

Palavras-Chave:

Processamento de Linguagem Natural, Engenharia de Requisitos, Análise de Requisitos, Avaliação de Requisitos

Sumário

Capítulo 1

Introdução

1.1 Contextualização do trabalho

A fase de elicitação de requisitos é uma das etapas mais importantes para o desenvolvimento de um sistema (NIGAM, 2012). Nela serão especificadas e modeladas todas as funcionalidades da aplicação, até mesmo interfaces e interações com o usuário. Essas especificações são definidas como requisitos, e serão utilizados para a concepção da ferramenta. Um requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos (SOMMERVILLE, 1997). Eles são importantes por estabelecer uma base de concordância entre o cliente e o fornecedor sobre o que o software deverá realizar. Um requisito pode passar por alterações, por meio de análises dos mesmos. A junção das etapas de elicitação e gerência de requisitos gera um processo chamado de engenharia de requisitos.

Conseguir identificar uma não conformidade logo na etapa de elicitação de requisitos é bem menos custoso que identificar já na etapa de testes por exemplo, além de ser bem mais fácil de corrigir tal erro (UMBER, 2011). Porém, essa análise, se feita de maneira manual, pode ser muito custosa para a equipe de desenvolvimento, já que algum membro da equipe terá que dedicar horas para realizar essa análise, ainda mais quando se trata de equipes pequenas ou de aplicações complexas, com uma lista extensa de requisitos.

Erros como requisitos ambíguos, incompletos, ou fora do contexto do projeto tendem a prejudicar o desenvolvimento, gerando ferramentas falhas, incompletas e que podem não corresponder ao requisitado. Na maioria dos projetos de software, os documentos de requisitos são escritos quase que inteiramente em linguagem natural, em outros documentos pode usar algum tipo de modelagem para representar os requisitos. Por isso, tecnologias de processamento de linguagem natural vem sendo cada vez mais utilizadas para análise de requisitos (NIGAM, 2012).

Além disso, esse trabalho focou na validação do contexto do requisito baseado no cenário de cidades inteligentes, que são aquelas que otimizam a utilização dos recursos para

servir melhor os cidadãos. Isso vale para a mobilidade, a energia ou para qualquer serviço necessário à vida das pessoas. O trabalho está inserido no contexto dos projetos de cidades inteligentes PROCAD Amazônia (CAPES), Universal (FAPEMA) e INCT - InterSCity (CAPES e CNPq).

Ferramentas que combinam técnicas de processamento natural com avaliação de requisitos se tornam bem viáveis considerando esse cenário. Além disso, com a utilização dessas técnicas, pode-se fazer análise sobre esses requisitos e validar, por exemplo, se eles estão em um determinado contexto, como cidades inteligentes.

1.2 Justificativa

Como citado acima, a maioria dos projetos de software são escritos quase por completo em linguagem natural. O uso de técnicas de processamento de linguagem natural para a análise de documentos de requisitos se apresenta sendo uma área com bastante potencial para se explorar.

Uma ferramenta para a etapa de requisitos pode facilitar o processo de análise de sistemas, já que fazer esse processo de maneira manual tem um custo muito alto o que acaba sendo inviável e podendo fazer com que requisitos ambíguos ou incompletos passem despercebidos por essa etapa, acarretando erros nas etapas subsequentes.

1.3 Objetivos

O principal objetivo deste trabalho é apoiar umas das atividades do processo de elicitação de requisitos, mais especificamente a etapa da análise requisitos, gerando uma ferramenta que faz uso combinado de técnicas de processamento de linguagem natural utilizando uma abordagem semântica.

Além disso, identificar problemas da análise de requisitos, como ambiguidade e incompletude. A ferramenta ainda é capaz de identificar o contexto do requisito, mais especificamente relacionado a cidades inteligentes e validar a completude para esse contexto em específico.

1.4 Organização do Trabalho

No Capítulo 2, apresenta-se uma introdução à elicitação de requisitos, mais especificamente sobre a etapa de análise de requisitos, onde é apresentado modelos e comparações sobre erros nas demais etapas da engenharia de software. No capítulo 2, também será apresentada uma introdução ao processamento de linguagem natural, suas principais técnicas e algumas ferramentas que se utilizam dela.

No capítulo 3 é apresentado um modelo comparativo sobre ferramentas que fazem o uso combinado de processamento de linguagem natural para análise de requisitos. O resultado dessa análise é apresentado por meio de uma tabela comparativa entre os trabalhos.

No capítulo 4, é apresentado a ferramenta proposta, sua arquitetura, funcionamento, entradas e saídas e como ela foi proposta, além de apresentar uma prova de conceito realizada sobre a ferramenta proposta. No capítulo 5 as conclusões e os trabalhos futuros. Por último, são listadas as referências utilizadas neste documento com a lista de apêndices.

Capítulo 2

Engenharia de Software, Requisitos e Processamento de Linguagem Natural

2.1 Engenharia de Software

O desenvolvimento de *software* é uma atividade complexa por natureza. Uma das razões para esta afirmação é que não existe uma única solução para cada cenário de desenvolvimento. Além disso, o desenvolvedor lida o tempo todo com pessoas, o que torna o sucesso do projeto bastante relacionado à competência da equipe e à forma como trabalham, e, para dificultar ainda mais, muitas vezes não se usa um processo bem definido para apoiar as atividades do projeto (SOMMERVILLE, 2011).

Esse processo é a Engenharia de Software, que é a aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, operação e manutenção de software. Sistemática porque parte do princípio de que existe um processo de desenvolvimento definindo as atividades que deverão ser executadas. Disciplinada porque parte do princípio de que os processos definidos serão seguidos. Quantificável por que se deve definir um conjunto de medidas a serem extraídas do processo durante o desenvolvimento de forma que as tomadas de decisão relacionadas ao desenvolvimento do software (por exemplo, melhoria de processo) sejam embasadas em dados reais, e não em "achismos" (SCACCHI, 2012).

A engenharia de software é dividida em 6 etapas: A **elicitação de requisitos** onde serão levantados os requisitos do sistema, a etapa de **projeto** onde será feita uma compatibilização dos requisitos em função das tecnologias existentes, a **implementação** onde ocorre o desenvolvimento do sistema propriamente dito. Na etapa de **verificação e validação** são feitos testes para medir a consistência da aplicação e na **implantação** são feitas as entregas das versões da ferramenta conforme elas forem sendo liberadas. Por último temos a etapa de **manutenção**, que serve para garantir o correto funcionamento do sistema no ambiente de produção, e quando necessário retornar para o início do ciclo de desenvolvimento (PFLEEGER, 2004).

2.2 Elicitação de Requisitos

Este trabalho focou em contribuir com a etapa de elicitação de requisitos, que é a fase do projeto onde são extraídas informações do cliente sobre o que ele deseja que seja construído. É a fase em que o profissional de TI entende a necessidade do cliente e o orienta. É o momento de conversa com o usuário, de sentimento sobre o que este espera que seja entregue a ele. Na elicitação de requisitos são percebidas as necessidades do sistema e as características que esse sistema deve ter (ARELLANO, 2012). A fase de levantamento de requisitos, em um projeto, representa a parte de negócio, ou seja, o que exatamente o cliente está precisando. Nessa fase, o profissional de TI busca informações como: funcionalidades que o sistema deve ter, as regras de negócio dessas funcionalidades, restrições, usabilidade do software, e assim por diante.

Nigam (2012), apresenta uma série de comparações referentes a diversos tipos de erros nas várias etapas de desenvolvimento de um sistema, as quais foram citadas na seção 2.1, e faz comparações desses dados em relação à etapa de elicitação de requisitos. Em seu estudo, ele afirma que aproximadamente 56% dos erros encontrados durante o desenvolvimento de um sistema se encontravam na fase de elicitação de requisitos, como apresentado na Figura 1.

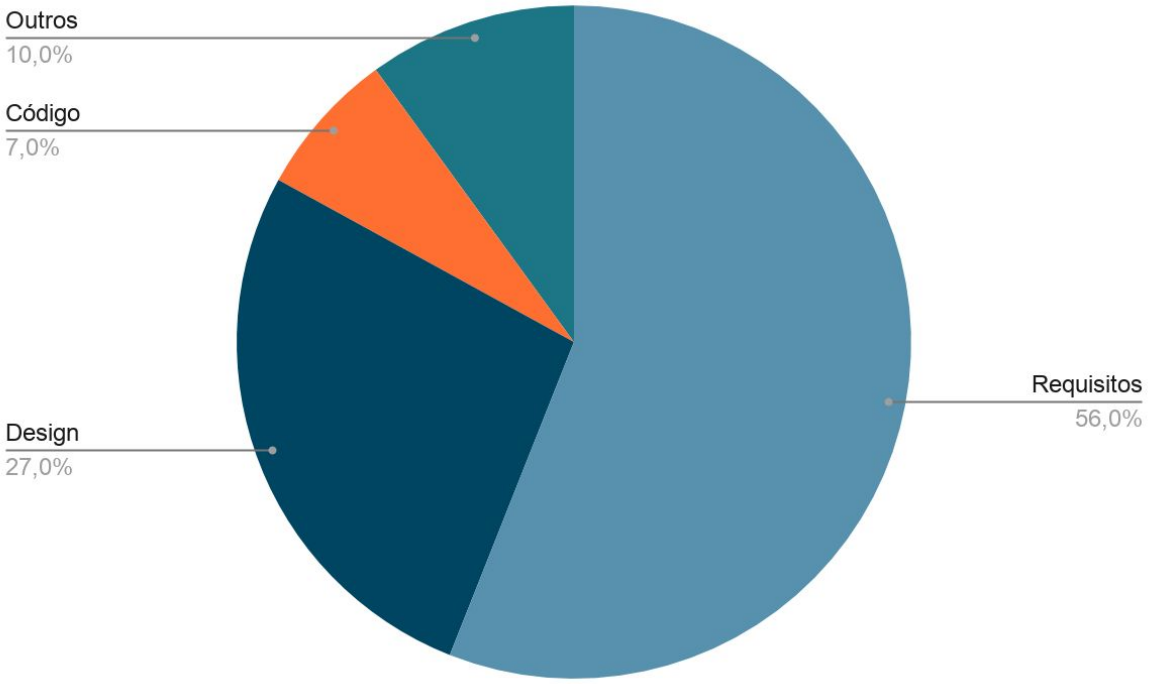

Figura 1: Erros nas Etapas de Desenvolvimento (NIGAM, 2012)

Além disso, seu trabalho (NIGAM, 2012) expõe comparativos de custos para resolver um problema em cada uma das etapas de desenvolvimento, como mostrado na tabela 1. Ele apresenta que, se esses problemas não forem resolvidos em estágio inicial, então o custo e o tempo de desenvolvimento serão diretamente afetados.

Tabela 1: Custo de Erro por Etapa de Desenvolvimento (NIGAM, 2012)

Fase em que o erro foi encontrado	Custos
Elicitação de Requisitos	1
Design	3 - 6
Desenvolvimento	10
Testes de Integração / Unitários	15 - 40
Testes de Aceitação / Sistema	30 - 70
Produção	40 - 10000

2.3 Análise de Requisitos

O sucesso de qualquer etapa de elicitação de requisitos é baseada no quão fiel ao seu documento de especificação e livre de inconsistências os requisitos são. Os erros encontrados na etapa de elicitação e análise de requisitos são mais fáceis de serem identificados e mais fáceis de serem corrigidos. Além disso, como apresentado na Tabela 1, o custo para reparo de um erro encontrado na etapa de elicitação de requisitos é bem menor do que nas outras etapas do desenvolvimento. Erros como ambiguidade de requisitos e requisitos incompletos devem gerar softwares falhos e incompletos. Por isso a importância da realização de uma análise desses requisitos, antes deles serem passados para as etapas subsequentes do desenvolvimento.

Por isso, a análise de requisitos é considerada de suma importância para o projeto. E como mostrado nos gráficos da seção anterior, é onde se pode identificar um erro de maneira mais simples e corrigi-lo de maneira menos onerosa (HUERTAS, 2012).

A análise ou verificação de requisitos consiste em inspeções para encontrar erros nos documentos ou projeto. A ideia é identificar ambiguidades e inconsistências para assegurar a qualidade do documento de requisitos antes de sua implementação (NIGAM, 2012). Porém,

avaliar requisitos de maneira manual pode ser muito custoso para a equipe de desenvolvimento, principalmente para sistemas muito complexos, que tendem a ter uma lista extensa de requisitos para analisar, em equipes muito pequenas.

Resolver manualmente problemas como ambiguidade dos requisitos de software é tedioso, demorado, sujeito a erros e, portanto, caro. Além disso, à medida que os sistemas se tornam capazes de lidar com produtos industriais de alta qualidade aplicativos, seus requisitos se tornam massivos, aumentando ainda mais o escopo de possíveis problemas dele (SHAH, 2011). Assim, um sistema automatizado com uma semi-abordagem automatizada para resolver problemas como ambiguidade e incompletude dos requisitos é completamente adequado.

2.4 Processamento de Linguagem Natural

O processamento de linguagem natural (PLN) é a subárea da Inteligência Artificial (IA) que estuda a capacidade e as limitações de uma máquina em entender a linguagem dos seres humanos, que se trata da pura forma como os humanos se comunicam (NAZIR, 2017). Os idiomas, como o inglês e o português são um ótimo exemplo de linguagem natural.

O termo “processamento”, no caso, significa análise e entendimento. Isto é, trata-se da capacidade das máquinas de lidar com a forma que falamos, superando nossos erros de ortografia, ambiguidades, abreviações, gírias e expressões coloquiais.

O objetivo do PLN é fornecer aos computadores a capacidade de entender e compor textos. “Entender” um texto significa reconhecer o contexto, fazer análise sintática, semântica, léxica e morfológica, criar resumos, extrair informação, interpretar os sentidos, analisar sentimentos e até aprender conceitos com os textos processados (NAZIR, 2017).

O PLN possui sete níveis de processamento, que vão do mais alto nível da linguística até o mais baixo nível, são eles (CHOWDHARY, 2020):

- Sintático
    - Estuda a composição, palavra por palavra, de uma frase. Essa camada analisa a formação da sentença e a especificação das estruturas permitidas na linguagem.
- Morfológico
    - Estuda as palavras de acordo com suas classes gramaticais. A morfologia cuida do que se refere à composição das palavras e sua natureza.

- Semântico
    - Carrega o significado e o sentido dos vocabulários de uma língua. Essa camada de processamento tem como objetivo compreender o significado completo da frase.
- Fonológico
    - Essa camada é voltada para quando a máquina precisa entender a linguagem falada. Ela existe para distinguir o significado dos sons das palavras.
- Discurso
    - Envolve todo o contexto da comunicação e do “diálogo”. Ela revela quem, para quem e sobre quem se fala em um determinado contexto no qual a frase está inserida.
- Lexical
    - Essa camada analisa os significados dos termos diante do uso comum de uma determinada língua. Quando dispostos em um texto, eles se tornam o vocabulário daquela linguagem, funcionando como um dicionário. Analisa a entrada de linhas de caracteres (como o código-fonte de um programa) e produz uma sequência de símbolos léxicos (*lexical tokens*).
- Pragmático
    - Interpreta os conceitos extraídos do texto, averiguando se o significado da análise semântica está correto e determinando significados que não estejam claros. Essa camada tenta buscar significados além das palavras, nas "entrelinhas".

Este trabalho focou na utilização das duas primeiras camadas apresentadas acima, no caso a sintática e morfológica, no contexto do projeto. Além disso, o processamento de linguagem natural possui diversas abordagens. Estas abordagens podem ser simbólicas (regras linguísticas estruturadas e precisas); estatísticas (modelos matemáticos induzem ao uso correto dos níveis de processamento); conexionistas (conecta aprendizado estatístico a outras teorias da representação do conhecimento); ou híbridas (combinação das três abordagens anteriores) (CHOWDHARY, 2020). No contexto deste trabalho, foi utilizada a abordagem simbólica.

2.5 Principais Técnicas de PLN

O PLN é uma subárea da inteligência artificial que é a aplicação da análise automatizada de técnicas de aprendizado de máquina para analisar textos básicos. PLN, independente da ferramenta utilizada, possui várias técnicas e funcionalidades que compõem o seu uso. As principais técnicas de PLN são:

- Segmentação de Sentenças
    - Essa componente separa o texto corrido bruto que é utilizado como entrada em frases, facilitando assim o seu entendimento e a sua visualização.
- *Tokenização*
    - Esse componente tem como entrada os elementos da entrada anterior ou até mesmo o texto corrido direto, separando os elementos do texto em *tokens,* um a um, e os salva em uma lista encadeada, podendo acessar cada elemento de maneira isolada.
- *Part of Speech Tagging (Pos - Tagger)*
    - Essa componente recebe como uma entrada uma lista de tokens e realiza uma análise morfológica das componentes, uma a uma, e retorna uma lista de duplas, onde cada elemento é composto do token em si com a sua classe gramatical.
- Reconhecedor de Entidades (*Chunking*)
    - Este componente trabalha da seguinte maneira, dada uma sentença S, obter as árvores de derivação para s de acordo com uma gramática G. Na análise sintática probabilística, pode-se estabelecer um ranking entre as possíveis análises, mostrando quão provável cada uma é ou, então, pode-se retornar apenas a análise mais provável para a sentença dada. Ou seja, reconhecer unidades estruturais de alto nível que permite compactar a descrição de uma sentença.

As técnicas apresentadas acima, na sequência mostrada, compõem o *pipeline* da maioria das ferramentas de PNL. Essa sequência é utilizada em várias ferramentas de PLN, como as que serão apresentadas na seção seguinte.

2.6 Análise de Ferramentas

Existem diversas técnicas e ferramentas para o PLN. Todas elas com suas vantagens e desvantagens e algumas e abordando alguns dos níveis citados acima (ANDRADE, 2018).

Nesse contexto, foram criados vários aplicativos para conversação humana com máquinas, como exemplo, os assistentes pessoais virtuais que estão presente em quase todos os sistemas operativos atualmente, tais como *Google Now* [1] , *Cortana* [2] e *Siri* [3] . Muitos deles são capazes de reconhecer sua fala e escrita, respondendo através da linguagem natural. Uma outra ferramenta é o *Powerset* [4] , um buscador que pretende revolucionar a forma de pesquisar na web fazendo uso da habilidade de interpretação de linguagem natural em mecanismo de busca. O objetivo da Powerset é trazer uma resposta clara ao contrário da busca do Google que traz uma lista de páginas, ou seja, o usuário poderá fazer uma pergunta clara e o mecanismo *Powerset* processa essa pergunta trazendo uma resposta objetiva. Em 2008, o *Powerset* foi comprado pela *Microsoft* por 100 milhões de dólares, mostrando o valor que PLN possui.

O trabalho realiza uma pesquisa qualitativa das ferramentas que auxiliam no desenvolvimento de softwares para PLN. Dessa forma, realizou-se uma revisão sistemática das ferramentas existentes e suas características. Foi feita uma seleção das ferramentas a serem analisadas e por fim, o resultado foi elaborado neste capítulo.

- Open NLP[5]
    - Biblioteca que consiste de várias ferramentas para aprendizagem de máquina para PLN. Escrita em JAVA e possui suporte para as principais tarefas de PLN.
    - Vantagens:
        - Solução em código aberto;
        - A própria empresa dispõe os materiais de apoio para o uso da biblioteca;
        - Realiza todas as principais funções PLN.

[1] https://www.google.com/intl/pt-BR/landing/now/
[2] https://developer.microsoft.com/pt-br/cortana
[3] http://www.apple.com/br/ios/siri/
[4] https://en.wikipedia.org/wiki/Powerset_(company)
[5] https://opennlp.apache.org/.

- Desvantagens:
  - Não contém suporte a língua portuguesa.

- Natural Language Toolkit - NLTK[6]
  - É uma plataforma para criação de programas de linguagem natural escrito na linguagem de programação *Python*. Tem suporte para tarefas tokenization, decorrente, marcação entre outros.
  - Vantagens:
    - Suporte a língua portuguesa (para algumas funcionalidades somente);
    - Solução em código aberto;
    - Manual de uso da ferramenta bem completo e elaborado.
  - Desvantagens
    - Funções como o *Pos-Tagger* somente para a língua inglesa;
    - Sem receber atualizações já a um certo tempo (última em 09 de abril de 2016);
    - Não possui suporte nativo à língua portuguesa.
- Stanford CoreNLP[7]
  - Stanford CoreNLP é um software que contém várias ferramentas para análise de linguagem natural, escrito em Java;
  - Vantagens:
    - Solução em código aberto.
  - Desvantagens:
    - Não possui material de apoio de utilização da plataforma;
    - Não possui fórum de debate sobre a plataforma;
    - Não a suporte para reconhecimento da língua portuguesa;
    - Não Realizar das principais funções PLN.

- UIMA[8]
  - A biblioteca apache UIMA é um framework onde pode ser projetado um sistema de software para análise de informações não estruturadas, tais como texto simples e identificador de entidades;

[6] http://www.NLTK.org/
[7] https://stanfordnlp.github.io/CoreNLP/
[8] https://uima.apache.org/

- Tem suporte de tarefas como detecção de entidade, detecção de relações, extração de tokens, lematização e etiquetagem morfossintática. Inicialmente foi desenvolvida pela IBM (IBM,2016), e agora pertence ao Apache Project;
- Vantagens:
  - Capaz de analisar grandes volumes de informações;
  - Possui material de apoio da própria ferramenta;
  - Realiza as principais funções da PLN.
- Desvantagens:
  - Não possui fórum de discussão;
  - Principais funções não funcionam para a língua portuguesa;
  - Não possui solução em código aberto.

2.7 Natural Language Toolkit - NLTK

Como apresentado na seção anterior, o *Natural Language Toolkit* (NLTK) é uma plataforma para criação de programas de linguagem natural escrito na linguagem de programação *Python*. Ele possui suporte as principais atividades de PLN e por ser escrita em *Python,* possui uma curva de aprendizado relativamente curta (LOPER, 2002). As principais funções utilizando o NLTK são apresentadas como segue:

- Segmentação de Sentenças
  - ```
    sentences = sentence_tokenizer.tokenize(my_string)
    print(sentences)
    ```
  - Utilizando a função *tokenize* e passando como entrada um texto, ela retorna todo o texto dividido em sentenças.
- *Tokenização*
  - ```
    tokens = word_tokenize(my_string)
    print(tokens)
    ```
  - Utilizando a função *word_tokenize* e passando como entrada um texto, ela retorna todo o texto dividido em tokens.
- *Part of Speech Tagging (Pos - Tagger)*
  - ```
    classes = nltk.pos_tag(tokens)
    print(classes)
    ```
    ```

    - Utilizando a função *pos_tag* e passando como entrada a lista de tokens, ela retorna lista de duplas, em que cada elemento dessa lista é composto pela palavra com sua classe gramatical.
- Reconhecedor de Entidades (*Chunking*)
    -
```
cp = nltk.RegexpParser(gramatica)
result = cp.parse(my_string)
```
    - Utilizando a função *RegexpParser* para o tratamento da gramática e a função *parse*, para a geração das árvores, é possível gerar as árvores de derivação para a sentença de entrada.

Para o contexto deste trabalho, foi utilizado o *Python* com o NLTK. O *Python* foi a escolha de linguagem de programação por ter uma curva de aprendizado menor, e por ser a linguagem que possui melhor suporte para as funcionalidades relacionadas a aprendizado de máquina e inteligência artificial. Além disso, o *Python* oferece suporte ao modelo orientado a objeto, permitindo uma melhor estruturação do código e por ser fácil de entender, mesmo para pessoas leigas no contexto do programa.

O NLTK foi escolhido como ferramenta para processamento de linguagem natural por ser uma biblioteca feita para a linguagem *Python* e por conseguir realizar todas as principais atividades do PLN de maneira simples. Os módulos do NLTK são bem divididos, além de ter uma excelente documentação (no próprio site da ferramenta, fóruns, repositórios), o que facilita seu aprendizado. Ele não possui suporte nativo para a língua portuguesa, mas com o uso de uma gramática definida sobre os modelos da língua, é possível trabalhar com ele em todas as linguagens.

Capítulo 3

Trabalhos Relacionados

Após a definição do escopo do trabalho, foi feito um estudo sobre ferramentas, tecnológicas ou projetos que se encaixassem no contexto deste TCC. Foram feitas buscas no portal de periódico e artigos acadêmicos do *Google Scholar* com as seguintes *strings* de busca:

- *Natural Language Processes ;*
- *Requirement Engineer ;*
- NLTK ;
- *Natural Language Processes for Requirement Evaluation ;*

A partir da busca por trabalhos relacionados com as *strings* acima, foram encontradas algumas ferramentas que se encaixavam no cenário que este trabalho estava inserido. O procedimento utilizado para identificar os trabalhos relacionados foram a data do artigo, os problemas atrelados a requisitos que eles abordavam, além das técnicas utilizadas. Além disso, os aspectos de implementação da ferramenta, como linguagem utilizada e interações com o usuário. Foi feita uma análise das mesmas, além de apresentar pontos positivos e negativos de cada trabalho.

Huertas, Juárez-Ramírez e Reyes propõe uma solução que consiste no desenvolvimento de uma ferramenta de PLN chamada NLARE (HUERTAS et al, 2012), que usa um conjunto de regras definidas, bem como expressões para identificar problemas nos documentos de especificação de requisitos. Isso é feito automaticamente e os resultados são utilizados como uma maneira de dar suporte para o engenheiro de requisitos tentar reduzir os problemas em potencial que seriam levados para os próximos estágios do desenvolvimento. Além disso, a ferramenta apresenta um mecanismo de detecção de incompletude baseado em regras e modelos, abordagem que foi utilizada no contexto deste TCC. O sistema feito pelos autores utiliza um dicionário de dados local, porém este não é especificado no artigo. Ele aborda os problemas referentes a ambiguidade e atomicidade dos requisitos.

A ferramenta, assim como a feita no contexto deste TCC, foi feita utilizando a biblioteca NLTK, do *Python*, mais especificamente as funções de *tokenização* e a verificação de palavras. O artigo não apresenta nenhum mecanismo de interação do usuário da ferramenta. Ele trabalha especificamente com os problemas de atomicidade, ambiguidade e completude de requisitos, e no final mostra um experimento realizado, porém os resultados são relativamente baixos, com sua ferramenta tendo sucesso em apenas 56% dos casos.

Sven J Korner, em seu trabalho *RESI - A Natural Language Specification Improver (KORNER, 2009),* apresenta sua ferramenta que utiliza ontologias para melhorar o processo de engenharia de requisitos e tem como principal objetivo servir como complemento a outras ferramentas de análise de requisitos, lidando com o processamento linguístico de informação textual. Este artigo apresenta uma abordagem semântica ao dicionário utilizado, apresentando um modelo baseado em ontologias estáticas para definição e modelagem do dicionário de dados utilizado na ferramenta, abordagem que foi utilizada no contexto deste TCC.

A ferramenta usa um dicionário de dados local, que não é apresentado no artigo, e sua principal funcionalidade é criar de maneira automática requisitos a partir de documentos de especificação de requisitos. Ele aborda problemas como a normalização textual e detecção de ambiguidade e usa a linguagem JAVA juntamente com uma API para processamento de linguagem natural. Ele usa as técnicas de *tokenização* e de verificação de palavras como mecanismos de PLN, além de utilizar algoritmos de grafos e de ontologias. O artigo apresenta algumas telas da ferramenta como parte de interação com o usuário.

Já Ishrak Hussain, apresenta uma abordagem diferente em seu trabalho: *Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier (HUSSAIN, 2007)*. Ele propõe uma ferramenta que fornece uma assistência automatizada para avaliar a qualidade dos requisitos textuais, através da utilização de um classificador baseado em árvore de decisões, equipado com mecanismo de processamento de linguagem natural.

O objetivo é aplicar as técnicas de classificação de texto para construir um sistema que realize a detecção automática de ambiguidade nos documentos de especificação de requisitos com base nos indicadores de qualidade definidos no modelo proposto neste artigo. A abordagem utilizando árvores de decisão se apresentou muito boa quando a complexidade do que vai ser analisado é grande, ou seja, há muitos critérios a serem explorados.

A ferramenta proposta por Ishrak Hussain também apresenta uma base de dados local, que não é apresentada no artigo. O foco da análise é na qualidade dos requisitos e para isso ele propõe uma ferramenta utilizando JAVA desktop com uma API para processamento de linguagem natural. É utilizado mecanismos de *tokenização* com verificação de palavras por

meio do PLN e árvores de decisão para a classificação textual. Ele apresenta algumas telas da aplicação com partes do sistema, para uma melhor interação com o usuário.

O artigo *Toward a Text Classification System for the Quality Assessment of Software Requirements Written in Natural Language (ORMANDJIEVA, 2007),* de Olga Ormandjieva, aborda o problema da avaliação automática da qualidade dos textos de requisitos, com utilização da classificação de textos do PLN. O artigo propõe um modelo de qualidade para o texto de requisitos e um sistema de classificação de texto para automatizar o processo de avaliação da qualidade. Este artigo aponta diversos problemas relacionados a documentos de software, e apresenta uma seção muito bem detalhada sobre eles, detalhando cada um dos problemas, entre eles ambiguidade e incompletude, que são os que estão sendo analisados no contexto deste TCC.

Ele utiliza um mecanismo para geração automática dos dicionários, utilizando técnicas de aprendizado de máquina para o mesmo. Tem como principal objetivo a avaliação automática de documentos de elicitação de requisitos. O artigo não apresenta como a ferramenta foi modelada, porém apresenta que foi utilizado mecanismos de *tokenização* e verificação de palavras, além de árvores de decisão para análise.

Oliveira (2019), em sua dissertação de mestrado, desenvolveu uma ferramenta denominada *Requirements Analysis Tool (RANT)*. O sistema utiliza várias técnicas de PLN como *Sentence Splitting, Tokenization e Part-of-Speech tagging*, para aferir a qualidade dos requisitos através da produção de uma análise que avalia a estrutura e a potencial ambiguidade que possa existir nos requisitos.

Este trabalho, além de apresentar uma aplicação para análise de requisitos, demonstra um guia sobre como projetar uma ferramenta para análise de requisitos, e apresenta diversos modelos e pseudocódigos, dando uma visão bem completa sobre este tipo de ferramenta. Ele ainda apresenta uma comparação sobre diversas tecnologias de processamento de linguagem natural, incluindo o NLTK, que é utilizado neste TCC.

A ferramenta faz uso de *plugins* para a validação automática de requisitos e avalia vários aspectos de qualidade dos requisitos, como completude, corretude, clareza, consistência e verificabilidade de maneira automática. Ele não apresenta aspectos de implementação, mas apresenta as técnicas utilizadas, como *tokenização* e verificação de palavras. O trabalho exibe algumas telas para validar a interação com o usuário além de uma arquitetura bem definida.

Imran Sarwar Bajwa, em seu trabalho *Resolving Syntactic Ambiguities in Natural Language Specification of Constraints - NL2OCL (BAJWA, 2012),* destaca os casos identificados de ambiguidades sintáticas e também apresenta uma ferramenta para automaticamente resolver as ambiguidades sintáticas identificadas.

O artigo não apresenta qual mecanismo de dicionário é utilizado. Ele aborda o problema da detecção e remoção de ambiguidade sintática em textos de elicitação de requisitos além de apresentar diversos pseudocódigos com vários exemplos, dando uma visão bem prática de como uma ferramenta de análise de requisitos deve se comportar. Além disso, ele expõe um capítulo inteiro e bem detalhado sobre o experimento no qual ele aplicou a sua ferramenta.

Ele resolve o problema da ambiguidade utilizando ferramentas como ontologias, além de *tokenização* e verificação de palavras. O artigo apresenta alguns pseudo-códigos além de modelos da arquitetura da ferramenta. O mecanismo de interação do usuário é somente alguns exemplos de saídas no terminal da IDE utilizada.

Ayan Nigam propõe em seu trabalho *Tool for Automatic Discovery of Ambiguity in Requirements (NIGAM, 2012),* uma ferramenta que pode revisar rapidamente os requisitos e identificar palavras ambíguas, além de fornecer possíveis fontes de interpretação errada. Atualmente a ferramenta suporta identificação de ambiguidades lexicais, sintáticas e semânticas, destacando palavras ambíguas e fornecendo um gráfico para avaliar a exatidão da análise.

O artigo expõe um estudo bem amplo sobre o problema da ambiguidade, com diversos tipos da mesma. Além disso, ele apresenta uma abordagem baseada em regras e modelos e modela a ferramenta para analisar os diversos tipos de ambiguidade e a detecção delas. Ele apresenta uma arquitetura bem definida, assim como um dicionário completo validado para a detecção de ambiguidade em requisitos.

Este trabalho apresenta um dicionário de dados local e estático, que é apresentado no artigo. O foco desta ferramenta é identificar ambiguidade e para isso propõe uma aplicação feita em JAVA, utilizando técnicas como *tokenização* e verificação de palavras, além de possuir telas apresentadas no artigo juntamente com a arquitetura do sistema.

Todos estes trabalhos apresentados foram catalogados e plotados na Tabela 2. Os parâmetros analisados na tabela são: como a ferramenta trabalhou com seu dicionário de dados e qual a abordagem utilizada, quais problemas relacionados a requisitos ele está disposto a analisar, qual a linguagem de programação, padrões utilizados, quais técnicas de PLN foram utilizadas e tecnologias apresentadas no desenvolvimento da ferramenta, além da

disponibilidade da ferramenta. Ele também apresenta quais mecanismos de interação com o usuário ele possui, se são telas ou *prints* de terminais, por exemplo.

Tabela 2: Comparativo de Trabalhos Relacionados

Artigos	**Dicionário de dados**	**Contexto Específico ou Geral**	**Problema a serem Resolvido**	**Linguagem ou padrões utilizados**	**Técnica de NPL Utilizadas Tecnologias**	**Disponibilidade da ferramenta**	**Interação com o usuário**
RESI - A Natural Language Specification Improver	Dicionário de dados local, mas não especificado no artigo	Criação automática de requisitos a partir de documentos de especificação de requisitos	Ambiguidade, Normalização	Java Desktop com api para PLN	*Tokenização,* Verificação de Palavras, Ontologias, Grafos	Repositório fechado, sem disponibilidade aberta	Telas com interação por parte do usuário
Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier	Dicionário de dados local, mas não especificado no artigo	Avaliação automática de documentos de avaliação de requisitos	Qualidade	Java Desktop com API para PLN	*Tokenização,* Verificação de Palavras, Árvore de decisão	Repositório fechado, sem disponibilidade aberta	Telas com interação por parte do usuário
Toward a Text Classification System for the Quality Assessment of Software Requirements Written in Natural Language	Dicionário de dados alterado em arquivos, puxado pela aplicação	Avaliação automática de documentos de avaliação de requisitos	Não Especificado	Não Especificado	*Tokenização,* Verificação de Palavras, Árvore de decisão, *Pos - Tagger*	Ferramenta nem é apresentada no artigo	Nenhuma
Automated Requirements Analysis using Natural Language Processing - RANT	Não utiliza dicionário de dados, mas sim ferramentas que validam os requisitos (*Plugins*)	Avalia vários aspectos de qualidade de requisitos e avaliação automática de documentos de avaliação de requisitos	Completude, corretude, clareza, consistência, verificabilidade	Não especificado	*Tokenização,* Verificação de Palavras, Árvore de decisão	Possui telas apresentadas no artigo, e a arquitetura	Somente exemplos das saídas em terminal

Artigos	**Dicionário de dados**	**Contexto Específico ou Geral**	**Problema a serem Resolvido**	**Linguagem ou padrões utilizados**	**Técnica de NPL Utilizadas Tecnologias**	**Disponibilidade da ferramenta**	**Interação com o usuário**
Resolving Syntactic Ambiguities in Natural Language Specification of Constraints -NL2OCL	Não apresentado no artigo	Detecção e remoção de ambiguidade sintática em textos de elicitação de requisitos	Ambiguidade	Não especificado	Ontologias, *Tokenização,* Verificação de Palavras	Apresenta a arquitetura e alguns pseudo-códigos	Somente exemplos das saídas em terminal
Tool for Automatic Discovery of Ambiguity in Requirements	Dicionário de dados local, e especificado no artigo	Detecção automática de ambiguidade de requisitos	Ambiguidade	Java Desktop com API para PLN	*Tokenização,* Verificação de Palavras, *Pos - Tagger*	Possui telas apresentadas no artigo, e a arquitetura	Somente exemplos das saídas em terminal
NLARE, A Natural Language Processing Tool for Automatic Requirements Evaluation	Dicionário de dados local, mas não especificado no artigo	Avaliação automática de documentos de avaliação de requisitos	Ambiguidade, Atomicidade	*Python* com NLTK	*Tokenização,* Verificação de Palavras, *Pos - Tagger*	Pedido Pessoal	Nenhuma

Com essas informações em mãos, foi possível analisar a ferramenta produzida neste TCC. A linguagem utilizada foi o *Python*, com o NLTK como tecnologia para processamento de linguagem natural. Como citado no capítulo 2, ambos foram escolhidos por terem uma curva de aprendizado simples e por atenderem a todos os pré-requisitos estabelecidos previamente.

Os problemas selecionados para abordar foram a ambiguidade e a incompletude, por serem problemas com grande relevância no contexto de documentos de requisitos, e como foi apresentado acima, ainda são problemas bastante explorados. Foi escolhido o contexto da avaliação automática de documentos de especificação de requisitos, pois foi visto que o problema de análise de requisitos ainda tem muito a ser investigado, de maneiras diferentes, como apresentado na tabela 2.

Foram utilizadas as principais técnicas de processamento de linguagem natural, como a *tokenização* e o *pos-tagger*. Todos os exemplos da utilização da ferramenta serão apresentados via terminal, além de apresentar diversos trechos de códigos, isso será apresentado no capítulo seguinte. Foi montada uma arquitetura com divisões em módulos para uma melhor visualização da execução do sistema.

Capítulo 4

Proposta de Solução

4.1 Introdução

Como foi apresentado nos capítulos anteriores, o objetivo deste trabalho é auxiliar o processo de elicitação de requisitos, mais especificamente a etapa de análise de requisitos. Vários problemas podem vir a gerar requisitos considerados falhos, entre os problemas estão, por exemplo, incompletos, não gerenciados, não especificados, imprecisos, ambíguos (SHAH, 2011). No contexto deste trabalho foram analisados os problemas de ambiguidade e de incompletude.

Além disso, este trabalho faz uma análise a respeito do contexto de cidades inteligentes, validando se o requisito está inserido ou não dentro deste cenário e se ele é considerado completo dentro deste contexto mais específico, avaliando a necessidade de completar o requisito com mais informações. Para isso foi gerada uma ferramenta, capaz de realizar essa análise dos requisitos de maneira automática.

4.2 Problemas a serem solucionados

Um dos principais problemas de pesquisa em aberto em engenharia de requisitos está em resolver ambiguidade. Ambiguidade é definida como "uma declaração tendo mais de um significado" (NIGAM, 2012). Uma ambiguidade pode ser léxica, sintática, ou de sintaxe.

- **Ambiguidade léxica:** - A ambiguidade lexical ocorre quando uma palavra tem vários significados. Ela também ocorre quando duas palavras de origem diferente vêm para a mesma grafia e pronúncia;
- **Ambiguidade sintática**: - Ambiguidade sintática, também chamada ambiguidade estrutural, ocorre quando uma determinada sequência de palavras pode ter mais de uma estrutura gramatical, e cada um tem um significado diferente. Por exemplo, quando a frase permite diferentes árvores de análise;

- **Ambiguidade de sintaxe**: - Esta ambigüidade é particular para a ferramenta desenvolvida. Este erro ocorre se uma frase não terminar com um ponto (.), segundo se o agente do usuário não for especificado a frase, então é considerado um erro de sintaxe.

Este trabalho focou, em um primeiro momento, na análise e detecção da ambiguidade do tipo lexical já que pelo que foi apresentado nos trabalhos e ferramentas relacionadas, era onde se concentravam a maioria dos erros do tipo de ambiguidade. Para isso, foi desenvolvido um dicionário de palavras ambíguas, gerado a partir da análise de outras ferramentas que utilizavam a mesma abordagem. O conteúdo do dicionário pode ser visto na Tabela 3.

Tabela 3: Lista de Palavras Ambíguas por Classificação

Always, Every, None, Never	Essas palavras denotam algo como certo ou absoluto, e seu uso deve ser sempre utilizado com muita cautela a se escrever qualquer tipo de texto. Em requisitos, o uso dessas palavras pode generalizar algo de maneira indevida.
Certainly, Cleary, Therefore, Obviously	Essas palavras tendem a persuadir o acatamento de algo como algo certo, gerando frases “rasas”.
Good, Fast, Small, Cheap, Stable and Efficient, Easy, Sufficient, Normal, Adequated, Robust	Esse conjunto de palavras pode ser classificado como não quantificável. Se eles aparecerem em uma especificação, eles devem ser muito bem definidos, para explicar exatamente o seu contexto.
Some, Sometime, Often, Usually, Ordinarily, Customarily, Most, Mosty	Essas palavras apresentam um sentido muito vago, e é impossível de testar um recurso que opera com essas palavras dependendo da situação.
Handled, Processed, Rejected, Skipped, Eliminated, Effective	Esses termos podem vir a omitir uma quantidade enorme de funcionalidades que devem ser especificadas.

Outro problema que foi trabalhado pela ferramenta é a questão da incompletude, e para isso foram utilizadas algumas regras definidas pela língua inglesa para identificar e trabalhar com o problema da incompletude. São elas:

- ***Sentence Fragment***: Ocorre quando a sentença não possui um objeto direto relacionado a uma ação (verbo). Um *"Sentence Fragment"* é uma frase que não tem sujeito ou não tem verbo ou não é uma ideia completa (uma razão pode ser porque não tem um objeto direto se o verbo precisar de um);
- ***Missing Subject***: Esse é um problema sintático no qual ocorre a falta de um sujeito de referência para uma determinada ação (verbo e suas variações formais). Esse problema pode ser descrito como: (vazio) faz algo;
- ***Missing Verb Mistake***: Esse é um problema sintático no qual ocorre a falta de um verbo na frase, gerando assim uma frase sem "ação". O sujeito daquela frase fica sem uma ação para acompanhar, quebrando a estrutura sintática da frase;
- ***Dummy Subjects***: Existem sentenças as quais o sujeito não "existe", e são utilizados sujeitos de maneira vaga, podendo gerar dúvidas sobre seu verdadeiro significado. As cláusulas inglesas que não são imperativas devem ter um assunto. Às vezes, precisamos usar um sujeito "fictício" ou "vazio" ou "artificial" quando não há sujeito anexado ao verbo, e onde o sujeito real está em outro lugar na oração. *IT* e *THERE* são os *Dummy Subjects* usados na língua inglesa.

Sobre o problema relacionado ao cenário de cidades inteligentes, foi utilizada uma ontologia para mapear o contexto daquele requisito e validar se ele de fato está dentro do contexto de cidades inteligentes. Paul Buitelaar define ontologia como:

> "A Ontologia é classificada na filosofia como o ramo geral da metafísica e é comum que os termos ontologia e metafísica sejam utilizados como sinônimos, embora o primeiro esteja inserido no segundo. Em Ciência da Computação, ontologias são aplicadas à modelagem, tanto em sistemas baseados em bancos de dados quanto em sistemas de representação do conhecimento. Em Representação do Conhecimento, Inteligência Artificial, o termo ontologia tem sido usado desde o século passado para se referir a uma estrutura de entidades representada por termos de um vocabulário lógico. Uma ontologia é uma especificação explícita de uma conceitualização, e define as classes, as instâncias e as relações de um determinado modelo, facilitando assim a manipulação desses dados" (Buitelaar, 2008).

Mapear uma cidade utilizando uma ontologia e utilizar a mesma como base de dados para cidades inteligentes se apresentou como uma abordagem promissora, pois com uma

ontologia é possível mapear todo o cenário de um ambiente, desde sensores e atuadores, até ambientes mais complexos como cidades inteligentes.

Ontologias também permitem inserir instâncias desses objetos, como um sensor específico de temperatura que está inserido em um ambiente específico de um prédio dentro da cidade. Além disso, permite uma visualização mais completa e de maneira mais simples.

A ontologia escolhida para este trabalho foi a *M3 - Ontology*[9], uma ontologia para internet das coisas e cidades inteligentes. Ela permite projetar cenários de cidades inteligentes com uma grande riqueza de detalhes, e define desde sensores até campos de interesse, como saúde e agorpecuaria. Ela apresenta mais de 30 tipos de sensores, medidas, unidades e cerca de 10 domínios diferentes. Uma representação em alto nível dela pode ser vista na Figura 2.

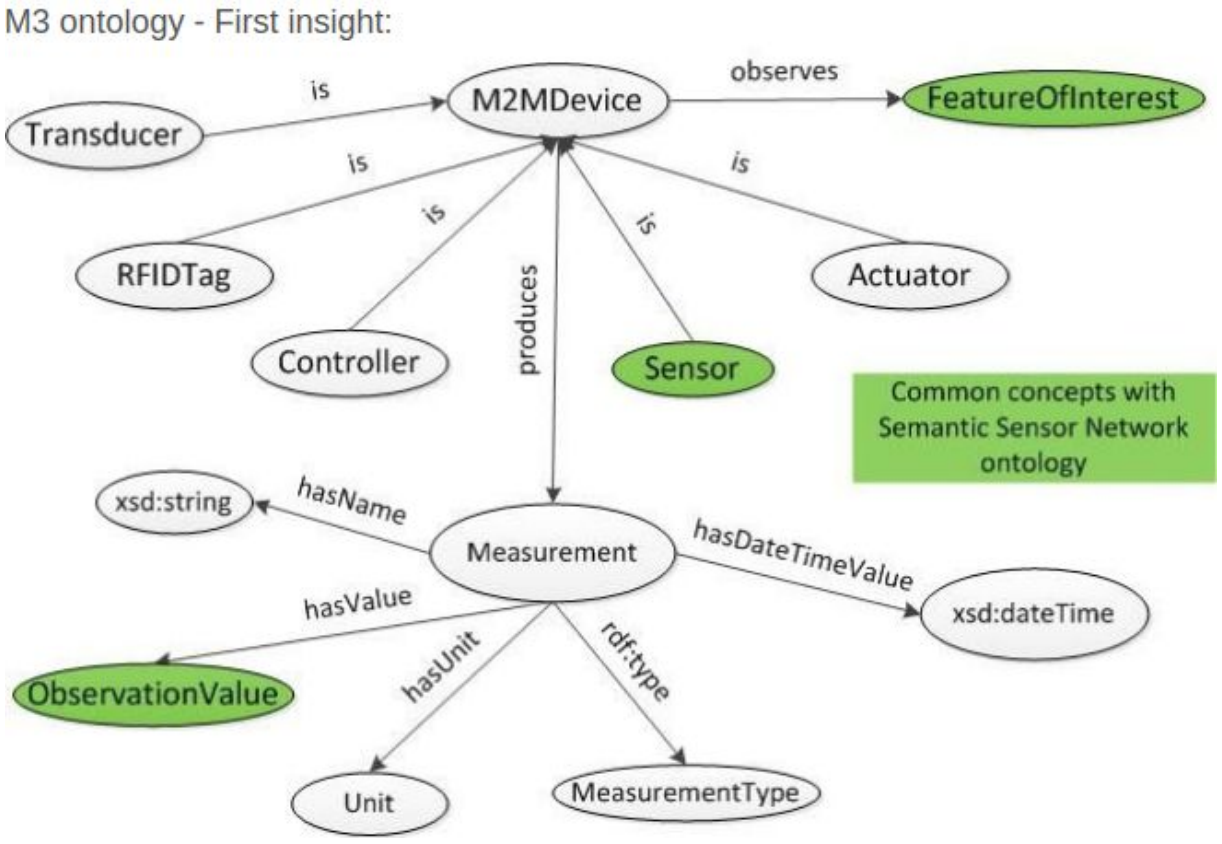

Figura 2: Ontologia M3

[9] http://sensormeasurement.appspot.com/?p=m3

4.3 Arquitetura da Solução e Aspectos de Implementação

O sistema foi definido e separado por módulos, onde cada um possui suas respectivas entradas e saídas e suas especificações próprias. Cada problema apresentado na seção anterior foi trabalhado em um módulo separadamente. Para uma melhor visualização das componentes e do funcionamento do sistema foi modelada uma arquitetura da ferramenta. Ela pode ser vista na figura 4, apresentada abaixo.

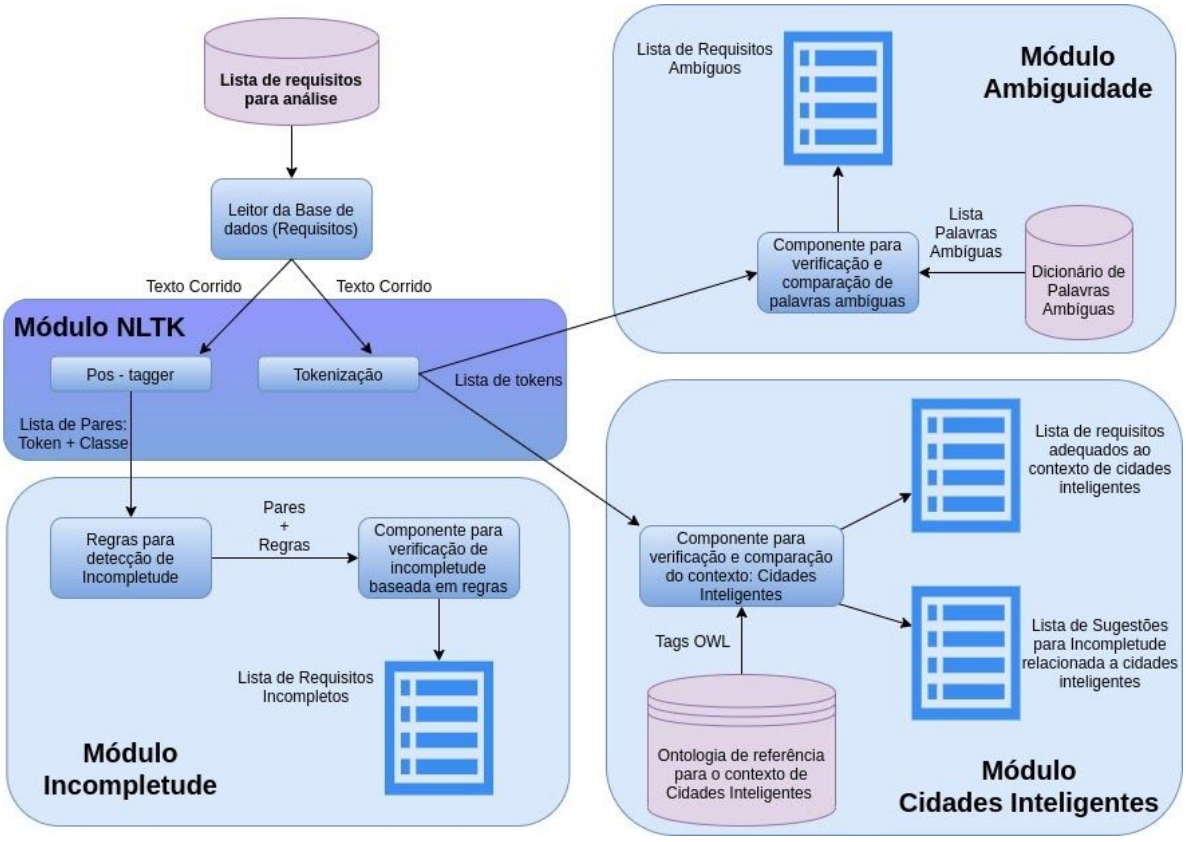

Figura 4: Arquitetura

O primeiro ponto da arquitetura é o leitor da base de dados, que irá ler a lista de requisitos para análise. Essa lista de requisitos pode ser salva como um arquivo e a leitura é feita utilizando os mecanismos de leitura de arquivos do *Python*. As outras componentes recebem os dados que saem deste módulo.

4.3.1 Módulo NLTK

O módulo NLTK recebe o texto corrido vindo da base de requisitos e faz e realiza duas operações, de maneira distintas. A primeira é a *Tokenização*, onde ele converte o texto corrido para uma lista de tokens, utilizando métodos do próprio NLTK. A saída desse componente pode ser vista na Figura 5.

```
['#', 'rq15', ':', 'management', 'of', 'sensor', 'and', 'actuator', 'data', ';']
```

Figura 5: Lista de Tokens

A outra componente é a que realiza o processo do *pos-tagger,* que recebe o texto corrido e faz a análise gramatical de cada componente da frase, gerando uma lista de duplas, em que cada elemento é composto de uma palavra com uma "etiqueta", correspondendo a sua classe gramatical. Essas etiquetas representam as classes gramaticais e são geradas pelo próprio NLTK. A lista de duplas pode ser vista na Figura 6.

```
[('#', '#'), ('rq15', 'NN'), (':', ':'), ('management', 'NN'), ('of', 'IN'), ('sensor', 'NN'), ('and',
 'CC'), ('actuator', 'NN'), ('data', 'NNS')]
```

Figura 6: Lista de Duplas

No caso do exemplo da Figura 6, o **NN** significa substantivo, **IN** preposição, **CC** conjunção coordenativa e o **NNS** substantivos no plural.

4.3.2 Módulo Incompletude

O módulo incompletude recebe a lista de duplas provenientes da componente do *pos tagger*, no módulo NLTK. Neste módulo estão as regras que foram apresentadas na seção 4.1, as regras de *Sentence Fragment, Missing Subject, Missing Verb Mistake e a Dummy Subjects.*

As regras juntamente com as duplas são analisadas e geram a saídas do módulo, que apresentam quais requisitos apresentam erros de incompletude ou não. A saída pode ser vista na Figura 7. Os requisitos considerados incompletos pela ferramenta são listados pela ferramenta com a cor vermelha.

Figura 7: Resultado do Módulo Incompletude

4.3.2 Módulo Ambiguidade

Esse módulo recebe como entrada a lista de tokens gerados no módulo NLTK. Essa lista de tokens é passada para a componente que irá realizar a análise de ambiguidade. Além disso, essa componente recebe a lista de palavras ambíguas, que foram apresentadas na seção 4.1.

Com as duas entradas, o módulo de ambiguidade realiza uma análise sobre os requisitos. Esta análise é feita baseada em comparações sobre a lista de requisitos com a lista de tokens, atrás de igualdade de elementos. A saída pode ser vista na Figura 8. Os requisitos considerados ambíguos pela ferramenta são listados pela ferramenta com a cor amarela.

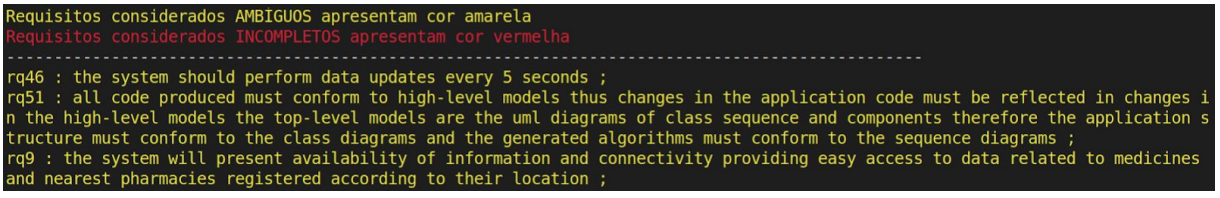

Figura 8: Resultado do Módulo Ambiguidade

4.3.2 Módulo Cidades Inteligente

Neste módulo são feitas as análises sobre o contexto de cidades inteligentes. É aqui onde será aplicada a ontologia de referência apresentada na seção 4.1, a M3. Nela estão catalogados vários elementos de cidades inteligentes, como sensores e atuadores.

A componente de verificação e comparação para cidades inteligentes recebe a lista de tokens oriundos do módulo NLTK, juntamente com as tags OWL 's vindas da ontologia.

Essas tags são similares ao modelo XML e são enviadas para a componente, que faz o tratamento nessas tags a fim de realizar a comparação delas.

Esse módulo realiza duas funções, a primeira é para detectar se um requisito está contigo dentro do contexto de cidades inteligentes, avaliando se ele contém algum elemento definido na ontologia de referência. Ele tem como saída a seguinte mensagem, apresentada na Figura 9, em verde.

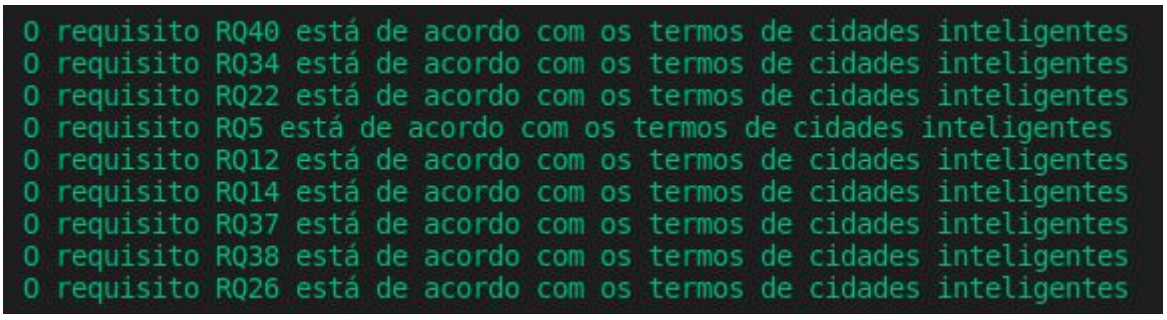

Figura 9: Resultado da Detecção do Contexto do Requisito

A segunda função é para verificar se o requisito está completo, no sentido de definir um contexto completo para um cenário de cidades inteligentes. Essa funcionalidade foi feita analisando palavras específicas no contexto do grafo definido pela ontologia de referência. Esse mecanismo foi implementado para a palavra sensor, e a sua saída pode ser apresentada abaixo, na Figura 10, em azul.

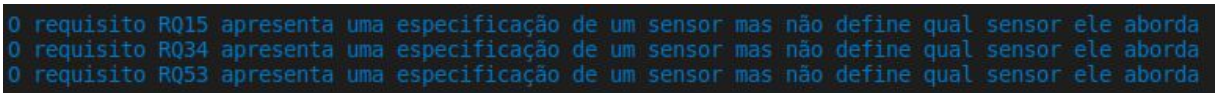

Figura 10: Resultado da Especificação da Palavra

4.4 Interação com o usuário

O sistema foi todo modelado para rodar via terminal, e a sua interação com o usuário se dá por meio de um entradas no mesmo, o usuário que irá selecionar como proceder. Ele deverá inserir o arquivo de teste de requisitos na pasta do projeto, e a ferramenta já irá

identificar ele. Após isso é só ir seguindo o pequeno menu elaborado para facilitar o entendimento. A visualização dos módulos de ambiguidade e incompletude trabalham juntos, como apresentado na figura 11. O módulo de cidade inteligente com suas duas funcionalidades trabalha de maneira isolada, e, por isso, possui uma opção de visualização própria, como é apresentado na figura 12. Requisitos que se encaixam em mais de um problema são caracterizados como em ambos os casos.

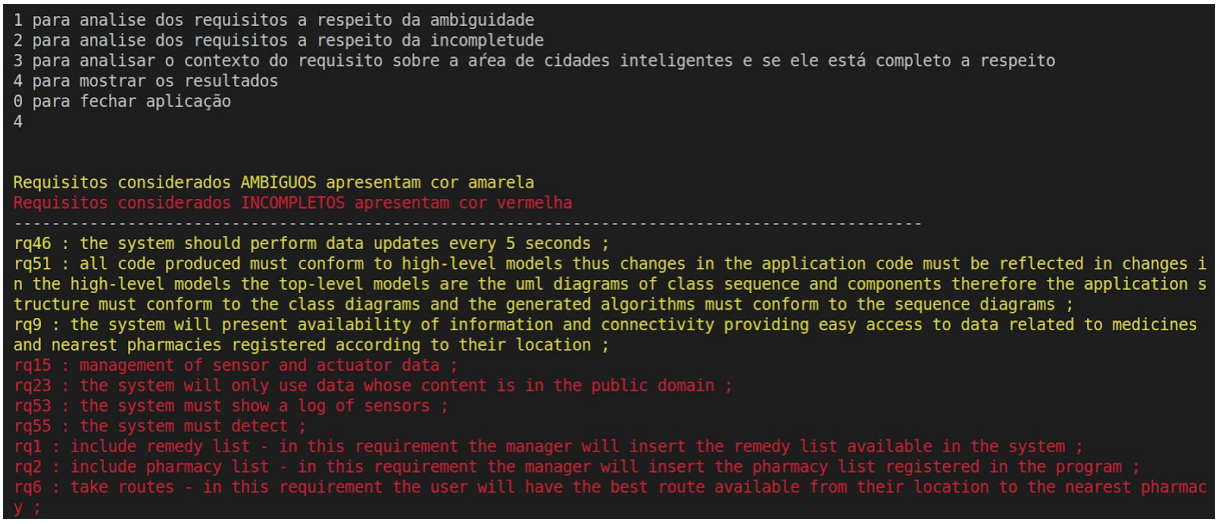

Figura 11: Visualização do Módulo Ambiguidade e Incompletude

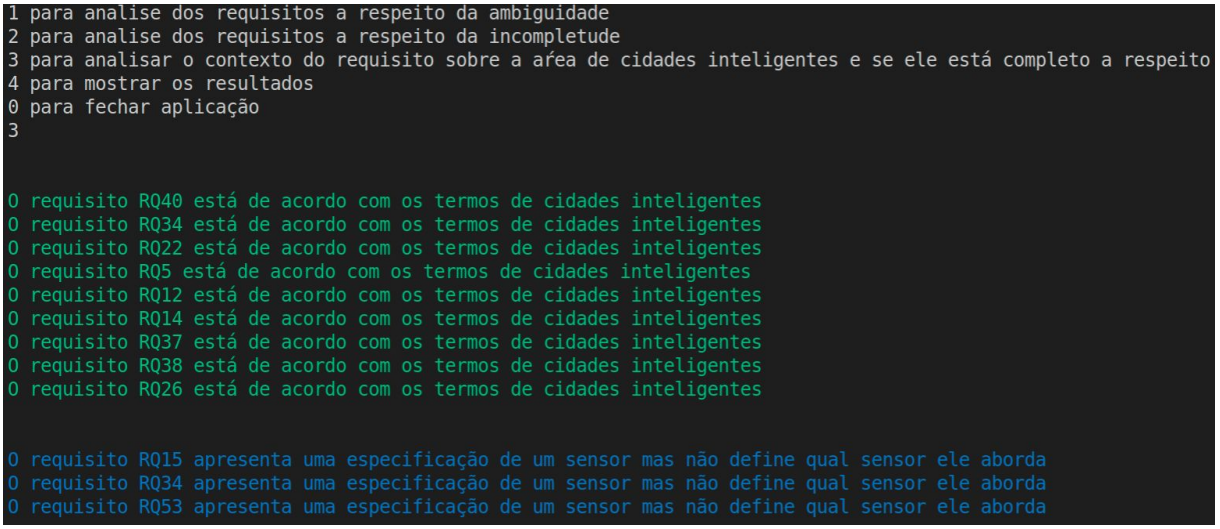

Figura 12: Visualização do Módulo Cidades Inteligente

4.5 Prova de Conceito

Para analisar a adequabilidade da ferramenta proposta ao seu propósito, foi realizada uma prova de conceito em cima da mesma. Na prova de conceito aplicada na ferramenta proposta, foram utilizados os requisitos definidos nos projetos da disciplina de Tópicos Avançados em Cidades Inteligentes, ministrada pelo programa de pós-graduação da Universidade Federal do Maranhão, nos períodos de 2018.2 e 2019.1. A listagem dos requisitos pode ser vista nos Apêndice de A a G.

Para a condução da prova de conceito, foram aplicadas à ferramenta os requisitos apresentados nos Apêndices de A a G. Os mesmos requisitos foram avaliados por dois especialistas da área de engenharia de software. Ao receber os requisitos, eles foram instruídos a classificá-los como ambíguos, incompletos ou dentro do contexto de cidades inteligentes, caso fosse necessário.

Foram avaliados, no total, 55 requisitos de oito projetos diferentes. A aplicação proposta detectou nove requisitos considerados ambíguos e nove requisitos considerados incompletos. Além disso, a ferramenta detectou que somente 12 dos requisitos apresentados, se lidos de maneira isolada, poderiam ser considerados dentro do contexto de cidades inteligentes, além de detectar e sinalizar a melhoria de três destes requisitos.

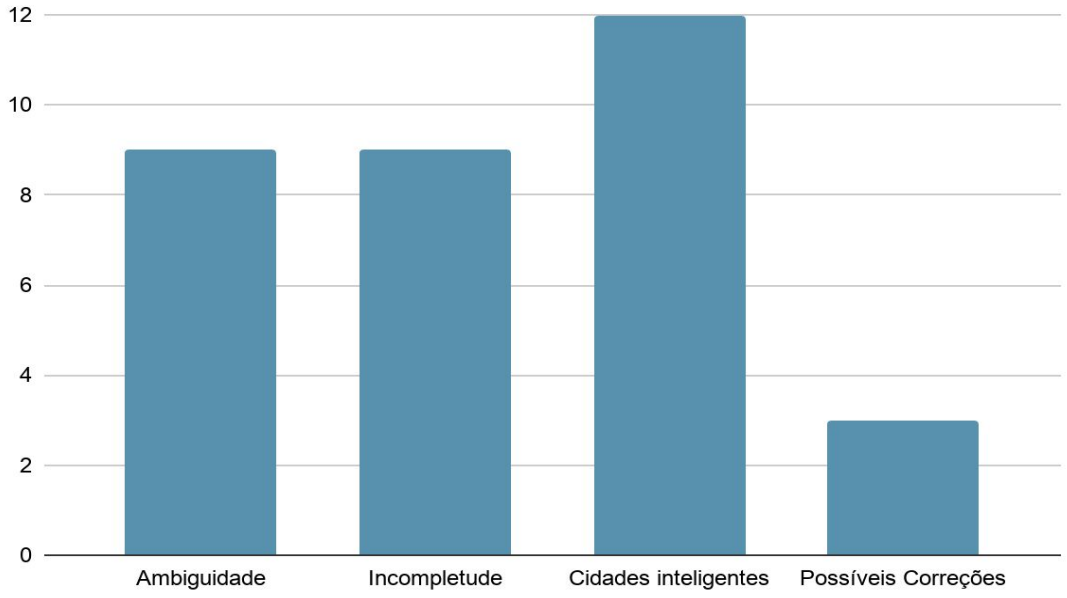

Figura 13: Gráfico de Erro por Tipo

O especialista A, na sua análise, detectou seis requisitos considerados ambíguos, 19 requisitos incompletos e 18 caracterizados dentro do contexto de cidades inteligentes. Já o especialista B detectou 15 requisitos considerados ambíguos, 23 incompletos e 23 dentro do contexto de cidades inteligentes.

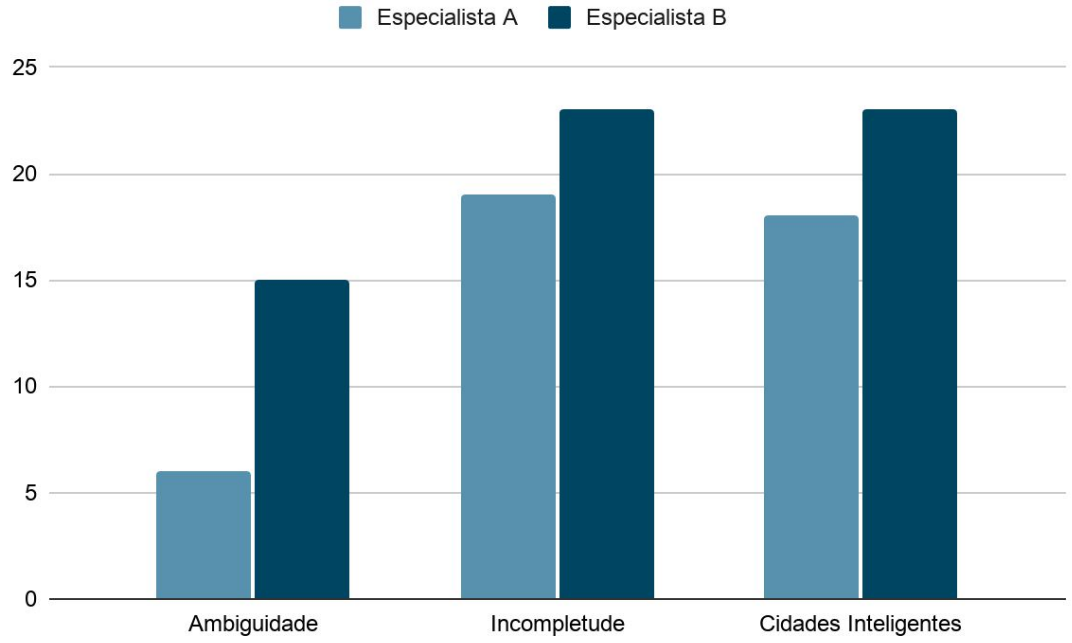

Figura 14: Gráfico de Erro por Tipo e por Especialista

A respeito da ambiguidade, dos seis requisitos apresentados pelo especialista A, quatro também foram encontrados pela ferramenta, que encontrou três erros a mais. Em relação ao especialista B, dos 15 requisitos apontados, sete também foram encontrados pela ferramenta, com uma taxa de acerto de 46%.

Sobre incompletude, dos 19 requisitos declarados incompletos pelo especialista A, cinco também foram apontados pela ferramenta, tendo assim uma taxa de acerto de 27%. Já com o especialista B apontou 23 requisitos como incompletos, desses a ferramenta encontrou nove, tendo uma taxa de acerto de 39%.

Na análise do contexto para cidades inteligentes, dos 18 requisitos apontados como no contexto para o especialista A, sete também foram detectados pela ferramenta, com taxa de acerto de 38%. Em relação ao especialista B apontou 23 requisitos considerados no contexto, e desses oito foram encontrados pela ferramenta, tendo taxa de aproveitamento de 34%. Todos os aproveitamentos foram apresentados na Tabela 4, para facilitar a visualização dos mesmos.

Tabela 4: Porcentagem de Acerto da Ferramenta

	Especialista A	Especialista B
Ambiguidade	66%	46%
Incompletude	27%	39%
Cidades Inteligentes	38%	34%

Os dados expostos ainda não servem para validar a ferramenta, mas servem para comprovar seu funcionamento. É visto que, somente a análise de dois especialistas não são suficientes para validar uma ferramenta, porém o principal objetivo desta prova de conceito era analisar o comportamento da ferramenta avaliando requisitos de sistemas reais e comparar seus resultados com algum especialista da área.

Os dados obtidos nessa prova de conceito serão fundamentais para a melhoria e a continuação da ferramenta no futuro. Os dados serviram para comprovar a eficácia de certas componentes e apontar possíveis melhorias em outras, com a ideia de continuar o desenvolvimento e gerando melhorias para a mesma.

Capítulo 5

Conclusão

A engenharia de requisitos é uma das etapas mais importantes do desenvolvimento de software, e é nela onde se pode encontrar a maioria dos erros durante a etapa de desenvolvimento. É nela também, onde o erro pode ser reparado, de maneira menos custosa, e sem prejudicar as demais etapas de desenvolvimento. Porém, essa análise dos requisitos de software, se feitas de maneira manual, é custosa e pode atrapalhar o processo de desenvolvimento.

A grande maioria dos documentos de especificação de software são escritos em linguagem natural, entre elas o inglês e o portugues. E por isso, combinar técnicas de processamento de linguagem natural, com a análise de requisitos de software se apresentou como uma abordagem promissora. Assim, um sistema automatizado com uma automatizada para resolver problemas como ambiguidade e incompletude de requisitos de software é completamente adequado.

Este trabalho teve como principal objetivo, contribuir com a análise de requisitos, utilizando abordagens de processamento de linguagem natural. Para isso foi feito um estudo sobre ferramentas e tecnologias que se encaixam no escopo do tema, e foi montado um referencial teórico para isso, comparando diversos trabalhos da literatura baseado em alguns parâmetros.

Com os trabalhos em mãos, foi proposta uma ferramenta que faria análises sobre ambiguidade, incompletude e sobre o contexto para qualquer conjunto de requisitos dado. Esta ferramenta foi dividida em módulos e componentes para facilitar o seu entendimento e a sua aplicação.

Na ferramenta proposta, foi aplicada uma prova de conceito, a fim de comprovar seu funcionamento. Os resultados da ferramenta se mostraram promissores, porém passíveis de melhorias e de alterações para deixá-la mais completa. As abordagens utilizadas para trabalhar com cada módulo podem ser melhoradas a fim de aumentar a taxa de acerto da ferramenta, e outras técnicas de avaliação devem ser aplicadas, ainda mais quando consideradas que houveram diferenças entre os resultados dos avaliadores.

Como trabalhos futuros, pretende-se melhorar o funcionamento do sistema, e avaliar as técnicas utilizadas, baseado no resultado da prova de conceito. Além disso, após as melhorias na ferramenta, aplicar experimentos com mais especialistas da área para validar o uso dela por engenheiros de software. Pretende-se ao fim destas melhorias, realizar uma publicação sobre a ferramenta.

Referências

1. Andrade, L. M. S., Barros, R. C., & dos Santos, M. A. B. *PROCESSAMENTO DE LINGUAGEM NATURAL (PLN): FERRAMENTAS E DESAFIOS*.
2. Arellano, A., Carney, E., & Austin, M. A. (2015, April). *Natural language processing of textual requirements.* In *The Tenth International Conference on Systems (ICONS 2015), Barcelona, Spain* (pp. 93-97).
3. Ayan Nigam, Neeraj Arya, Bhawna Nigam, Deepika Jain. "*Tool for automatic discovery of ambiguity in requirements*." *International Journal of Computer Science Issues (IJCSI)* 9.5 (2012): 350.
4. Bajwa, Imran Sarwar, Mark Lee, and Behzad Bordbar. "*Resolving syntactic ambiguities in natural language specification of constraints*." *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, Berlin, Heidelberg, 2012.
5. Buitelaar, Paul, and Philipp Cimiano, eds. *Ontology learning and population: bridging the gap between text and knowledge*. Vol. 167. Ios Press, 2008.
6. Chowdhary, K. R. *Natural language processing.* Fundamentals of Artificial Intelligence. Springer, New Delhi, 2020. 603-649.
7. HUERTAS, Carlos; JUÁREZ-RAMÍREZ, Reyes. *NLARE, a natural language processing tool for automatic requirements evaluation*. In: Proceedings of the CUBE International Information Technology Conference. 2012. p. 371-378.
8. Hussain, Ishrar, Olga Ormandjieva, and Leila Kosseim. "*Automatic quality assessment of SRS text by means of a decision-tree-based text classifier*." *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, 2007.
9. Ian Sommerville and Pete Sawyer. 1997. *Requirements Engineering: A Good Practice Guide (1st. ed.)*. John Wiley & Sons, Inc., USA.
10. KORNER, Sven J.; BRUMM, Torben. *Resi-a natural language specification improver.* In: 2009 IEEE International Conference on Semantic Computing. IEEE, 2009. p. 1-8
11. Loper, Edward, and Steven Bird. *"NLTK: the natural language toolkit."* arXiv preprint cs/0205028 (2002).
12. Mariana Oliveira. "*Automated Requirements Analysis using Natural Language Processing*" Tese de Mestrado Faculdade de Engenharia da Universidade do Porto

13. NAZIR, Farhana. *c.* In: International conference on information science and applications. Springer, Singapore, 2017. p. 485-493.
14. Ormandjieva, Olga, Ishrar Hussain, and Leila Kosseim. "*Toward a text classification system for the quality assessment of software requirements written in natural language*." *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*. 2007
15. Pfleeger, Shari Lawrence. *Engenharia de software: teoria e prática*. Prentice Hall, 2004.
16. Scacchi, Walt. "Process models in software engineering." *Encyclopedia of software engineering* (2002).
17. SHAH, Unnati S.; JINWALA, Devesh C. *Resolving ambiguities in natural language software requirements: a comprehensive survey.* ACM SIGSOFT Software Engineering Notes, v. 40, n. 5, p. 1-7, 2015.
18. Sommerville, Ian ; tradução Ivan Bosnic e Kalinka G. de O. Gonçalves ; revisão técnica Kechi Hirama / *Engenharia de Software* — 9. ed. — São Paulo : Pearson Prentice Hall, 2011.
19. UMBER, Ashfa; BAJWA, Imran Sarwar. *Minimizing ambiguity in natural language software requirements specification*. In: 2011 Sixth International Conference on Digital Information Management. IEEE, 2011. p. 102-107.

Lista de Apêndices

Apêndice A - Projeto EasyMed

RQ1 : Include Remedy List - In this requirement, the manager will insert the remedy list available in the system ;
RQ2 : Include Pharmacy List - In this requirement, the manager will insert the pharmacy list registered in the program ;
RQ3 : Consult Remedy in nearest pharmacies - In this requirement the user will have the possibility to consult a medication in the application ;
RQ4 : Send Warnings on Medicines - In this user case the user will receive notification of availability of medications at nearby pharmacies ;
RQ5 : Display traffic data, according to predefined rules ;
RQ6 : Take routes - In this requirement, the user will have the best route available from their location to the nearest pharmacy ;
RQ7 : The system will present a security policy that ensures the reliability of the information registered by users in the application ;
RQ8 : The system will present ease of use of its interface, providing the user with little effort to reach a goal ;
RQ9 : The system will present availability of information and connectivity, providing easy access to data related to medicines and nearest pharmacies registered according to their location ;

Apêndice B - Projeto Trânsito Inteligente

RQ10 : Collect sensor data ;
RQ11 : Analyze sensor information ;
RQ12 : Display traffic flow history graphically ;
RQ13 : Low response time for congestion detection ;
RQ14 : Low response time when the traffic light is activated ;
RQ15 : Management of sensor and actuator data ;
RQ16 : Dynamic change according to the context ;

Apêndice C - Projeto InterCrime

RQ17 : The system should display on a map markers for crimes that occurred in downtown Chicago in the years of 2016, 2017, 2018 ;
RQ18 : The system should display on a map the markers related to the crimes that occurred in downtown Chicago at a interval specified by the user ;
RQ19 : The system should display the number of crimes that have occurred in a given time interval specified by the user ;
RQ20 : The system should provide information about a specific crime to the user ;
RQ21 : The system should list crimes of a type specified by the user ;
RQ22 : The system should display a bar graph with the number of the main crimes that occurred in the years 2016, 2017 and 2018. Optionally, the user can specify the interval, in months, that he wishes to consult ;
RQ23 : The system will only use data whose content is in the public domain ;
RQ24 : The system will be entirely WEB, the HTML markup language and the Javascript scripting language will be used. The Leaflet API will be used to display the map. The data necessary to meet the demands of the system must be obtained through the InterSCity platform ;

Apêndice D - Projeto BikeShare

RQ25 : When launched, the application should open with the initial display, containing the developers names, UFMA and LSDI logo, application name and the button for the user to click and access the map ;
RQ26 : Bearing in mind that the system will use data from shared bikes in Paris ,France, and it will be developed for use in Brazil. In this specific case, points of the City of Paris will be correlated with the City University Dom Delgado - UFMA. The user location will be obtained automatically ;
RQ27 : The system should identify which bike station is closest to the user location, which has at least one free bike. There should be a marker, at the station location, in addition to a text box, containing the station name and the number of bikes available ;
RQ28 : The system must be connected to the internet to provide the necessary information to the end user ;
RQ29 : The system will be designed for the android platform ;

Apêndice E - Projeto Controle de Energia em um Campus Universitário

RQ30 : Configure the ContextNet Gateway IP and Port - As the system works based on the Client / Server paradigm it is necessary to previously configure the IP and Port of the gateway that will receive the data ;
RQ31 : Smart Meter needs to be connected and functional - Stores IP and Port using Android key value persistence mechanism ;
RQ32 : Set the UUID, from the intercity, of the resource ;
RQ33 : The system sends a request with the hello message to the gateway and the gateway is expected to send a response ;
RQ34 : The system reads the voltage, current and power information from the sensor ;
RQ35 : When a Processing Node receives an electrical consumption message, it must register it with InterSCity platform;
RQ36 : The user must log in to the monitoring system ;
RQ37 : The system graphically displays information on the energy consumption in KWh of each building ;
RQ38 : The system constantly monitors the energy consumption of each building and alerts users if the building experiences energy consumption ;
RQ39 : The system allows messages to be sent to people inside the building ;

Apêndice F - Projeto Monitoramento das Características das águas das praias de Chicago

RQ40 : The system should receive data from the Chicago Data Portal website api and show the user the water temperature of a Chicago beach at a certain date and time ;
RQ41 : The system should monitor the turbidity of the water through the API data ;
RQ42 : The system should monitor the height of the waves using the API data ;
RQ43 : The application will have a map showing the location of the beaches through markers ;
RQ44 : The application will have a map showing the location of the beaches through markers. When selecting a marker, the user will obtain the beach information ;
RQ45 : The system must consume data from a REST API in JSON format ;
RQ46 : The system should perform data updates every 5 seconds ;
RQ47 : The system should be able to add new monitoring points dynamically ;

Apêndice G - Projeto Smart Parking

RQ48 : The occupancy rate for parking spaces will be displayed on the web page provided in percentage via a pie chart ;
RQ49 : The service will be saving the records of the highest and lowest historical occupancy rate of each month. These records can be consulted at any time and the data for the highest occupancy rate and the lowest occupancy rate will be displayed on the screen ;
RQ50 : The system will use the usability guide provided by the federal government of Brazil, highlighting the 7 guidelines addressed by the government ;
RQ51 : All code produced must conform to high-level models. Thus, changes in the application code must be reflected in changes in the high-level models. The top-level models are the UML diagrams of class, sequence and components. Therefore, the application structure must conform to the class diagrams, and the generated algorithms must conform to the sequence diagrams ;