

HERMES: UM MODELO DE INTERFACE DE LINGUAGEM NATURAL PARA TRANSFORMAÇÃO EM SOFTWARE

Michael William Maciel Chagas¹

Kleinner Silva Farias de Oliveira²

Resumo: A manutenção de software é uma atividade custosa e propensa a erros tanto para desenvolvedores de software quanto para usuários finais. Ao saber como e quais requisitos do software precisam ser alterados, os usuários finais poderiam realizar manutenções principalmente do tipo perfectivas e adaptativas assistidas por ferramentas. Porém, a literatura atual carece de ferramentas que suportam estes tipos de manutenção automática em cenários reais e permitam a expressão de requisições de mudança através de linguagem natural. Ainda mais grave, as ferramentas atuais são incapazes de entender a semântica das requisições, bem como executar as devidas transformações no software em manutenção. Este artigo, portanto, propõe o Hermes, um modelo de interface de linguagem natural para transformação em software. Ele combina técnicas de linguística computacional e programação em lógica para realizar requisições em um formalismo conhecido pelo modelo MITRAS. O Hermes interage com o usuário final através de linguagem natural, interpreta a semântica das requisições de mudança e mapeia tais requisições para regras de produção de grafos do MITRAS que alteram o software. Hermes foi avaliado através de um estudo empírico com 8 participantes para investigar o seu desempenho e o seu nível de aceitação e usabilidade. Os resultados mostram que o Hermes foi preciso, ao encontrar um número elevado de transformações corretas, e altamente aceito pelos participantes. Os resultados são encorajadores e mostram o potencial de uso da Hermes para produzir adequadamente solicitações de manutenção de software.

Palavras-chave: PLN, *Parser* de Dependência, Análise Semântica.

1 INTRODUÇÃO

Ao longo dos últimos 20 anos, a Linguística Computacional (LC), também conhecida como Processamento de Linguagem Natural (PLN), cresceu tanto na área de pesquisa científica quanto na tecnologia de forma prática e está cada vez mais incorporada a produtos destinada a usuários (como o tradutor da Skype e Apple Siri). (FOUCART; FRENCK-MESTRE, 2015).

Dentre os diversos ramos da LC a engenharia de software (ES) oferece possibilidades muito interessantes de pesquisa. Normalmente, há demanda por pequenas mudanças em produtos de software, enquanto em projetos maiores, usuários ficam sem opção a não ser esperar longos períodos e por serviços caros. Neste contexto de alterações constantes, a

¹ Aluno do Curso de Ciência da Computação. E-mail: mwmaci@edu.unisinos.br

² Orientador, professor da Unisinos, Doutor em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2012), Mestre em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (2008). E-mail: kleinnerfarias@unisinos.br

manutenção de software tem se tornado uma atividade altamente custosa e propensa a erros tanto para desenvolvedores de software quanto para usuários finais. Usuários de sistemas então geram requisições de mudança para desenvolvedores, os quais de fato irão implementar as melhorias alterando o código da aplicação. Na prática, este ciclo de identificação, comunicação e desenvolvimento dos pontos de melhorias tem se mostrado bastante oneroso. Uma abordagem inovadora seria permitir aos usuários finais realizarem as devidas alterações no software através de algum suporte ferramental, visto que eles sabem exatamente quais requisitos precisam ser alterados, evitando problemas de comunicação e reduzindo o ciclo de manutenção.

Porém, a literatura atual carece de ferramentas que suportem manutenção automática em cenários reais e permitam a expressão de requisições de mudança através de linguagem natural. PLN tem sido amplamente utilizado para tratar variados desafios como, por exemplo, a criação de *parsers* de processos para linguagem natural (RODRIGUES; AZEVEDO; REVOREDO, 2010), de linguagem natural para processos (AMORIM et al., 2016), detecção de padrões em ambiente web (ALBERTO et al., 2017), entre muitos outros. Ainda pior, as ferramentas atuais utilizadas para manter software ainda são incapazes de entender a semântica das requisições de mudança geradas pelos usuários finais, bem como executar as devidas transformações no software em manutenção.

Este artigo, portanto, propõe Hermes, um modelo de interface de linguagem natural para transformação em software. Ele combina técnicas de linguística computacional e programação em lógica para realizar manutenções automáticas em software. O Hermes interage com o usuário final através de linguagem natural, interpreta a semântica do texto das requisições de mudança e mapeia para um formato utilizado pelo MITRAS para identificar as produções de grafos que irão implementar a manutenção em software. O MITRAS (KÜPSSINSKI; GLUZ, 2018) é um projeto do Programa de Pós-Graduação em Computação Aplicada (PPGCA) da Unisinos e trata-se de um modelo inteligente que utiliza o formalismo de grafos para representar e realizar transformações no software em manutenção. O Hermes ataca uma lacuna do MITRAS que seria a falta de uma interface que permita ao usuário final interagir via texto com o sistema para efetuar tais modificações.

Hermes foi avaliado através de um estudo empírico com 8 participantes para investigar o seu desempenho e o seu nível de aceitação e usabilidade. Os resultados mostram que o Hermes foi preciso, ao encontrar um número elevado de transformações corretas, e altamente aceito pelos participantes. Os resultados são encorajadores e mostram o potencial de uso da Hermes para produzir adequadamente solicitações de manutenção de software.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica que servem como base para a criação e aplicabilidade do Hermes. Posteriormente na Seção 3 serão apresentados os trabalhos relacionados com o modelo proposto. Na Seção 4 será apresentado o modelo proposto e sua arquitetura. Em seguida, a Seção 5 descreve a implementação do Hermes. Por fim a seção 6 apresenta a avaliação, mostrando os materiais, cenários de avaliação e os métodos de análise para posteriormente apresentar os resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados os conceitos teóricos sobre o Processamento de Linguagem Natural e as ontologias empregadas como apoio que servem de base para a construção e avaliação do Hermes. Na Seção 2.1 é introduzido o conceito e apresentado as tecnologias de PLN. A Seção 2.2 apresenta os conceitos sobre ontologias e linguagem OWL.

2.1 Stanford CoreNLP

O objetivo da ciência da linguagem é poder caracterizar e explicar as múltiplas observações linguísticas que circundam a nossa volta, nas conversações, na escrita e em outros meios de comunicação. Parte disso tem a ver com o lado cognitivo de como os seres humanos adquirem, produzem e entendem a linguagem, parte tem a ver com a compreensão da relação entre as expressões linguísticas e o mundo, e outra parte tem a ver com a compreensão das estruturas linguísticas pela qual a linguagem humana se comunica. (MANNING; SCHUTZE, 1999).

O Processamento de Linguagem Natural (ou Linguística Computacional) ou Computação linguística, é um subcampo da ciência da computação envolvido com o uso de técnicas computacionais para aprender, compreender e produzir conteúdo da linguagem humana. (FOUCART; FRENCK-MESTRE, 2015).

Para tratar o problema de compreender as estruturas linguísticas da comunicação da linguagem humana, linguistas propuseram que existem regras que estruturam expressões linguísticas. Essa abordagem básica existe a centenas de anos, mas nas últimas décadas esta abordagem ficou muito mais formal e rigorosa pelo fato de linguistas explorarem a gramática mais detalhadamente mostrando a diferença entre frases bem formadas de uma linguagem para uma linguagem com uma estrutura de frases malformada. (MANNING; SCHUTZE, 1999).

Genericamente ao processo de análise de uma estrutura textual quanto as suas características léxicas, sintáticas e semânticas, será chamado neste artigo de processo de *parsing* de uma linguagem. O Stanford CoreNLP (MANNING et al., 2014) fornece um conjunto de ferramentas de tecnologia de linguagem humana e será a base para trabalhar no processo de *parsing* de uma linguagem natural. Um diferencial desse *parser* é a implementação do conjunto de dependências universais que tem como objetivo unificar o conjunto de *tags* e classificação permitindo a análise através de vários idiomas. O conjunto de dependências universais surgiu a partir da observação de pesquisadores de diversas instituições, entre elas a própria Stanford e Google, de que existiam esforços paralelos em capturar similaridades bem como idiossincrasias em diferentes tipos de idiomas. A partir de 2014, começou-se a trabalhar com um banco de textos classificados utilizando *tags* universais, o Universal Dependency Treebank. (NIVRE et al., 2016).

Como o *parser* de Stanford trabalha com modelos probabilísticos e neurais previamente treinados, cabe ao cliente escolher qual modelo disponível é o mais adequado para a aplicação, inclusive podendo treinar um modelo caso seja necessário. Estas são algumas das características que tornam o *parser* de Stanford uma opção flexível de usar.

As subseções seguintes trazem de forma resumida as principais técnicas e ferramentas empregadas neste processo.

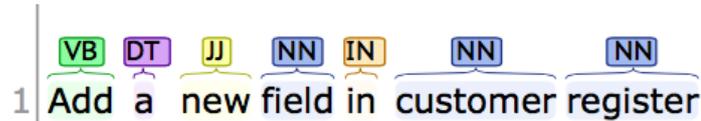
2.1.1 *Part of Speech Tagger*

Ao realizar uma análise gramatical de uma frase, uma possível observação a se fazer é como as palavras estão organizadas no decorrer de um texto bem como identificar suas classes gramaticais. Um sistema capaz de receber como entrada um texto de um determinado idioma e associar a cada palavra sua classe gramatical, é dado o nome de *Part of Speech Tagger* (*POS Tagger*).

Diversos algoritmos de aprendizado de máquina são empregados na criação de *taggers* melhores. Dentre as abordagens de métodos de melhor desempenho se destacam o uso de Modelos Ocultos de Markov (BRANTS, 2000), modelos de máxima entropia (ADWAIT RATNAPARKHI, 1996) e aprendizado por transformação. (BRILL, 1994). Essas abordagens usam largamente a mesma fonte de informação para *tagging* bem como frequentemente possuem as mesmas características e chegam a acertar mais de 96% dos casos, sendo que sua principal dificuldade é trabalhar com palavras desconhecidas. (TOUTANOVA et al., 2003).

Na Figura 1 é possível ver a saída de uma frase processada por um *POS Tagger*. Para

Figura 1 – Frase processada pelo *POS Tagger*



Fonte: Elaborado pelo autor.

cada palavra é atribuída uma *tag* que representa a classe gramatical (exemplo: substantivo, verbo, adjetivo, etc.) Nesta etapa é extraída uma parte que, junto com outras técnicas necessárias, possibilita a identificação das relações entre as palavras.

O sistema *POS Tagger* usado neste artigo é baseado no conjunto de dados do *Wall Street Journal* do *Penn Treebank III*. (MARCUS et al., 1994).

2.1.2 Name Entity Recognizer

É comum encontrar nomes próprios e números escritos em formato não decimal ao realizar processamento de linguagem natural. A técnica *Name Entity Recognizer* (NER) é usada para reconhecimento de nomes e entidades, identificando de forma mais fácil nomes próprios e numerais. Esse tipo de distinção ocorre tanto pelo contexto da frase quanto a partir de vocabulários pré-definidos de termos e conceitos. (FINKEL; GRENAGER; MANNING, 2005).

Na Figura 2 é possível observar que tanto no nome próprio bem como no número ordinal *first* foram corretamente identificados pelo classificador *Name Entity Recognizer* e tiveram seus significados expostos.

Mesmo considerando que é possível inferir entidades como nomes próprios, uma das formas de trabalhar com *Name Entity Recognition* pode envolver o uso de ontologias, de modo a criar uma classificação dos termos conhecidos para posterior consulta no *A-box* da ontologia, ao invés de criar um novo classificador treinando uma rede neural que aprenda determinadas entidades dentro de um contexto. Munindo-se desse artifício, um classificador de entidades pode ser melhorado a partir da inserção de novos termos na ontologia.

2.1.3 Grafo de dependências

Na esfera do processamento de linguagem natural, um desafio é a identificação de dependências entre as palavras de uma determinada sentença. Uma das formas de representar esta estrutura é a utilização de grafos.

Figura 2 – Frase processada pelo *Name Entity Recognizer*

1 | Put PERSON Michael 's address in the ORDINAL first 1.0 position

Fonte: Elaborado pelo autor

Considerando $S = w_1, w_2, \dots, w_n$ uma frase onde w_i representam cada palavra, $G = (V, A)$ é o grafo de dependências da frase sendo que $V = (w_1, w_2, \dots, w_n)$ são os vértices e $A \subseteq V \times V$ é o conjunto de arestas que representa as dependências entre as palavras. A Figura 3 ilustra o grafo de dependências gerado por uma frase.

O tipo de grafo que se pode notar na Figura 3 é um grafo dirigido. A direção das arestas representa o significado de uma palavra de origem para uma de destino, um exemplo é a relação de um verbo e um objeto como no caso do exemplo o verbo *move* aponta para *field* detectando a relação de objeto direto, representando a ação de mover um campo.

2.2 Ontologias

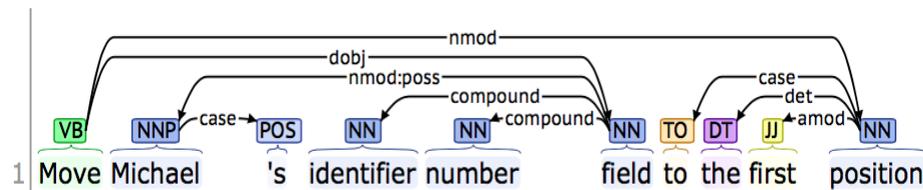
No contexto da representação do conhecimento, uma ontologia pode ser descrita como uma teoria lógica que representa a semântica de entidades considerando suas regras restritivas. (GIARETTA; GUARINO, 1995). Dessa forma é possível criar e organizar conceitos que se referem a uma realidade, bem como relacionar esses conceitos.

Diversas técnicas podem ser usadas para modelar a representação do conhecimento. (GÓMEZ-PÉREZ et al., 2004). Nos últimos anos, técnicas de representação do conhecimento baseadas em Lógicas de Descrição (LD) têm sido usadas para construir ontologias e novas linguagens de descrição lógica. (BAADER et al., 2003).

O uso de LD para representação de conhecimento de ontologias é motivado porque as LD formam uma família de linguagens lógicas de representação do conhecimento do domínio de aplicação bem compreendidas formalmente e estruturalmente. Dessa forma um domínio de conhecimentos é descrito de forma precisa através de expressões construídas por conceitos atômicos (predicados unários) e regras atômicas (predicados binários) que seguem regras fornecidos por uma LD específica. Dentro da esfera da LD, *Attributive concept Language (ALC)* é a descrição lógica básica da sintática e semântica além de complementos de uma linguagem de representação do conhecimento. (VAN HARMELEN, FRANK; LIFSCHITZ, VLADIMIR; PORTER, 2008).

Ontologias podem ser modeladas usando diferentes técnicas e estas podem ser

Figura 3 – Exemplo de um grafo de dependências



Fonte: Elaborado pelo autor

implementadas em várias linguagens. (USCHOLD; GRUNINGER, 1996). A linguagem utilizada neste artigo para modelagem, tanto das entidades de software bem como da criação de bases de conhecimento que irão auxiliar no desenvolvimento do modelo, será a linguagem OWL. (VAN HARMELEN, FRANK; LIFSCHITZ, VLADIMIR; PORTER, 2008).

3 TRABALHOS RELACIONADOS

Esta seção tem objetivo de realizar uma análise comparativa dos trabalhos relacionados. Para isso, a Seção 3.1 analisa quatro artigos selecionados que exploram o problema de pesquisa abordado neste artigo. A Seção 3.2 apresenta uma análise comparativa destes trabalhos.

Hermes preenche a lacuna do projeto MITRAS, que visa permitir que usuários sem qualquer conhecimento em programação possam efetuar requisições para modificações em software através de um sistema inteligente de manutenção automática. Também é possível notar que o MITRAS não é uma ferramenta de transformação e modelagem convencional, feita para desenvolvedores ou designers e sim para pessoas sem conhecimento de programação, utilizando processamento de linguagem natural.

O conceito de um software poder aplicar mudanças em outro software ou nele mesmo não é novo e existem diversos exemplos em áreas de programação automática utilizando várias técnicas que incluem programação com exemplos por Gulwani, (2011), programação genética por heurística por Igwe e Pillay, (2013) e Weimer et al. (2009), sintetização por Duong, Nguyen e Chandra (2013), conceito de mineração proposto por Talanov, Krekhov e Makhmutov (2010) e técnicas de geração e validação para reparo por Weimer, Fry e Forrest, (2013).

Outra tendência de pesquisa é a de reparo automático em programas já existentes, aplicadas em programas procedurais e orientados a objetos (utilizando técnicas de métodos de invocação). (SAHA et al., 2017).

Porém a proposta do MITRAS difere de todas essas abordagens por se centrar na etapa de manutenção perfectiva e adaptativa combinado com o uso de PLN. Assim não foram

encontrados trabalhos similares especificamente voltados ao uso de PLN como meio de comunicação com sistemas de manutenção automática de software. Porém foram encontrados trabalhos que podem ser relacionados ao modelo proposto, por utilizarem técnicas de PLN preferencialmente para fins de programação ou desenvolvimento de software.

3.1 Análise dos trabalhos

A escolha dos trabalhos levou em conta se o artigo tratava a recuperação de expressões digitadas por usuários ou coletada de alguma base de dados, se houve a recuperação de entidades no contexto do trabalho, se havia a troca de mensagens (conversa) entre usuário e máquina caso necessitasse complementar alguma informação faltando e se havia uma ação em alto nível que executasse um plano.

Programming Bots by Synthesizing Natural Language Expressions into API Invocations (ZAMANIRAD et al., 2017). Este trabalho propõe *BotBase*, o qual emprega técnicas de síntese de expressões de usuários em linguagem natural para efetuar chamadas concretas de API's apoiados por um grafo de uma base de conhecimento de API's. Esta base contém informações sobre diversas API's, suas declarações, expressões, parâmetros e possíveis valores. A aquisição da base de conhecimento das API's ocorre em dois passos principais: utilizando *crowdsourcing* visando uma abordagem incremental e coletiva e utilizando técnicas de *deep learning* para servir de camada para automatizar o processo. Logo, usuários utilizando linguagem natural podem efetuar chamadas de API's de uma forma mais direta.

A fim de extrair palavras que combinem com alguma API, Zamanirad et al. (2017) utiliza o *POS Tagger* de Stanford e a saída serve como primeiro passo para interpretar a intenção do usuário. Já com estes dados em mãos, uma comparação de similaridade léxica é feita através da similaridade com uma API candidata e posteriormente técnicas de seletores de declarações e mapeamento entidade-parâmetro são executadas a fim de encontrar a semântica das requisições de usuários.

Generating Semantically Precise Scene Graphs from Textual Descriptions for Improved Image Retrieval (SCHUSTER et al., 2015). Este trabalho propõe um *parser* de descrições de imagem para grafos de cena e podem ser usados também como *queries* para recuperação de imagens. Foram criados dois *parsers* e ambos operam na representação linguística que se refere a um grafo semântico. Esse grafo semântico é obtido através da técnica

de *parsing* que recupera descrições de imagens para gerar árvores de dependência. Os autores propuseram esta técnica utilizando técnicas de PLN que, através de uma base de sentenças digitadas em linguagem natural, utilizando o Parser de Stanford como analisador, e com técnicas de treinamento analisando uma comparação exata entre uma *string* e palavras vindas do *parser* e técnicas de comparação de menor distância euclidiana permitem executar a tarefa de *parsing* de descrições de sentença para grafos de cena.

Program Synthesis using natural language (DESAI et al., 2016). O *framework* proposto mapeia requisições em linguagem natural para a sintetização³ de programas de linguagem de domínio específico, do inglês *domain-specific languages* (DSL). No cenário de síntese de programas, o artigo trata o PLN impreciso na tarefa de sintetização. Em vez disso, é gerado um ranking de programas e deixa a cargo do usuário selecioná-los, tanto inspecionando o código-fonte ou executando com algumas entradas como teste. A lógica usada por Desai et al. (2016) converte cada palavra de entrada de usuário em um ou mais terminais usando linguagem natural para produzir um dicionário de terminais. Como avaliação, o algoritmo de síntese usou o *POS Tagger* de Stanford para tratar os dados de entrada de usuário, mas não identificou quais tratativas foram usadas.

Learning Language Games through Interaction (WANG; LIANG; MANNING, 2016). WANG, LIANG e MANNING (2016) cita a ideia de humanos cumprir alguma tarefa utilizando o conceito de linguagem de jogos (por exemplo, alcançar uma certa configuração de blocos), mas somente se comunicando com um computador, o qual configura o cenário desejado. Inicialmente o computador não sabe nada sobre a linguagem utilizada e então deve aprender por método de exaustão através da interação com o usuário, o qual adapta as capacidades do computador.

O artigo criou um jogo chamado *SHRD-LURN* e o objetivo é transformar um estado inicial em um estado final, mas o único ato possível que o usuário precisa realizar para chegar no objetivo é digitar expressões em linguagem natural. Então o computador faz o *parse* destas expressões e produz uma lista com o ranking de possíveis interpretações de acordo com o modelo corrente, logo o usuário escolhe qual interpretação de comando está correta e então o computador entende como *feedback* do usuário que para aquela linguagem digitada, o comando a ser executado é o escolhido pelo usuário.

³ É a tarefa de automaticamente sintetizar programas e neste caso sobre algum DSL a fim de descobrir/satisfazer a intenção dada pelo usuário.

3.5 Análise comparativa dos trabalhos

A Tabela 1 apresenta a análise comparativa dos trabalhos, a qual é feita utilizando seis critérios de comparação. Embora três trabalhos (ZAMANIRAD et al., 2017), (DESAI et al., 2016) e (WANG; LIANG; MANNING, 2016) sejam capazes de gerar software, nenhum deles trata de alteração de software. Considerando o critério *Entity Recognizer*, apenas os trabalhos (ZAMANIRAD et al., 2017) e (DESAI et al., 2016) abordam esta temática. Pode-se observar também que os trabalhos relacionados não oferecem suporte a conversação. Referente ao tópico sobre as ontologias, somente o trabalho de Desai et al. (2016) utiliza uma base de dados contendo informações sobre voos para posteriormente comparar *queries* de entrada com esta base de dados. Enquanto três artigos (ZAMANIRAD et al., 2017), (DESAI et al., 2016) e (WANG; LIANG; MANNING, 2016) apresentaram soluções de comparação léxica com alguma base de dados própria ou de terceiros, o artigo de Schuster et al. (2015) usa técnicas de *parsing* utilizando um parse estatístico.

O diferencial do Hermes é, portanto, propor um modelo de PLN que utiliza uma abordagem de conversação com usuários de softwares para processar suas requisições, utilizando um conjunto de Ontologias como base para trazer significado as diversas entidades relacionadas a aplicação. Outro diferencial está na criação e na abordagem de utilização de um método de *parsing* parecida com o trabalho de Schuster et al. (2015) diferenciando apenas as tratativas do pós-processamento dos dados de saída do *parser* estatístico que é feita através de regras lógicas.

4 MODELO HERMES

Esta seção apresenta Hermes, um modelo de interface de linguagem natural para transformação em software. Para isso, a Seção 4.1 apresenta as transformações consideradas pelo MITRAS seguindo da arquitetura geral do modelo. Seção 4.2 detalha os módulos da arquitetura e as diferentes ontologias empregadas. A Seção 4.3 introduz o módulo de PLN. A Seção 4.4 descreve o módulo de transformação semântica.

4.1 Transformações consideradas pelo MITRAS

O sistema de transformações do MITRAS é representado por regras de transformações em grafo (GT) que são produções de grafos previamente escritos em gramáticas de grafo.

Tabela 1 – Quadro comparativo dos trabalhos relacionados

Trabalhos	Gera Software	Altera Software	Entity Recognizer	Suporte a conversação	Utiliza ontologias como base	Tipo de abordagem de parsing
(ZAMANIRAD et al., 2017)	Sim	Não	Sim	Não	Não	Parser com comparação léxica
(SCHUSTER et al., 2015)	Não	Não	Não	Não	Não	Parser estatístico
(DESAI et al., 2016)	Sim	Não	Sim	Não	Sim	POS Tagging e parser com comparação léxica
(WANG; LIANG; MANNING, 2016)	Sim	Não	Não	Não	Não	Parser com comparação léxica

Fonte: Elaborado pelo autor.

Dependendo do nível da granularidade do modelo de grafo de um software, GT podem representar qualquer tipo de transformações em software. No entanto, o modelo MITRAS inicialmente utilizou esforços para representar GT de alto nível, efetuando manutenções do tipo perfectivas e adaptativas as quais objetivam aprimorar um software em uso. Já manutenções do tipo corretivas e preventivas visam corrigir erros em um software e estariam em um nível de abstração não considerado pelo MITRAS.

Dois tipos de alto nível em transformações foram considerados:

- tipos de características de “adicionar mais do mesmo”: inserir novas entidades, atributos e processos em software ou mudanças na interface do usuário, as quais podem ser facilmente incorporadas no modelo de software trabalhado.
- tipos de características “ver menos do mesmo”: ocultam entidades e atributos da apresentação do modelo para diminuir a carga cognitiva do usuário da interface, sem efetivamente deletar esses elementos do modelo do software.

4.2 Arquitetura do modelo

Com base nas GT possíveis dentro do MITRAS essa seção irá apresentar a arquitetura do modelo proposto, que consiste em um *pipeline* que integra três grandes módulos e três tecnologias diferentes:

- Módulo de processamento de linguagem natural que consiste em um conjunto de ferramentas do *CoreNLP* de Stanford oferecidas por Manning et al. (2014) para análise sintática das frases digitadas.
- Módulo analisador semântico que utiliza programação lógica para correlacionar as *tags* organizadas em forma de grafo de dependências entregues pelo módulo de PLN com três bases ontológicas relacionadas ao software que está sob processo de manutenção.
- Módulo de transformação em grafo do MITRAS, é o módulo externo, que não é objeto dessa proposta, que trabalha cooperativamente com o módulo de transformação semântico para implementar mudanças em software.

A descrição do fluxo do *pipeline* completo que abrange também o sistema MITRAS, externo a este modelo, ocorre da seguinte maneira: inicia com a mineração do código-fonte e posteriormente espera que os dados em forma de um grafo de ação façam o mapeamento inferindo em quais regras de transformação em grafo são necessárias para posteriormente transpor para o código fonte as transformações necessárias através de técnicas de injeção em código-fonte. A Figura 4 apresenta a visão geral da arquitetura.

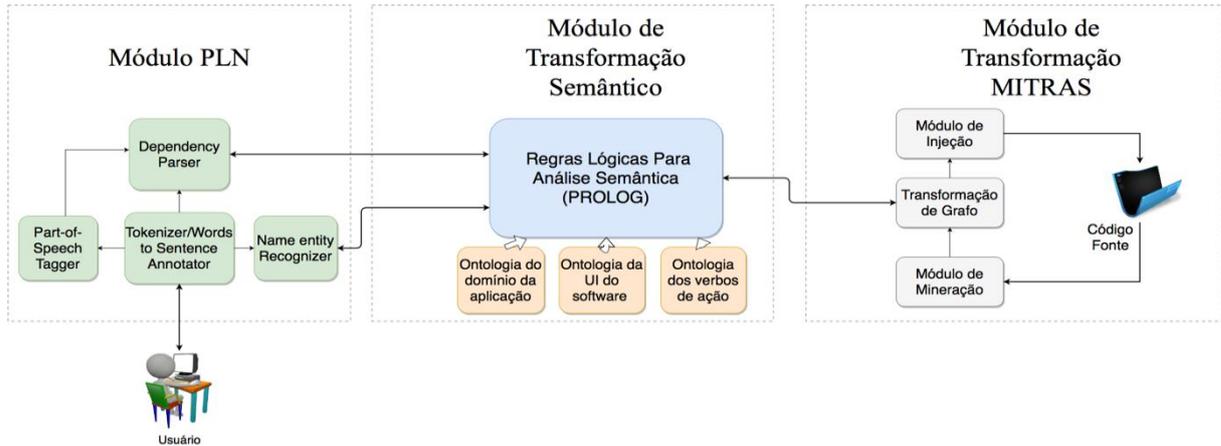
Neste trabalho foi necessário encontrar um sistema bem estruturado, documentado e que fosse possível acessá-lo inteiramente, desde o código-fonte até a execução do software e que teria ainda como critério ser um sistema real atualmente implementado e em uso. Foi então pesquisado alguns softwares que satisfaçam os seguintes critérios:

- a) um projeto de código-fonte aberto (*open source project*);
- b) que fizesse uso da programação em linguagem orientada a objetos;
- c) existência de uma comunidade ativa e;
- d) uma aplicação diferente das conhecidas e trabalhadas previamente pelo autor deste trabalho.

Apenas projetos com código aberto foram escolhidos devido a necessidade de se conhecer o código-fonte pois é através do levantamento dos conceitos e entidades do sistema que serão trabalhadas as requisições de manutenção por usuários na fase da avaliação do protótipo de linguagem natural.

Após as pesquisas foi escolhido o software *OpenMRS* de Seebregts et al. (2009) que é uma aplicação de um sistema de gerenciamento em saúde, ou em inglês *Healthcare Management System* (HMS). Este software se encaixa em todos os critérios acima, sendo um software de código aberto, completamente escrito em Java, aplicado em uma grande

Figura 4 – Arquitetura do Hermes



Fonte: Elaborado pelo autor.

comunidade ativa e que não se tinha conhecimento prévio, além de possuir mais de 180 mil linhas de código e com certeza não é um *toy-system*.

Nas seções a seguir serão detalhados os dois primeiros módulos que condizem com o desenvolvimento do protótipo aqui apresentado e também como é implementado o analisador semântico.

4.3 Módulo de Processamento de Linguagem Natural

O conjunto de ferramentas do *CoreNLP* de Stanford proposto por Manning et al. (2014) é capaz de efetuar o *parser* de uma sentença e gerar uma árvore da sintaxe correspondente e tem várias outras funcionalidades como o *Part-of-Speech Tagger (POS Tagger)* que analisa as formas gramaticais de cada *token* gerado, um reconhecedor de entidades e um gerador de grafo de dependência, os quais são utilizados pelo módulo de processamento de linguagem natural.

Grafo de dependências universal de De Marneffe et al. (2014) é um grafo que representa as dependências dos termos de uma sentença. Este tipo de grafo muitas vezes está próximo de uma representação semântica superficial e logo é um bom ponto de partida como base para tratar a semântica de requisições de usuário.

Inicialmente, o usuário entra com uma frase de requisição qualquer e então o *Parser* de Stanford separa em *tokens* cada palavra de entrada e posteriormente organiza em forma de uma sentença. Após esta etapa o *POS Tagger* analisa os *tokens* e classifica as formas gramaticais de cada palavra, indicando qual é um verbo, adjetivo ou substantivo por exemplo. Além disso, o *Name Entity Recognizer* auxilia a inferir nomes próprios e indexa números inteiros aos números escritos por extenso obtidos através do POS Tagger, permitindo trabalhar de forma mais simples

com este tipo de dados. Após esta etapa o *parser* de dependências infere as relações entre cada palavra já classificadas gramaticalmente e entrega um grafo sintático de dependências como resultado. Esta etapa finaliza o primeiro módulo e entrega para o módulo analisador semântico o *grafo de dependências* sintático que é o primeiro grafo obtido após a entrada da frase do usuário. A Figura 5 ilustra a frase de exemplo “*Create a field called height in the name panel*” digitada pelo usuário e representa o grafo de dependências gerada pelo do *parser* de Stanford.

4.4 Módulo de transformação semântico

Este módulo tem o objetivo de possibilitar o entendimento do que de fato o usuário está tentando realizar através do uso do protótipo, ou seja, tratar a forma semântica de entrada do usuário.

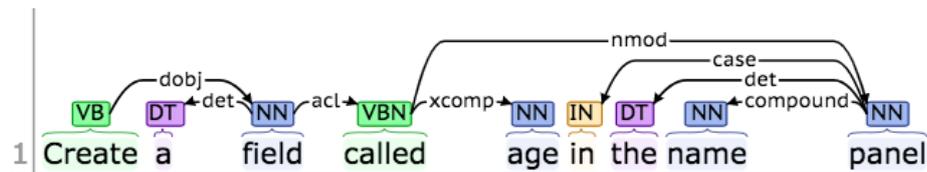
Dado que o módulo anterior entrega um grafo sintático de dependências, esse pode ser descrito em Lógica de Primeira Ordem por predicados monádicos como *verb(create)*, o qual define a categoria sintática (vb – verb) para a palavra “*create*” e predicados diádicos como *compound(panel,name)*, o qual define uma palavra composta da dependência entre as palavras “*panel*” e “*name*”.

Quando usuários interagem com o Mitras, é esperado que ele ou ela se refiram à alguma instancia da aplicação que condiz aos elementos da interface gráfica, o qual é a parte do sistema que o usuário conhece e enxerga. Para cada X e Y que satisfaçam uma das duas fórmulas lógicas ($vb(X) \vee nn(X)$) e ($nn(X) \wedge nn(Y) \wedge compound(X, Y)$), a ontologia do domínio da aplicação é consultada para verificar se X ou a concatenação entre X e Y fazem parte da aplicação, se um ou mais termos são encontrados na ontologia, então o grafo é modificado com informações a respeito de elementos que se referem a interface gráfica do software, resultando em um *grafo de dependências semântico*, também representado por predicados monádicos e diádicos em lógica de primeira ordem. Nas subseções a seguir serão apresentadas as diversas ontologias e suas relações com o modelo de linguagem natural dando sequência a produção do *pipeline*.

4.4.1 Ontologia do domínio da aplicação

A ontologia do domínio da aplicação foi criada para guardar as informações que dizem respeito ao conceito das entidades e/ou serviços referentes ao software *OpenMRS*. Um exemplo seria uma requisição de usuário para adicionar um novo campo “endereço” no painel do

Figura 5 – Grafo de Dependências



Fonte: Elaborado pelo autor

paciente. A ontologia então será a base das informações “campo” e “painel” que são conceitos referentes a entidades da interface gráfica do software, assim fazendo uso da ontologia como representação do domínio da aplicação. A Figura 6 ilustra uma pequena parte da ontologia contendo conceitos do domínio da aplicação da interface gráfica.

4.4.2 Ontologia da interface gráfica do software (UIO)

A ontologia da interface gráfica do software permite identificar que tipo de elemento da interface gráfica o usuário está se referindo. Neste caso a ontologia será a base para nomes de entidades reais do software *OpenMRS*, como por exemplo um painel chamado “*address panel*”, ou também um nome de um campo, exemplo “*middle name*”, estes são exemplos de entidades com nomes que o usuário pode visualizar.

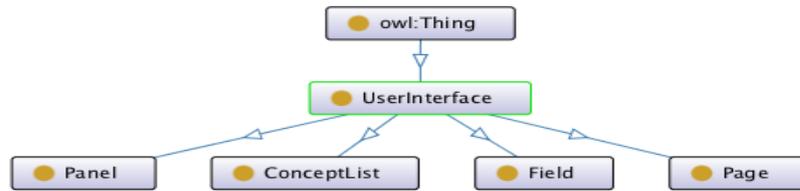
Para que os usuários não precisem digitar nomes, ações e comandos idênticos aos dados à arquitetura de um software, foi criada uma ontologia à parte capaz de guardar e atualizar sinônimos, assim usuários podem se referir coloquialmente ao software e o modelo mantém a ontologia atualizada para cada perfil de usuário.

Para mapear nodos do grafo semântico às entidades do software, como classes reais, variáveis e métodos, todos os objetos na ontologia da interface gráfica tem uma anotação única chamada *Id* com o nome da entidade correspondente no código-fonte. Após identificar na ontologia qual entidade do software corresponde a informação digitada pelo usuário, o grafo de dependência semântico é novamente modificado para se obter um *grafo de dependências mapeado*. Este grafo então é obtido através da consulta na ontologia da interface do software obtendo os *Id's* específicos que representam estes nodos no código-fonte.

4.3.3 Ontologia dos verbos de ação (VAO)

O último passo do *pipeline* é inferir a intenção do usuário e construir um *grafo de ação* correspondente (grafo final). Este passo ocorre através de um processo de inferência executado

Figura 6 – Conceitos da Interface Gráfica



Fonte: Elaborado pelo autor

sobre os verbos presentes no grafo de dependências mapeado, visando eleger precisamente qual transformação correspondente a ação o usuário quer fazer. Esta inferência é baseada na ontologia dos verbos de ação, representa sinônimos de verbos e identifica quais verbos são considerados uma ação de transformação.

Após obter o grafo de dependências mapeado que contém somente as formas canônicas de verbos e entidades do software e identificar quais verbos na frase correspondem a ação de transformação, o grafo de ação é construído através de correlação de regras lógicas que expressam as condições suficientes e necessárias para inferir uma ação de transformação dentro do módulo de transformação do MITRAS.

Um exemplo de uma regra lógica que expressa as condições para inferir a inserção de um novo campo em um específico painel pelo usuário é apresentada a seguir:

$$\begin{aligned}
 &vb(create) \wedge dobj(create, element) \wedge acl(element, named) \wedge xcomp(named, N) \wedge \\
 &nmod(call, P1) \wedge compound(P1, P2) \wedge concat(P1, P2, P) \wedge uid(P, O) \rightarrow \\
 &has(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge notin(N) \quad (1)
 \end{aligned}$$

A constante *create* na regra lógica (1) é a forma canônica para qualquer verbo relacionado com a criação de um novo campo (poderia ser “inserir”, “adicionar”, etc.). Do mesmo modo *element* é o identificador canônico para o novo elemento, campo, variável, enquanto *named* é o identificador para um adjetivo (*acl* – *adjectival clause*) identificando o nome da entidade (poderia ser também “called”, “labeled”, etc.).

As variáveis, *N*, *P1* e *P2*, são palavras da entrada pelo usuário, *P* é o resultado da concatenação através do predicado *concat* e *O* é o Id correspondente ao termo *P*, obtido na ontologia da interface gráfica (UIO) através do predicado *uid*. Os demais predicados são do grafo de dependências: *vb* é o predicado monádico vindo do POS que identifica a palavra sendo um verbo, *dobj*, *acl*, *xcomp*, *nmod* e *compound* são predicados das dependências universais.

É importante deixar claro que esta é apenas uma das tantas regras lógicas desenvolvidas que inferem a intenção do usuário. O modelo aqui proposto é composto de dezenas de regras e que abrangem outras classes de transformações e podem ser encontradas no repositório do GitHub⁴ e as classes de regras estão nomeadas em arquivos *t1.pl*, *t2.pl* e *t3.pl*.

A parte direita da regra (1): $has(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge notin(N)$ é o grafo de ação, representado por predicados em lógica de primeira ordem. Esta parte contém informação necessária para encontrar a regra transformação apropriada para ser aplicada ao software. O predicado $in(O)$ indica que o nodo O já deve estar presente no modelo de grafo do software, enquanto $notin(N)$ indica que o nodo N é um novo nodo não presente no grafo atual.

4.3.4 Limitações da transformação semântica

Com base na construção das regras lógicas, juntamente com a inferência das ontologias, é possível verificar que palavras que fazem parte da ontologia da interface gráfica do software e ao mesmo tempo são classificadas como verbos pelo POSTagger, impossibilitam verificar de forma que esta palavra é um elemento da interface gráfica como por exemplo o campo “*Created By*”. Para contornar este problema existe a necessidade de treinar um reconhecedor de entidades para que deixem de serem classificadas como verbos e passem a ser classificadas como substantivos. Esta é uma das lacunas deixada por este trabalho que pode ser explorado em trabalhos futuros.

Outra limitação é a exaustiva construção de regras lógicas de inferência para abranger as transformações válidas dentro do escopo do MITRAS. Acredita-se que treinar um *parser* dedicado para o cenário de manutenção em software utilizando como base as frases digitadas no experimento deste artigo mais as frases de teste na construção do protótipo pode ser um bom caminho para diminuir a carga de trabalho na construção de regras e esta técnica também pode ser explorada no futuro.

5 IMPLEMENTAÇÃO

A construção do protótipo foi dividida em duas etapas, sendo a primeira o desenvolvimento do programa em Java que coleta das frases de entrada do usuário e a análise sintática probabilística destas frases que gerou um grafo de dependências para a segunda etapa,

⁴ Disponível em: <https://github.com/lucaskup/MITRAS/blob/master/Prolog/NLP>

a qual através de regras lógicas, utilização de ontologias combinadas e a utilização de agentes, permitem a comunicação em linguagem natural. Para a interação entre o usuário e o protótipo foi desenvolvida uma interface *web* utilizando WebStorm e o *framework* Bootstrap.

A primeira parte da aplicação foi desenvolvida em Java na versão 1.8 e utilizado a API do *Core* de Stanford na versão 3.8.0 que dentre várias ferramentas possui o *parser* de Stanford. Optou-se por este *parser* pois o mesmo possui boa documentação como base para referência teórica e trabalha com diversos modelos probabilísticos e neural, cabendo ao cliente escolher qual melhor se adapta a necessidade.

A construção do grafo sintático de dependências é a saída do programa em Java e todo o desenvolvimento posterior a esta etapa se dá através da linguagem de programação SWI-Prolog, na versão 7.6.4. Para possibilitar a comunicação entre Prolog e o programa em Java, foi utilizado o programa *JavaLog*, o qual permite simplificar chamadas de comandos em Java através da programação em Prolog e para trabalhar em um sistema voltado a agentes implementado em Prolog foi utilizado o programa *Agilog*. Para efetuar a programação da aplicação em Java foi utilizado o Eclipse na versão 3.8 e para a aplicação em Prolog foi utilizado o editor de texto *gedit* do SO Linux Ubuntu.

Como a aplicação utiliza os programas *Agilog* e o *JavaLog* escritos em Prolog e estes foram desenvolvidos para rodar em Linux, optou-se por desenvolver toda a aplicação em Linux, facilitando a interação entre as partes.

A aplicação foi desenvolvida orientada a agentes. Três agentes são iniciados ao rodar o *shell*: *webserver_agent*, *nlp_agent* e *trans_agent* e cada agente é uma *thread* do sistema. A descrição geral dos agentes é apresentada a seguir.

Webserver_agent: É responsável por inicializar o servidor web da aplicação e trata requisições do tipo GET, POST, e registra as frases provindas do *nlp_agent*.

Nlp_agent: É o agente que faz o maior trabalho proposto no processamento de linguagem natural. Primeiramente carrega os modelos do Stanford CoreNLP que tratam a sintática das frases e em seguida ficam esperando eventos do *webserver* que recebem frases digitadas pelo usuário na página HTML, enviam estas frases para o *parser* de Stanford e a partir da resposta, executa as regras que inferem qual transformação o usuário está fazendo referência (usando as ontologias como base para a inferência), devolve ao *webserver* a frase de saída da resposta do sistema e apresenta para o usuário, por exemplo, que a requisição foi entendida pelo sistema e que o mesmo encontrou uma transformação válida e também chama o agente *trans_agent* enviando os dados necessários para efetuar a transformação.

Trans_agent: Este agente não faz parte do escopo deste trabalho e é o agente que recebe os dados do *nlp_agent* e efetua as transformações em grafo e sintetizam um novo código-fonte a partir das alterações executadas pelo usuário do sistema.

O software *OpenMRS* foi instalado no SO Linux Ubuntu a fim tornar possível a visualização da interface gráfica para possibilitar o mapeamento das entidades e funcionalidades do software em ontologias.

A aplicação está sendo executada em um computador cujas configurações de hardware possuem um processador Core I5 1.8 GHz, com 8 GB de memória e 128 GB SSD.

6 AVALIAÇÃO

Esta seção tem como objetivo apresentar a metodologia que avalia o Hermes e apresenta os resultados obtidos. A Seção 6.1 descreve a configuração do sistema para uso do Hermes nos testes. A Seção 6.2 contempla o objetivo e as perguntas de pesquisa. Na Seção 6.3 é introduzido os questionários para prover a avaliação do Hermes. Já a Seção 6.4 fala sobre os cenários de avaliação e sobre a seleção dos participantes. Na Seção 6.5 é apresentado o design experimental. Por fim, na Seção 6.6 é feita a análise dos dados coletados.

6.1 Configuração do sistema para implementação dos testes

Os primeiros experimentos com o Hermes foram realizados para verificar a funcionalidade, bem como checar a capacidade do sistema entregar de forma rápida os dados encontrados referentes a entrada do usuário, tanto como feedback ao usuário, como também ao agente de transformações em software do Mitras. Experimentos foram conduzidos em laboratório em uma máquina equipada com um processador Core I7 7500U, 16GB/2400 MHz de memória RAM. Uma média de execução para as três classes de transformações foi calculada, e verificou-se que os tempos não ultrapassaram os 40ms, logo o protótipo ficou com tempo ótimo para uma resposta ao usuário (e ao Mitras) sobre qual transformação foi encontrada.

6.2 Objetivo e perguntas de pesquisa

O estudo realizado na análise semântica da linguagem natural de requisições de usuários para manutenções perfectivas e adaptativas em software têm como objetivo avaliar o desenvolvimento do Hermes no sentido de encontrar transformações válidas dentro do escopo

do MITRAS. As avaliações foram exploradas através do uso de cenários realísticos, baseados em interações com usuários e o Hermes. Logo é avaliado o uso do Hermes no sentido de encontrar transformações corretas. Além disso, também foi avaliada a aplicabilidade e aceitação pela perspectiva de usuários com alguma expertise em uso e/ou manutenção de software. Portanto, este artigo propõe duas perguntas de pesquisa, do inglês *Research Questions* (RQ):

- **RQ1:** O modelo Hermes proposto favorece de forma correta o formalismo de requisições de manutenção em software dentro do escopo do MITRAS?
- **RQ2:** Qual a aceitação do Hermes pelos usuários de software?

6.3 Questionários

Para responder a estas perguntas de pesquisa foram elaborados questionários⁵ e avaliação através de cenários respondidos pelos participantes (introduzidos na subseção 6.4). Os dois questionários elaborados são detalhados a seguir.

Questionário 1: Perfil dos participantes. Tem como objetivo coletar dados referente as características e opinião dos participantes. Criar um perfil dos usuários e analisar os dados coletados é importante para selecionar apenas usuários potenciais para os testes com o Hermes. Para isso, perguntas foram feitas para coletar dados de características em geral, como idade, sexo, profissão, nível de escolaridade e se já utilizou sistemas de *ChatBot*. Informação a respeito de experiência em manutenção em software também foi considerada, no que diz respeito tanto do lado de usuário como de desenvolvedor, além de coletar informações do tempo de experiência. Além disso, informações do nível de conhecimento em inglês também foram coletadas pois trouxeram experiências diferentes para usuários. Ao construir perfis de usuários, entende-se que é de suma importância para avaliar que a técnica proposta será experimentada por pessoas que têm o perfil de futuros usuários do Hermes.

Questionário 2: Questionário TAM⁶. Este segundo questionário avalia questões sobre a usabilidade e aceitação do modelo e visa explorar a RQ2. Este questionário está baseado no Modelo de Aceitação de Tecnologia (TAM), proposto por Davis (1989) e revisado por Marangunic e Granic, (2015). A pesquisa foi dividida em três categorias a fim de avaliar a intenção do usuário em relação a aceitabilidade do Hermes e foram desenvolvidas 8 afirmações conforme a Tabela 2 (Apêndice A). As 4 primeiras questões referem-se à facilidade de uso do

⁵ Questionários disponíveis em <https://goo.gl/forms/fyfPDo6NvJIIRXHD2> e <https://goo.gl/forms/qHRPKQY62skII3V62>

⁶ Tabela TAM traduzida manualmente para o português pelo autor.

Hermes, já da quinta a sétima questão é verificado o nível de percepção de utilidade e a última questão se preocupa em verificar o comportamento por trás da intenção do usuário em validar o Hermes. Usando a escala *Likert*, os usuários podem classificar como “Concordo Totalmente”, “Concordo”, “Imparcial”, “Discordo Parcialmente” e “Discordo”. Ao final do questionário foi disponibilizado um espaço para comentários, sugestões, críticas e elogios. As questões formuladas lidam com tópicos de percepção de facilidade de uso, percepção de utilidade e intenção de comportamento.

6.4 Cenários de Avaliação e Seleção dos Participantes

Os cenários foram escolhidos para representar a realidade de usuários utilizarem o Hermes a fim de solicitar manutenções no software *OpenMRS*. A elaboração dos cenários pretende abranger a RQ1 avaliando se o protótipo é capaz de encontrar de forma completa as três classes de ações de usuário sobre as transformações do Mitras (descrito na Seção 4.2) empregadas no software *OpenMRS*: a adição de um novo elemento na interface gráfica, a movimentação de elementos já existentes e ocultar/excluir elementos. A tela do Hermes pode ser vista na Figura 7.

Para coletar os dados, foram criados quatro cenários abrangendo as três transformações (descritas na Seção 4.1) possíveis pelo usuário, brevemente descritos na Tabela 3. O cenário 1 contém a ação de adicionar um novo elemento ao software *OpenMRS*. Já o cenário 2 possibilita ao usuário deletar um campo da tela. No cenário 3, o usuário é instruído a mover um campo de lugar. Por fim, o quarto cenário contém uma solicitação de dois tipos de modificação, aumentando a tarefa do usuário a fim de criar um cenário um pouco mais complexo e mais abrangente.

Devido à complexidade em selecionar usuários reais do software *OpenMRS*, optou-se por selecionar usuários acadêmicos com formação em áreas da tecnologia da informação, os quais são aptos a avaliar a usabilidade do Hermes no cenário do software *OpenMRS*.

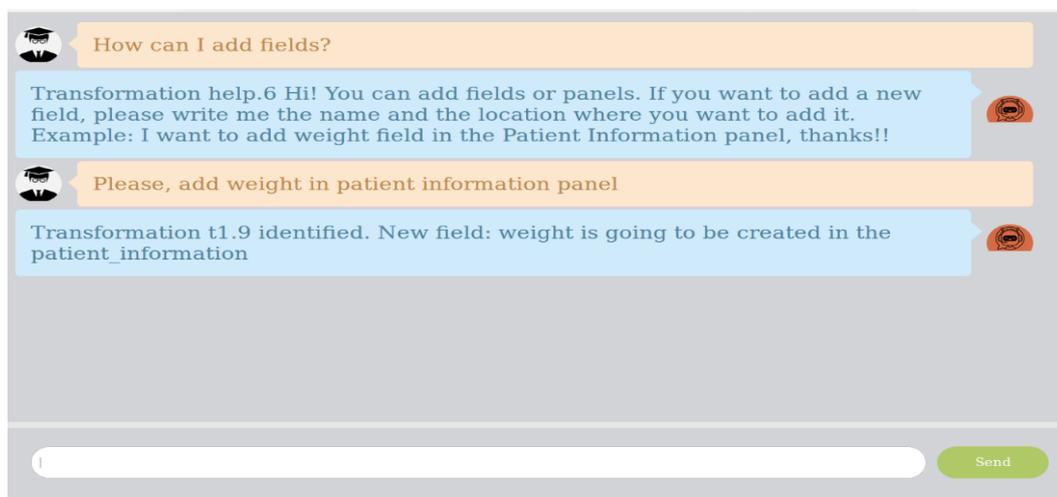
O experimento foi conduzido com 8 participantes, sendo 5 estudantes do Mestrado e Doutorado do programa de pós-graduação em computação aplicada da Unisinos. Também foi escolhido 1 participante do curso de bacharelado em Ciência da Computação também da Unisinos e 2 graduados em Análise de Sistemas pela Ulbra, logo é possível obter diferentes usuários com diferentes perfis de conhecimento e experiência. Uma interface de edição de um cadastro foi escolhida dentro do software *OpenMRS*, a qual é bem completa e possibilita diversas formas de interação entre as possíveis ações pelo usuário. Esta interface é apresenta na

Tabela 3 – Tarefas relacionadas aos cenários do experimento

Tarefa	Transformação	Entidade
1	Adicionar entidade	<i>Nickname</i>
2	Ocultar entidade	<i>Longitude</i>
3	Mover entidade	<i>Postal Code</i>
4	Mover e ocultar entidade	<i>Family Name</i> e <i>Middle</i> respectivamente

Fonte: Elaborado pelo autor.

Figura 7: Tela de interação com o Hermes



Fonte: Elaborado pelo autor.

Figura 8 (Apêndice B). A descrição completa dos cenários de avaliação se encontra no Apêndice C. O experimento foi desenhado similar a um exercício prático de laboratório. Cada participante recebeu o mesmo treinamento para desenvolver os processos experimentais. O processo experimental é apresentado na próxima seção.

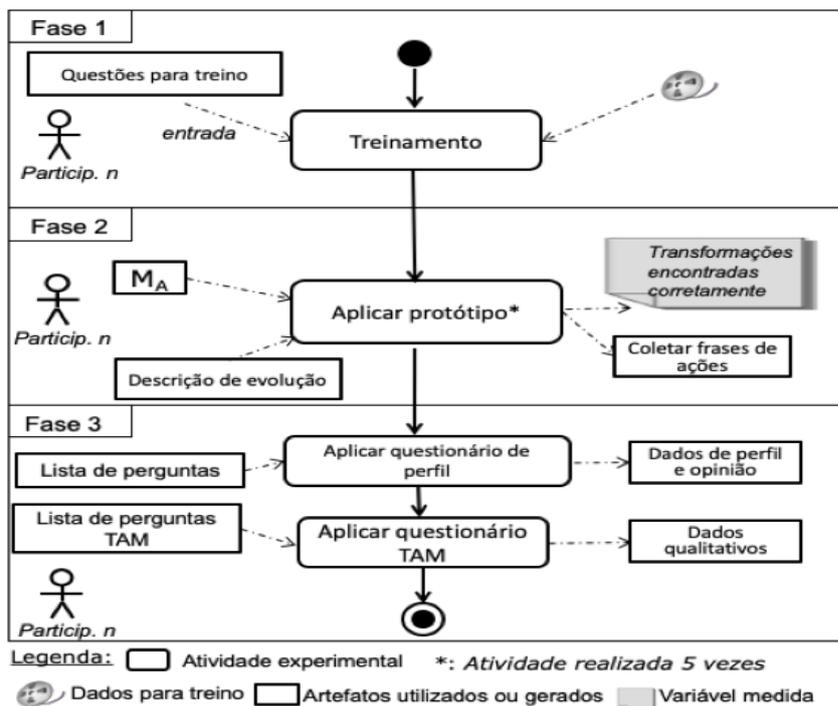
6.5 Processo experimental

A Figura 9 apresenta o processo experimental – formado por uma lista de atividades agrupadas em três fases descritas a seguir.

Fase 1 primeiramente apresenta para ao usuário o objetivo do experimento e o que se espera alcançar com os testes. Nesta fase é rodado um treinamento de como interagir com o Hermes e também é apresentado um curto vídeo autoexplicativo. Com isso é possível ajudar usuários a entender melhor a utilidade do Hermes e consequentemente verificar se entenderam a técnica proposta.

Fase 2 aplicar o Hermes. Participantes são submetidos a rodar 4 cenários e recebem um

Figura 9: Processo experimental



Fonte: Elaborado pelo autor.

script desejado (dados de entrada) para ser produzido. Este *script* de ações (transformações) deve então ser produzido a partir da aplicação de uma descrição de evolução (dado de entrada) para ser aplicado no processo. Nesta etapa, usuários podem ou não encontrar transformações válidas, isso vai depender de como foram digitadas as frases em inglês e se o Hermes foi capaz de compreender a semântica das requisições. Caso encontre todas as transformações com sucesso, 5 grafos de ação serão gerados (saídas de dados para cada usuário) baseado no cálculo de quantas transformações foram encontradas. Este cálculo está relacionado com o sucesso na geração do grafo de ação para cada transformação em cada cenário.

Fase 3 possui duas atividades. A primeira tem o objetivo de coletar informações sobre o perfil e opinião dos usuários que irão responder uma lista de perguntas (dados de entrada) e as respectivas respostas são a saída dos dados coletados como resultado desta atividade. A segunda atividade tem o foco em aplicar o questionário de aceitação de tecnologia (como dados de entrada). Participantes recebem uma lista de perguntas sobre a percepção da facilidade de uso, percepção de utilidade e intenção de comportamento relacionados ao protótipo. Dados qualitativos (saída de dados) são gerados a partir da usabilidade e aceitação do protótipo pela perspectiva dos usuários. Os participantes realizam todas as atividades (Fase 1 a 3) para evitar qualquer inconsistência no processo experimental. Os dados coletados serão discutidos no próximo subcapítulo.

6.6 Resultados

6.6.1 Resultados do perfil dos usuários

A Tabela 4 descreve os resultados do perfil dos usuários, informando as características e opinião dos participantes. Os dados foram coletados no período de 20 de outubro até 07 de novembro de 2018. No total, 8 participantes fizeram parte do experimento. A idade ficou entre 24 e 44 anos. Considerando o nível de escolaridade, mais de 60% dos participantes são estudantes de pós-graduação, 25% possuem graduação completa e apenas 1 participante está com o curso de graduação em andamento. Todos os entrevistados cursam/cursaram graduação na área da computação.

Quase todos os participantes (87,5%) possuem experiência com desenvolvimento de software, sendo que 75% possuem 2 anos ou mais. Metade dos participantes já atuaram em desenvolvimento de software no cenário de manutenção sendo que 2/3 dos participantes (66,6%) informaram que trabalharam mais de 2 anos com desenvolvimento. Mais de 60% dos participantes informaram que já necessitaram requisitar manutenção para um software como usuários de um sistema. Quando perguntado se já haviam utilizado algum sistema de *ChatBot* existente, mais de 60% responderam que sim.

Para a questão do nível de conhecimento em inglês, 37,5% informaram ter pouco conhecimento, já o restante dos participantes informou que tinham conhecimento avançado. Todos os participantes concordam que sistemas de PLN poderiam auxiliar na manutenção automática de código-fonte. Portanto, acredita-se que a amostra é pequena, mas adequada para uma avaliação inicial da proposta sugerida.

6.6.2 RQ1: Resultados que encontraram corretamente transformações válidas.

A Tabela 5 apresenta os dados coletados que avaliam se os participantes conseguiram expressar em linguagem natural ações de manutenção em software corretamente. Os dados estão organizados em quantos acertos ocorreram para cada transformação de cada cenário, quantos *rounds* foram necessários para chegar a uma resposta correta e a porcentagem de acerto. Com o objetivo de avaliar a performance de acerto do protótipo, ou seja, com as frases analisadas de todos os usuários, foi possível verificar o quanto a ferramenta é capaz de encontrar transformações válidas. O alto número de transformações encontradas corretamente, sugerem que o protótipo é apropriado para o objetivo proposto. No pior caso (Transformação 4), 62,5%

Tabela 4: Resultados do perfil dos usuários do protótipo

Característica e opinião (n = 8)	Resposta	#	%
Idade	< 20 anos	0	0.0%
	20-29 anos	3	37,5%
	30-39 anos	3	37,5%
	40-49 anos	2	25%
	> 49 anos	0	0.0%
Nível de escolaridade	Graduação	3	37,5%
	Mestrado	4	50.0%
	Doutorado	1	12,5%
	Outros	0	0.0%
Experiência com desenvolvimento de software	< 2 anos	3	37,5%
	2-4 anos	3	37,5%
	4-6 anos	0	0.0%
	> 6 anos	2	25,0%
Experiência com manutenção de software	< 1 anos	2	33,3%
	1-3 anos	2	16,7%
	3-5 anos	1	16,7%
	> 5 anos	1	16,7%
Já precisou solicitar manutenção em software como usuário	Sim	5	62,5%
	Não	3	37,5%
Já utilizou sistemas de <i>ChatBot</i>	Sim	5	62,5%
	Não	3	37,5%
Nível de conhecimento em inglês	Pouco	3	37,5%
	Intermediário	0	0.0%
	Avançado	5	62,5%
Sistemas de PLN poderiam auxiliar na manutenção automática de código-fonte	Concordo totalmente	4	50,0%
	Concordo	4	50,0%
	Neutro	0	0.0%
	Discordo	0	0.0%
	Discordo totalmente	0	0.0%

Fonte: Elaborado pelo autor

das transformações foram encontradas corretamente. Já na Tabela 6 é possível verificar a saída dos grafos de ação gerados para cada transformação em software requisitada.

Alguns participantes cometeram alguns erros ao escrever em inglês e nestes casos, comprometeu a efetividade do Hermes pois necessariamente um dos passos do pipeline do modelo é passar a frase digitada pelo usuário para o *parser* de Stanford. Frases do tipo: “*delete*

Tabela 5 – Resultado das transformações encontradas

Tarefa	Transformação	Respostas corretas	Max. Rounds	Porcentagem
1	Adicionar campo <i>nickname</i>	8	2	100%
2	Ocultar campo <i>longitude</i>	7	3	87,5%
3	Mover campo <i>postal code</i>	7	4	87,5%
4	Mover campo <i>family name</i>	5	1	62,5%
5	Ocultar campo <i>middle</i>	8	2	100%

Fonte: Elaborado pelo autor.

Tabela 6 – Grafos de ação gerados

Tarefa	Transformação	Grafo de ação gerado
1	Adicionar campo <i>nickname</i>	$O = Patient\ Names, N = nickname$ $has(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge notin(N)$
2	Ocultar campo <i>longitude</i>	$O = Patient\ Addresses, N = longitude$ $hide(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge in(N)$
3	Mover campo <i>postal code</i>	$O = Patient\ Addresses, N = postal\ code$ $move(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge tin(N)$
4	Mover campo <i>family name</i>	$O = Patient\ Names, N = family\ name$ $move(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge notin(N)$
5	Ocultar campo <i>middle</i>	$O = Patient\ Addresses, N = middle$ $hide(O, N) \wedge panel(O) \wedge field(N) \wedge in(O) \wedge in(N)$

Fonte: Elaborado pelo autor

to *longitude Patient Addresses*” e “*move family name position first in patient names*” não são bem compreendidas pelo *parser*, por estarem escritas de forma equivocada, e nestes casos não é possível efetuar uma correção prévia, diferentes dos erros ortográficos que estes sim podem sofrer ajustes antes de ser enviado para o *parser*. Caso usuários consigam perceber erros na criação da sentença, os mesmos podem reescrever as frases e tentar submeter novamente ao Hermes até que o mesmo consiga compreender a frase digitada.

De todas as frases digitadas corretamente, apenas 2 requisições não foram encontradas com sucesso pelo Hermes (5%). Com uma média de 95% de acerto pelo protótipo mostra que o modelo tem potencial para abranger todas as requisições possíveis dentro do escopo do MITRAS, bem como ser útil na automatização da manutenção em software.

6.6.3 RQ2: Resultados referente a aceitação tecnológica

Na Tabela 7 é possível verificar informações coletadas através da aplicação do questionário TAM sobre percepção de facilidade de uso, percepção de utilidade e intenção de

Tabela 7 - Resultados Referentes ao questionário TAM

	Concordo totalmente	Concordo	Neutro	Discordo	Discordo totalmente
Percepção de facilidade de uso					
Achei o protótipo fácil de usar	6	2	0	0	0
Achei o protótipo fácil de aprender	7	1	0	0	0
Achei o protótipo fácil de dominar	7	1	0	0	0
Percepção de utilidade					
O protótipo facilitaria na manutenção de software	5	3	0	0	0
O protótipo ajudaria na produtividade dos usuários	5	3	0	0	0
O protótipo reduziria o tempo de espera dos usuários	6	2	0	0	0
Intenção de comportamento					
Utilizaria o protótipo como <i>ChatBot</i> de um sistema de manutenção automática de código-fonte	6	2	0	0	0

Fonte: Elaborado pelo autor.

comportamento, referente ao uso do Hermes. Em respeito a facilidade de uso percebida pelos usuários, 75% ou mais dos entrevistados concordam que o Hermes é de fácil uso (75% concordam totalmente e 25% parcialmente), no caso de que não haveria dificuldades de se tornar um usuário hábil do Hermes, que achou que o Hermes é fácil de dominar e que não haveria necessidade de demasiado esforço para utilizar as funcionalidades oferecidas pelo Hermes, (87,5% concordam totalmente e 12,5% concordam parcialmente).

Com relação a percepção de utilidade, todos os participantes concordam que o Hermes facilitaria no cenário da manutenção em software e a aumentar a produtividade dos usuários (62,5% concordam totalmente e 36,5% concordam parcialmente). Com relação ao Hermes proporcionar uma redução de espera para efetuar manutenções em software, 75% dos usuários concordam totalmente enquanto 25% concordam parcialmente. Da mesma forma, também concordam que teriam intenção de usar o modelo como *ChatBot* para efetuar manutenções automática em software. Portanto, os dados coletados e analisados sugerem que o Hermes tem potencial de aceitação por pessoas com um perfil adequado com os dos participantes. Os resultados encorajam e mostram o potencial de usabilidade do modelo proposto em um ambiente real.

7 CONCLUSÃO

Foi apresentado Hermes, o qual utilizando técnicas de PLN juntamente com a inferência de regras lógicas utilizando ontologias para representação do domínio da aplicação a ser trabalhada, mostra que o modelo tem potencial para abranger todas as requisições possíveis dentro do escopo requerido pelo projeto MITRAS.

Como contribuição científica destaca-se a construção de um modelo que preenche a lacuna do projeto MITRAS apresentando com detalhes a construção do modelo e protótipo que possibilita ao usuário modificações automáticas em software utilizando linguagem natural, permitindo manutenções em um alto nível de abstração em código-fonte. Outra contribuição se dá pelo fato do Hermes ser um modelo genérico, possibilitando ser estendido e adaptado a outros trabalhos.

Como trabalhos futuros é possível explorar a utilização de redes neurais utilizando a base de dados (frases) geradas através dos experimentos, para treinar um novo *parser* de dependência voltado ao cenário de manutenção automática de software, possibilitando uma comparação com o modelo atual Hermes. Outro trabalho futuro seria treinar um novo reconhecedor de entidades que possa classificar melhor as classes gramaticais de determinadas palavras contidas dentro de um software. Além disso, é esperado que este trabalho seja o primeiro passo para construir uma interface que permita requisições para manutenção automática em código-fonte. Por fim se espera que os problemas percebidos no modelo encorajam outros pesquisadores a estender este estudo com diferentes estratégias da análise semântica de linguagem natural para o cenário de manutenção em software.

REFERÊNCIAS

ADWAIT RATNAPARKHI. A maximum entropy model for part-of-speech tagging. **In Proceedings of the Empirical Methods in Natural Language Processing Conference**, v. 1, n. 49, p. 133–142, 1996.

ALBERTO, C. et al. MDLText e Indexação Semântica aplicados na Detecção de SPAM nos Comentários do Youtube. **iSys - Revista Brasileira de Sistemas de Informação**, v. 10, n. 3, p. 49–73, 2017.

AMORIM, L. A. et al. Agente de Suporte à Decisão Multicritério com Soma Ponderada-Fuzzy em Gestão Pública Participativa: Um Estudo de Caso em Gestão Ambiental. **iSys - Revista Brasileira de Sistemas de Informação**, v. 8, n. 3, p. 28–41, 2016.

BAADER, F. et al. **The Description Logic Handbook: Theory, Implementation and Applications**. [s.l.] Cambridge University Press, 2003.

BRANTS, T. **TnT - A Statistical Part-of-Speech Tagger**2000Disponível em: <<http://arxiv.org/abs/cs/0003055>>

BRILL, E. Some Advances in Transformation-Based Part of Speech Tagging. 1994.

DE MARNEFFE, M.-C. et al. Universal Stanford Dependencies: A cross-linguistic typology. **In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)**, p. 4585–4592, 2014.

DESAI, A. et al. Program synthesis using natural language. **Proceedings - International Conference on Software Engineering**, v. 14-22-May-, n. ii, p. 345–356, 2016.

DUONG, H.; NGUYEN, T.; CHANDRA, S. SemFix : Program Repair via Semantic Analysis. **Icse**, p. 772–781, 2013.

FINKEL, J. R.; GRENAGER, T.; MANNING, C. Incorporating non-local information into information extraction systems by gibbs sampling. **in Acl**, n. 1995, p. 363–370, 2005.

FOUCART, A.; FRENCK-MESTRE, C. Language processing. **The Cambridge Handbook of Second Language Acquisition**, v. 349, n. 6245, p. 394–416, 2015.

GÓMEZ-PÉREZ, A. et al. **Ontological Engineering**. Amsterdam: Springer, 2004.

GULWANI, S. Automating string processing in spreadsheets using input-output examples. **SIGPLAN Not.**, v. 46, n. 1, p. 317–330, 2011.

IGWE, K.; PILLAY, N. Automatic Programming Using Genetic Programming. **Proceedings of the 2013 Third World Congress on Information and Communication Technologies (WICT 2013)**, p. 339–344, 2013.

KÜPSSINSKI, L. S.; GLUZ, J. C. MITRAS Intelligent Model for Software Transformation. **Computer on the Beach**, p. 921–923, 2018.

MANNING, C. et al. The Stanford CoreNLP Natural Language Processing Toolkit. **Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations**, p. 55–60, 2014.

MANNING, C. D.; SCHUTZE, H. **Foundations of Statistical Natural Language Processing**. Second ed. Cambridge - Massachusetts: The MIT Press, 1999.

MARANGUNIĆ, N.; GRANIĆ, A. Technology acceptance model: a literature review from 1986 to 2013. **Universal Access in the Information Society**, v. 14, n. 1, p. 81–95, 2015.

MARCUS, M. et al. The Penn Treebank: Annotating Predicate Argument Structure. **ARPA Human Language Technology Workshop**, p. 114–119, 1994.

NIVRE, J. et al. Universal Dependencies v1: A Multilingual Treebank Collection.

Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016), p. 1659–1666, 2016.

RODRIGUES, R. D. A.; AZEVEDO, L. G.; REVOREDO, K. BPMN2TEXT : A Language-Independent Framework to Generate Natural Language Text from BPMN models. **Revista Brasileira de Sistemas de Informação**, v. 9, n. 4, p. 38–56, 2010.

SAHA, R. K. et al. Elixir: Effective object-oriented program repair. **ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering**, p. 648–659, 2017.

SCHUSTER, S. et al. Generating Semantically Precise Scene Graphs from Textual Descriptions for Improved Image Retrieval. **Emnlp**, n. September, p. 70–80, 2015.

SEEBREGTS, C. J. et al. The OpenMRS Implementers Network. **International Journal of Medical Informatics**, v. 78, n. 11, p. 711–720, 2009.

TALANOV, M.; KREKHOV, A.; MAKHMUTOV, A. Automating programming via concept mining, probabilistic reasoning over semantic knowledge base of SE domain. **2010 6th Central and Eastern European Software Engineering Conference, CEE-SECR 2010**, p. 30–35, 2010.

TOUTANOVA, K. et al. **Feature-rich part-of-speech tagging with a cyclic dependency network** Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - NAACL '03. **Anais...2003** Disponível em: <<http://portal.acm.org/citation.cfm?doid=1073445.1073478>>

VAN HARMELEN, FRANK; LIFSCHITZ, VLADIMIR; PORTER, B. **Handbook of Knowledge Representation Edited by Frank van Harmelen**. 1. ed. [s.l.] Elsevier B.V., 2008.

WANG, S. I.; LIANG, P.; MANNING, C. D. Learning Language Games through Interaction. 2016.

WEIMER, W. et al. Automatically finding patches using genetic programming. **Proceedings - International Conference on Software Engineering**, p. 364–374, 2009.

WEIMER, W.; FRY, Z. P.; FORREST, S. Leveraging program equivalence for adaptive program repair: Models and first results. BT - 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013. p. 356–366, 2013.

ZAMANIRAD, S. et al. Programming bots by synthesizing natural language expressions into API invocations. **ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering**, p. 832–837, 2017.

APÊNDICE A – FORMULÁRIO TAM

Tabela 2: Questionário TAM

Item	Afirmação
1	Eu achei o Hermes fácil de usar
2	Não haveria dificuldade de me tornar um usuário hábil do Hermes
3	Eu achei o Hermes fácil de dominar.
4	Não é necessário demasiado esforço para utilizar as funcionalidades oferecidas pelo Hermes
5	Hermes facilitaria na manutenção de software.
6	O uso do Hermes ajudaria aumentar a produtividade dos usuários.
7	O uso do Hermes proporcionaria uma redução no tempo de espera para efetuar manutenções perfectivas e adaptativas em software.
8	Se for o caso, utilizaria o Hermes como chatbot de um sistema de manutenção automática de código-fonte.

Fonte: Elaborado pelo autor.

APÊNDICE B – TELA DO SISTEMA OPENMRS

Figura 8 – Tela de cadastro de paciente do sistema OpenMRS

OpenMRS Currently logged in as Super User | [Log out](#) | [My Profile](#) | [Help](#)

[Home](#) | [Find/Create Patient](#) | [Dictionary](#) | [Administration](#)

[Admin](#) | [Manage Patients](#) | [Manage Tribes](#) | [Find Patients to Merge](#) | [Manage Identifier Types](#)

Patient

[View Patient Dashboard](#) | [Merge this patient with another](#)

Patient Identifiers

OpenMRS Identification Number	Preferred <input checked="" type="checkbox"/>
<input type="button" value="Add New Identifier"/>	Identifier* <input type="text" value="2105KP-8"/>
	Identifier Type* <input type="text" value="OpenMRS Identification Number"/>
	Location* <input type="text" value="Wishard Hospital"/>
	Created By Super User - 18 January 2006 00:00:00 BRST
	Changed By Super User - 16 September 2018 19:55:34 BRT
	Deleted <input type="checkbox"/>

Patient Names

Lucy Turguit	Preferred <input type="checkbox"/>
<input type="button" value="Add New Name"/>	Given* <input type="text" value="Lucy"/>
	Middle <input type="text" value="Monyagi"/>
	Family Name* <input type="text" value="Turguit"/>
	Created By Super User - 18 January 2006 00:00:00 BRST
	Deleted <input type="checkbox"/>

Patient Addresses

Kamolo	Preferred <input type="checkbox"/>
<input type="button" value="Add New Address"/>	Address <input type="text"/>
	Section/Homestead <input type="text"/>
	Estate/Nearest Centre <input type="text"/>
	Sublocation <input type="text"/>
	Division <input type="text"/>
	Province <input type="text"/>
	Latitude <input type="text"/>
	Country <input type="text"/>
	Town/Village <input type="text" value="Kamolo"/>
	Location <input type="text"/>
	District <input type="text"/>
	Postal Code <input type="text"/>
	Longitude <input type="text"/>
	Created By Super User - 18 January 2006 00:00:00 BRST
	Deleted <input type="checkbox"/>

Fonte: Tela do sistema de cadastro de paciente do OpenMRS

APÊNDICE C – Script de ações para o usuário interagir com o protótipo para efetuar requisições de manutenção em software.

1. É solicitado ao usuário que utilize o protótipo para adicionar um novo campo que se tornou necessário no formulário do paciente. Com a visualização da tela, é solicitado que o usuário digite a alteração desejada na tela do protótipo para adicionar um novo campo chamado *nickname* no painel *Patient Names*, simulando uma necessidade de adicionar um novo dado pertinente ao cadastro.
2. O usuário não utiliza o campo *Longitude* referente ao painel *Patient Addresses* no preenchimento do formulário e gostaria de deletar este campo que não faz mais parte de sua rotina e está poluindo a interface do formulário. O usuário então deve solicitar ao protótipo a exclusão deste campo do formulário.
3. Neste cenário, o usuário será dirigido a mover o campo *Postal Code* para a primeira posição do painel de endereço, pois agora o sistema é capaz de completar alguns campos do painel endereço dependendo do código postal.
4. Houve uma mudança na regra de preenchimento do formulário e agora o usuário precisa fazer com que o campo *Family Name* seja o primeiro campo e também precisa deletar o campo *Middle* pois este campo não será mais utilizado no painel de nome do paciente.