Hermes: A Natural Language Interface Model for Software Transformation

Hermes: Um Modelo de Interface de Linguagem Natural para Transformação em Software

Michael William Chagas, Kleinner Farias,
Lucian Gonçales, Lucas Kupssinskü
Universidade do Vale do Rio dos Sinos - UNISINOS
São Leopoldo, Brasil
mwmaciel@edu.unisinos.br,kleinnerfarias@unisinos.br
{lucianj,lkupssinsku}@edu.unisinos.br

João Carlos Gluz
Instituto Federal do Rio Grande do Sul - IFRS
Canoas, Rio Grande do Sul, Brasil
jcgluz@gmail.com

ABSTRACT

Software maintenance is a costly task and error-prone for both software developers and users as well. By knowing how and what software requirements need to be changed, end users could perform maintenance assisted by tools. However, current literature lacks for tools that support automated maintenance in real-world scenarios and allow users interaction via natural language. Even worse, the current tools are unable to understand the semantic of requests, as well as perform the necessary transformations in the maintenance software. This paper, therefore, proposes Hermes, a natural language interface model for software transformation. It combines computational linguistics techniques and logic programming to perform automated maintenance requests in software. Hermes interacts with end user through state of the art language parsers and domain ontologies by interpreting the semantics of changes requests to build a typed graph that change the software. Hermes was evaluated through an empirical study with 8 participants to investigate its performance, the level of acceptance, and usability. The collected data show that Hermes was accurate, producing a high elevated correctness number of hits by finding correct transformations and has been highly accepted by the users. The results are encouraging and show the potential for using Hermes to properly produce software maintenance requests.

CCS CONCEPTS

• **Human-centered computing** → **Natural language interfaces**;
• **Information systems** → *Ontologies*.

KEYWORDS

Natural Language Processing, Parsing, Ontologies

ACM Reference Format:
Michael William Chagas, Kleinner Farias, Lucian Gonçales, Lucas Kupssinskü and João Carlos Gluz. 2019. Hermes: A Natural Language Interface Model for Software Transformation. In *XV Brazilian Symposium on Information Systems (SBSI'19), May 20–24, 2019, Aracaju, Brazil*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3330204.3330253

SBSI'19, May 20–24, 2019, Aracaju, Brazil

ACM ISBN 978-1-4503-7237-4/19/05…$15.00
https://doi.org/10.1145/3330204.3330253

1 INTRODUÇÃO

O desenvolvimento de sistemas de informação tem ocorrido em ambientes de negócios cada vez mais instáveis. Essa realidade tem tornado os requisitos de software voláteis e complexos, exigindo ciclos de desenvolvimento e manutenção mais curtos. Neste contexto de alterações constantes, a manutenção de software tem se tornado uma atividade altamente custosa e propensa a erros tanto para desenvolvedores de software quanto para usuários finais. Os sistemas de informação são utilizados pelos usuários finais nas empresas, os quais identificam pontos de melhorias no sistema como, por exemplo, a adição de um novo campo no formulário de cadastro de clientes. Os usuários finais então geram requisições de mudança para os desenvolvedores do sistemas de informação, 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 (LN). Processamento de linguagem natural (PLN) tem sido amplamente utilizado para tratar variados desafios como, por exemplo, a criação de *parsers* de processos para LN [12], detecção de padrões em ambiente web [14], identificação de classes UML utilizando PLN [1], entre muitos outros. Apesar disso, 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 o Hermes, um modelo de interface de LN para transformação em software. Ele combina técnicas de linguística computacional e programação em lógica para realizar requisições de manutenção automática em software. O Hermes interage com o usuário final através de LN, 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 [8] trata-se de um modelo inteligente que utiliza o formalismo de grafos para representar e realizar transformações no software em manutenção. O modelo MITRAS inicialmente utiliza esforços para representar grafos de transformação de alto nível (o qual não envolve edição de código-fonte dentro de métodos) e logo se encaixa bem em manutenções do tipo perfectivas e adaptativas. Dois tipos de alto nível em requisições para transformação em software foram considerados pelo MITRAS: a) "adicionar mais do mesmo", permite inserir novas entidades, atributos e processos em software, ou mudanças na interface do usuário e b) "ver menos do mesmo", oculta entidades e atributos da apresentação do modelo para diminuir a carga cognitiva do usuário da interface.

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 do Hermes para produzir adequadamente solicitações de manutenção de software.

Este artigo é organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica. A Seção 3 discute os trabalhos relacionados. A Seção 4 introduz o modelo proposto. A Seção 5 descreve os aspectos de implementação do Hermes. A Seção 6 apresenta a avaliação. Por fim, a Seção 7 discute algumas considerações finais e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Stanford CoreNLP. O objetivo da ciência da linguagem é poder caracterizar e explicar as múltiplas observações linguísticas encontradas na comunicação escrita. O 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 fornece um conjunto de ferramentas de tecnologia de linguagem humana [9], e será a base para trabalhar no processo de *parsing* do texto de requisições de mudança representadas em LN. Um diferencial deste *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.

Na esfera do PLN, 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. Considerando $G = w_1, w_2, \ldots,$ uma frase onde w_i representam cada palavra, $G = (V, A)$ é o grafo de dependências da frase sendo que $V = (w_1, w_2, \ldots, w_n)$ são os vértices e $A \subseteq V \times V$ é o conjunto de arestas que representa as dependências entre as palavras. A Figura 1 ilustra um grafo de dependências dirigido, gerado por uma frase. A direção das arestas representa o significado de uma palavra de origem para uma de destino. Um exemplo seria 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.

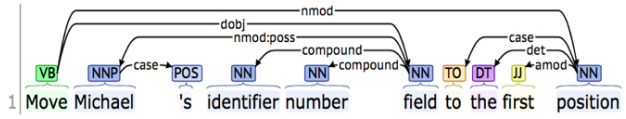

Figura 1: Exemplo de um grafo de dependências

MITRAS. O MITRAS permite que usuários sem qualquer conhecimento em programação possam efetuar requisições para modificações em software através de um sistema inteligente de manutenção automática. Dentro das opções de manutenção propostas pelo modelo MITRAS, é possível verificar que as transformações são executadas em um nível alto de abstração, mudando classes, entidades, atributos e relações de um sistema. 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 PLN. 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 [6], programação genética por heurística [7], sintetização [11], conceito de mineração [15] e técnicas de geração e validação para reparo [17].

3 TRABALHOS RELACIONADOS

Esta seção realiza 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.

3.1 Análise dos trabalhos

Zamanirad *et al.* [18] desenvolveu o trabalho *BotBase*, o qual emprega técnicas de síntese de expressões de usuários em LN para efetuar chamadas concretas de *API's* apoiados por um grafo de uma base de conhecimento de *API's*. Esta base contem informações sobre diversas *API's*, suas declarações, expressões, parâmetros e possíveis valores. Logo, usuários utilizando LN podem efetuar chamadas de *API's* de uma forma mais direta.

Schuster *et al.* [13] 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 LN, 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.

Desai *et al.* [4] desenvolveu um *framework* que mapeia requisições em LN para a sintetização 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.* [4] converte cada palavra de entrada de usuário em um ou mais terminais usando LN 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.

Wang, Liang & Manning [16] desenvolveu a ideia de humanos cumprir alguma tarefa utilizando o conceito de linguagem de jogos. 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 LN. 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.

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 a fim de gerar/alterar sofware, se havia a troca de mensagens (conversação) entre usuário e máquina e se havia uma ação em alto nível que executasse um plano.

3.2 Análise comparativa dos trabalhos

A Tabela 1 apresenta a análise comparativa dos trabalhos, a qual é feita utilizando cinco critérios de comparação. O primeiro critério apresenta informação se os trabalhos geram código-fonte para software existente ou não e embora três trabalhos [18][4][16] sejam capazes de gerar software, nenhum deles trata de alteração de software. Este segundo critério difere Hermes dos demais trabalhos no critério de alterar algum software existente através do uso de PLN para manutenção em software. Pode-se observar no terceiro critério que os trabalhos relacionados não oferecem suporte a conversação no cenário de manutenção. Referente ao tópico sobre as ontologias, somente o trabalho de Desai *et al.* [4] utiliza uma base de dados contendo informações sobre voos para posteriormente comparar *queries* de entrada com esta base de dados. Enquanto três artigos [18][4] [16] apresentaram soluções de comparação léxica com alguma base de dados própria ou de terceiros, o artigo de Schuster *et al.* [13] usa técnicas de *parsing* estatístico, próximo a ideia do Hermes, mas em contraste a aplicações para o cenário de manutenção em software.

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.* [13], 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 LN para transformação em software. Para isso, a Seção 4.1 apresenta a arquitetura geral do modelo, detalhando os módulos da arquitetura. A Seção 4.2 introduz o módulo de PLN. Por fim a Seção 4.3 descreve o módulo de transformação semântico bem como as diferentes Ontologias empregadas.

4.1 Arquitetura do modelo

Com base nos grafos de transformação 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 PLN*: consiste em um conjunto de ferramentas do CoreNLP de Stanford oferecidas por Manning *et al.* [9] para análise sintática das frases digitadas.
- *Módulo analisador semântico*: 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*: módulo externo, que não é objeto desse artigo, 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 software estas transformações através de técnicas de injeção em código-fonte. A Figura 2 apresentada a visão geral da arquitetura.

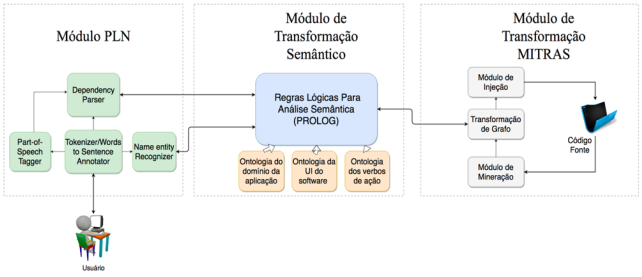

Figura 2: Arquitetura do Hermes

O software OpenMRS[1], um sistema de gerenciamento em saúde, foi utilizado para testar o Hermes. O OpenMRS foi escolhido por ser código-fonte aberto, implementado em linguagem orientada a objetos (Java), possui uma comunidade ativa (+180KLOC), e uma aplicação diferente das conhecidas e trabalhadas previamente pelos autores deste trabalho.

4.2 Módulo de Processamento de Linguagem Natural

O conjunto de ferramentas do CoreNLP de Stanford proposto por Manning *et al.* [9] é capaz de efetuar o parse de uma sentença e

[1] https://openmrs.org

Tabela 1: Quadro comparativo dos trabalhos relacionados

Trabalhos	Gera Software	Altera Software	Suporte a conversação	Utiliza ontologias como base	Tipo de abordagem de *parsing*
Zamanirad *et al.* [18]	Sim	Não	Não	Não	*Parser* com comparação léxica
Schuster *et al.* [13]	Não	Não	Não	Não	*Parser estatístico*
Desai *et al.* [4]	Sim	Não	Não	Sim	*POS Tagging* e *parser* com comparação léxica
Wang, Liang & Manning [16]	Sim	Não	Não	Não	*Parser* com comparação léxica

gerar uma árvore da sintaxe correspondente e possui várias outras funcionalidades as quais são utilizadas pelo módulo de PLN.

Grafo de dependências universal de de Marneffe *et al.* [3] 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.

Foi utilizada a representação básica da saída do *Parser* de Stanford [2]. Inicialmente, o usuário entra com uma frase e então o *Parser* infere as relações entre cada palavra já classificadas gramaticalmente e entrega um *grafo sintático de dependências* como resultado. Após, três etapas adicionais são processadas no módulo de transformação semântico para resolver problemas de classificação.

4.3 Módulo de transformação semântico

Este módulo trabalha com os dados gerados pelo Parser de Stanford, seguindo um fluxo de tratativas utilizando Ontologias para compreender a semântica por trás das requisições de usuários.

A interação em LN passa pelo *parser* e é avaliada em relação as Ontologias. O processo de popular as Ontologias foi efetuada manualmente e um processo semi-automático ou automático para o povoamento de Ontologias (ex. [5]) seria um avanço no processo de manutenção automática em geral.

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 LN dando sequência a produção do pipeline.

4.3.1 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 paciente. A ontologia então será a base das informações "campo" e "painel", que são conceitos referentes a entidades da interface gráfica do software.

4.3.2 Ontologia da interface gráfica do software (UIO). A Ontologia da interface gráfica do software identifica qual o tipo de elemento da interface 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"*, o qual o usuário enxerga.

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.

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*.

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 sobre os verbos presentes no grafo de dependências mapeado, elegendo 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.

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:

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

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.

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.

5 IMPLEMENTAÇÃO

A construção do protótipo foi dividida em duas etapas, sendo a primeira o pré-processamento das frases de entrada do usuário e a análise sintática probabilística destas frases que gerou um grafo de dependências para a segunda etapa, a qual através de regras lógicas, utilização de ontologias combinadas e a utilização de agentes, permitem a comunicação em LN.

Para a interação entre o usuário e o protótipo foi desenvolvida uma interface web.

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.

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. 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*.

A aplicação foi desenvolvida orientada a agentes. Foram construídos três agentes que são iniciados ao rodar o *shell* do programa: *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 PLN. 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 recebe os dados do *nlp_agent* e efetua as transformações em grafo. Este agente é externo ao escopo da arquitetura do Hermes.

6 AVALIAÇÃO

Esta Seção apresenta a metodologia que avalia o Hermes e apresenta os resultados obtidos. A Seção 6.1 descreve o objetivo e as perguntas de pesquisa. Na Seção 6.2 são introduzidos os questionários para prover a avaliação do Hermes. Já a Seção 6.3 fala sobre os cenários de avaliação e sobre a seleção dos participantes. Na Seção 6.4 é apresentado o design experimental. Por fim, na Seção 6.5 é feita a análise dos dados coletados.

6.1 Objetivo e perguntas de pesquisa

O estudo realizado na análise semântica da LN de requisições de usuários para manutenções perfectivas e adaptativas em software tem 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 requisições de manutenção em software no escopo do MITRAS?
- **RQ2:** Qual a aceitação do Hermes pelos usuários de software?

6.2 Questionários

Para responder a estas questões de pesquisa foram elaborados questionários e avaliação através de cenários respondidos pelos participantes (introduzidos na subseção 6.3). Os dois questionários elaborados são detalhados a seguir.

Questionário 1: Perfil dos participantes. Tem como objetivo coletar dados referente às 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 TAM. 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), [10]. 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. As 4 primeiras questões referem-se à facilidade de uso do 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". As

questões formuladas lidam com tópicos de percepção de facilidade de uso, percepção de utilidade e intenção de comportamento.

6.3 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[2]. 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 duas transformações em grafo do MITRAS empregadas no software OpenMRS. A tela do Hermes pode ser vista na Figura 3.

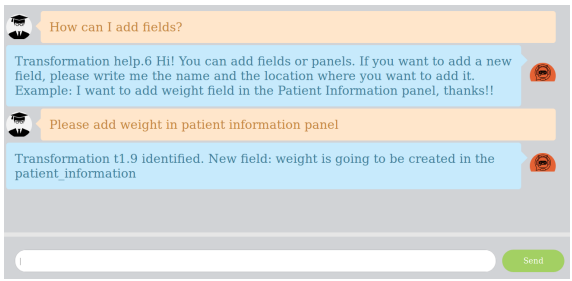

Figura 3: Tela de Interação com o Hermes

Para coletar os dados, foram criados quatro cenários abrangendo as duas transformações possíveis pelo usuário, brevemente descritos na Tabela 2. 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.

Tabela 2: Tarefas relacionadas aos cenários do experimento

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

Devido à complexidade em selecionar usuários reais do software OpenMRS, optou-se por selecionar usuários acadêmicos com formação em áreas da TI, 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 programa de pós-graduação em computação aplicada da Unisinos. Também foi escolhido 1 participante do curso de Ciência da Computação também da Unisinos e 2 graduados em Análise de Sistemas, logo é possível obter usuários com diferentes perfis de conhecimento e experiência. Uma interface de edição de um cadastro de paciente foi escolhida dentro do software OpenMRS, a qual abrange as três ações possíveis de execução pelo usuário.

[2]https://openmrs.org

O experimento foi desenhado similar a um exercício prático de laboratório. Cada participante recebeu o mesmo treinamento para desenvolver o processo experimentai. O processo é apresentado na próxima seção.

6.4 Processo experimental

A Figura 4 apresenta o processo experimental, formado por uma lista de atividades agrupadas em três fases descritas a seguir.

Fase 1 Esta fase apresenta 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 para ajudar os usuários a entender melhor a utilidade do Hermes e verificar se entenderam a técnica proposta.

Fase 2 Esta fase executa o Hermes. Os participantes executam 4 cenários e produzem um *script* desejado (dados de entrada). Este *script* de ações (transformações) é executado 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 São duas atividades. A primeira coleta informações sobre o perfil e opinião dos usuários que irão responder uma lista de perguntas (dados de entrada) e as respostas são a saída dos dados coletados como resultado desta atividade. A segunda atividade aplica o questionário TAM (dados de entrada). Os participantes recebem uma lista de perguntas sobre a percepção da facilidade de uso, percepção de utilidade e intenção de comportamento relacionados ao Hermes. Dados qualitativos (saída de dados) são gerados a partir da usabilidade e aceitação do Hermes pela perspectiva dos usuários. Os participantes executam todas as fases para evitar qualquer inconsistência no processo experimental. Os dados coletados serão discutidos na Seção 6.5.

6.5 Resultados

6.5.1 Resultados do perfil dos usuários. A Tabela 3 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 (66,6%) informaram que trabalharam mais de 2 anos com desenvolvimento. Mais de 60% dos participantes informaram que já necessitaram

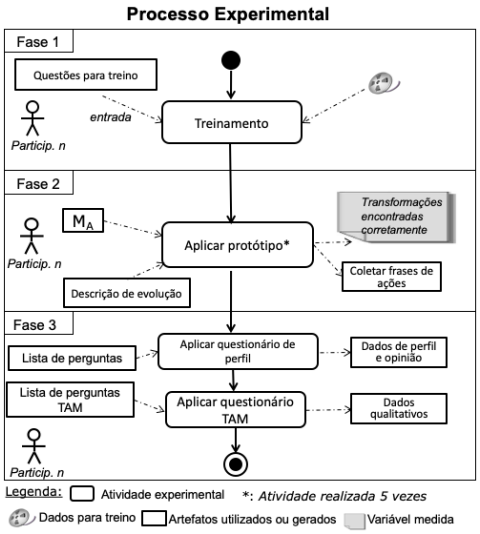

Figura 4: Processo experimental utilizado.

Tabela 3: Resultados do perfil dos participantes

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*	25.0%
	>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	67.5%
	Não	3	37.5%
Já utilizou sistemas de ChatBot	Sim	5	67.5%
	Não	3	37.5%
Nível de conhecimento em inglês	Pouco	3	37.5%
	Intermediário	0	0.0%
	Avançado	5	67.5%
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%

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.

6.5.2 RQ1: Resultados que encontraram corretamente transformações válidas. A Tabela 4 apresenta os dados coletados que avaliam se os participantes conseguiram expressar em LN ações de manutenção em software. 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, foi possível verificar o quanto a ferramenta é capaz de encontrar transformações válidas. O alto número de transformações encontradas corretamente, sugere que o protótipo é apropriado para o objetivo proposto. No pior caso (Transformação 4), 62,5% das transformações foram encontradas corretamente.

Tabela 4: Resultado das transformações encontradas

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

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 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 não foram encontradas 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.

6.5.3 RQ2: Resultados referentes à aceitação tecnológica. Através da aplicação do questionário TAM foram coletadas informações sobre percepção de facilidade de uso, percepção de utilidade e intenção de comportamento, referente ao uso do Hermes. A Tabela 5 traz informações sobre os dados coletados. Em respeito à 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 existiriam 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 à 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, os participantes também concordam que teriam intenção de usar o modelo como

Tabela 5: Resultados referentes ao questionário TAM.

	Concordo Totalmente	Concordo Parcialmente	Neutro	Discordo Parcialmente	Discordo Totalmente
Percepção de facilidade de uso					
Achei Hermes fácil de usar	6	2	0	0	0
Achei Hermes fácil de aprender	7	1	0	0	0
Achei Hermes fácil de dominar	7	1	0	0	0
Percepção de utilidade					
Hermes facilitaria na manutenção de software	5	3	0	0	0
Hermes ajudaria na produtividade dos usuários	5	3	0	0	0
Hermes reduziria o tempo de espera dos usuários	6	2	0	0	0
Intenção de comportamento					
Utilizaria Hermes como ChatBot de manutenção automática de código-fonte	6	2	0	0	0

ChatBot para efetuar manutenções automática em código-fonte. 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 E TRABALHOS FUTUROS

Este artigo apresentou o Hermes, o qual utilizando técnicas de processamento de linguagem natural juntamente com a inferência de regras lógicas e com Ontologias para representação do domínio da aplicação a ser trabalhada, apresenta resultados 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 LN, 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 para treinar um novo parser de dependência e um reconhecedor de entidades, voltado ao cenário de requisições de manutenção em software, possibilitando uma comparação com o modelo atual Hermes.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior -Brasil (CAPES) - Finance Code 001, the Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

REFERENCES

[1] Daniel Antonio Conte and Jean Hauck. 2016. Identificação Automatizada de Classes Utilizando Processamento de Linguagem Natural. Brazilian Computer Society. https://doi.org/10.13140/RG.2.1.4902.1687

[2] Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 740–750.

[3] Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology.. In *LREC*, Vol. 14. 4585–4592.

[4] Aditya Desai, Sumit Gulwani, Vineet Hingorani, Nidhi Jain, Amey Karkare, Mark Marron, Subhajit Roy, et al. 2016. Program synthesis using natural language. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 345–356.

[5] Carla Faria and Maria Gutierrez. 2011. Um Processo Semi-Automático para o Povoamento de Ontologias a partir de Fontes Textuais. *iSys - Revista Brasileira de Sistemas de Informação* 3, 1 (2011).

[6] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 317–330.

[7] Kevin Igwe and Nelishia Pillay. 2013. Automatic programming using genetic programming. In *Information and Communication Technologies (WICT), 2013 Third World Congress on*. IEEE, 337–342.

[8] Lucas Silveira Küpssinski and João Carlos Gluz. 2018. MITRAS Inteligent Model for Software Transformation. *Computer on the Beach* (2018), 921–923.

[9] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010

[10] Nikola Marangunić and Andrina Granić. 2015. Technology acceptance model: a literature review from 1986 to 2013. *Universal Access in the Information Society* 14, 1 (2015), 81–95.

[11] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. 2013. Semfix: Program repair via semantic analysis. In *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 772–781.

[12] Raphael Rodrigues, Leonardo Azevedo, and Kate Revoredo. 2016. BPM2Text: A language independent framework for Business Process Models to Natural Language Text. *iSys - Revista Brasileira de Sistemas de Informação* 9, 4 (2016).

[13] Sebastian Schuster, Ranjay Krishna, Angel Chang, Li Fei-Fei, and Christopher D Manning. 2015. Generating semantically precise scene graphs from textual descriptions for improved image retrieval. In *Proceedings of the fourth workshop on vision and language*. 70–80.

[14] Renato Silva, Túlio Alberto, Tiago Almeida, and Akebo Yamakami. 2017. MDLText e Indexação Semântica aplicados na Detecção de Spam nos Comentários do YouTube. *iSys - Revista Brasileira de Sistemas de Informação* 10, 3 (2017).

[15] Maxim Talanov, Andrei Krekhov, and Aidar Makhmutov. 2010. Automating programming via concept mining, probabilistic reasoning over semantic knowledge base of SE domain. In *Software Engineering Conference (CEE-SECR), 2010 6th Central and Eastern European*. IEEE, 30–35.

[16] Sida I Wang, Percy Liang, and Christopher D Manning. 2016. Learning language games through interaction. *arXiv preprint arXiv:1606.02447* (2016).

[17] Westley Weimer, Zachary P Fry, and Stephanie Forrest. 2013. Leveraging program equivalence for adaptive program repair: Models and first results. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 356–366.

[18] Shayan Zamanirad, Boualem Benatallah, Moshe Chai Barukh, Fabio Casati, and Carlos Rodriguez. 2017. Programming bots by synthesizing natural language expressions into API invocations. In *Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on*. IEEE, 832–837.