Gustavo Costa Duarte

Uma Ferramenta Web Baseada em Ontologias Operacionais para Visualização da Relação entre Requisitos

Vitória, ES

2021

Gustavo Costa Duarte

Uma Ferramenta Web Baseada em Ontologias Operacionais para Visualização da Relação entre Requisitos

Monografia Apresentada ao Curso de Ciência da Computação da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Departamento de Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Coorientador: Bruno Borlini Duarte, M.Sc

Vitória, ES

2021

Gustavo Costa Duarte
Uma Ferramenta Web Baseada em Ontologias Operacionais para Visualização da Relação entre Requisitos/ Gustavo Costa Duarte. – Vitória, ES, 2021-
79 p. : il.; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Monografia de Conclusão de Curso – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Departamento de Informática , 2021.

1. Software Systems 2. Ontologies. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Uma Ferramenta Web Baseada em Ontologias Operacionais para Visualização da Relação entre Requisitos

CDU 02:141:005.7

Gustavo Costa Duarte

Uma Ferramenta Web Baseada em Ontologias Operacionais para Visualização da Relação entre Requisitos

Monografia Apresentada ao Curso de Ciência da Computação da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, 14 de Outubro de 2021:

Prof. Dr. Vítor E. Silva Souza
Orientador

Bruno Borlini Duarte, M.Sc
Universidade Federal do Espírito Santo
Co-orientador

Profª. Patrícia Dockhorn Costa, Ph.D.
Universidade Federal do Espírito Santo

Jussara Teixeira
Instituto de Tecnologia da Informação e Comunicação do Espírito Santo

Vitória, ES
2021

Dedico este trabalho a todos que direta ou indiretamente contribuíram para que eu concluísse minha graduação.

Agradecimentos

Gostaria de agradecer a todos que de alguma forma me ajudaram durante todo o meu percurso acadêmico. Primeiramente a Deus por todas as orações respondidas, por não me deixar desanimar e por ter me dado forças para conseguir concluir a minha graduação.

A meus pais, Sandro Duarte e Rosemere Duarte, agradeço por todo amor e carinho. Por mesmo de longe, durante todo o curso me deram muito apoio e atenção, mas também agradeço por terem me ensinado o caminho onde eu deveria andar. Também a minha irmã, Larissa Duarte, agradeço pelo carinho, atenção e pela nossa grande amizade.

A Jéssica, que esteve comigo durante toda a graduação, agradeço por todo amor, carinho, paciência, companheirismo e principalmente por ser minha minha incentivadora número um. Agradeço também, a sua família que sempre me acolheu de braços abertos.

Aos grandes irmãos que eu ganhei na UFES, Vinícius Berger e Leonardo Nascimento, obrigado pelos estudos até tarde na biblioteca, quando as provas se aproximavam. Pelas inúmeras "duplas de três" nos trabalhos para "vencermos" cada período. Além de agradecer a todos os meus colegas de classe, o companheirismo de todos foi fundamental, para que todos chegassem ao final do curso, sempre trabalhando em equipe nas matérias. A meu amigo Daniel de Assis, que teve um papel chave, me apoiando na reta final do curso, com seus conhecimentos.

Gostaria de agradecer a todos os professores com quem tive a possibilidade de estudar. O conhecimento passado por vocês tem um valor inestimável. Em especial gostaria de agradecer ao meu orientador, Vitor Souza, por me aceitar como orientando, pelo apoio com as informações sobre o projeto, pela compreensão e paciência.

E gostaria de agradecer ao meu tio, Bruno Borlini Duarte, obrigado por ter me perguntado como andava o meu projeto, quando eu já pensava em desistir. Você me apresentou uma nova direção naquele dia. A sua orientação foi sem igual. Obrigado por todo o tempo dedicado a mim, todo o apoio, toda a direção e suporte até aqui.

"Agir, eis a inteligência verdadeira. Serei o que quiser. Mas tenho que querer o que for. O êxito está em ter êxito, e não em ter condições de êxito. Condições de palácio tem qualquer terra larga, mas onde estará o palácio se não o fizerem ali? (Fernando Pessoa)

Resumo

O software desempenha um papel essencial na sociedade contemporânea e se tornou indispensável em muitos contextos de nossas vidas. Este, é um artefato em constante crescimento, de complexidade, criticidade e grau de heterogeneidade. Logo, os softwares tendem a gerar um alto custo de desenvolvimento e podem então dar início a muitos problemas no ciclo de vida dos mesmos.

A Gerência de Requisitos de Software, processo da Engenharia de Requisitos que é responsável pela mudança dos requisitos, se faz necessária enquanto os softwares vão evoluindo. Uma de suas etapas mais importantes é a Rastreabilidade de Requisitos.

Há quase 25 anos a rastreabilidade de requisitos vem sendo reconhecida nas principais literaturas científicas, como uma atividade capaz de melhorar a qualidade de gestão de sistemas de softwares. A rastreabilidade necessariamente deve ser organizada em uma estrutura de modelagem adequada.

Para este trabalho, foi desenvolvido um sistema para a visualização da rastreabilidade de requisitos, baseando-se em modelos de referência já existentes e seguindo um padrão de arquitetura MVC com *frameworks* Java conhecidos e consolidados.

Palavras-chave: Sistemas de Software, Requisitos de Software, Rastreabilidade de Requisitos, Ontologias, ROSS, OSDEF, UFO.

Abstract

Software plays an essential role in contemporary society and has become indispensable in many contexts of our lives. This, is an artifact in constant growth, of complexity, criticality and degree of heterogeneity. Therefore, software tends to generate a high development cost and can then trigger many problems in its lifecycle.

The Software Requirements Management, a process of Requirements Engineering, that is responsible for changing requirements, is necessary as the software evolves. One of its most important steps is Requirement Traceability.

For almost 25 years, requirements traceability has been recognized in the main scientific literatures as an activity capable of improving the quality of software systems management. Traceability must necessarily be organized in a suitable modeling structure.

For this work, a system for the visualization of requirements traceability was developed. Building on existing reference models. Following an MVC architecture pattern. And Using some well known Java frameworks.

Keywords: Software Systems, Software Requirements, Requirements Traceability, Ontologies, ROSS, OSDEF, UFO.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFO	Unified Foundational Ontology
ROSS	Reference Ontology of Software Systems
OSDEF	Ontology of Software Defect Errors and Failures
SABiO	Systematic Approach for building Ontologies
CQ	Competency Question
BREQ	Business Requirement
STREQ	Stakeholder Requirement
SYSREQ	System Requirement
PROGREQ	Program Requirement
WWW	World Wide Web
HTML	HyperText Markup Language
XML	eXtensible Markup Language
SPARQL	SPARQL Protocol and RDF Query Language
RDF	Resource Description Framework
OWL	Ontology Web Language
CDI	Contexts and Dependency Injection
JSF	JavaServer Faces
XHTML	Extensible HyperText Markup Language

Sumário

1 Introdução

Esse capítulo apresenta a introdução deste trabalho, que consiste do desenvolvimento de uma aplicação Web, com foco em uso de modelos para rastreabilidade de requisitos.

1.1 Contexto e Motivação

O software desempenha um papel essencial na sociedade contemporânea e se tornou indispensável em muitos contextos de nossas vidas, papel este que motiva uma série de iniciativas de pesquisa destinadas a compreender a natureza do software e sua relação conosco. Uma concepção compartilhada nessas iniciativas é que o software é um artefato (social) complexo (WANG et al., 2014a; WANG et al., 2014b; IRMAK, 2013). Essa noção vem do fato de que um sistema de software moderno pode ser entendido como a combinação de uma série de elementos interagentes, especificamente organizados para fornecer um conjunto de funcionalidades para atender a propósitos humanos particulares (ISO, IEC, 2017b; BOURQUE; FAIRLEY et al., 2014).

Além disso, o software está em constante crescimento, não apenas em medidas simples como o número de linhas de código, mas também de acordo com outros fatores, como complexidade, criticidade e grau de heterogeneidade (CHENG; ATLEE, 2007). Este crescimento se dá devido a vários fatores, que levam a mudanças nos requisitos. Uma das principais fontes geradoras de alteração é o próprio conjunto de interessados (*stakeholders*), que inicialmente não tem muita clareza nos objetivos a alcançar, e a medida que o desenvolvimento evolui vai descobrindo novas possibilidades ou funcionalidades a incluir no projeto. Um adequado gerenciamento de requisitos necessita da rastreabilidade dos requisitos para controlar e documentar as modificações, além disso, a rastreabilidade ajuda a estimar variações em cronogramas e alterações nos custos de desenvolvimento. Sem a rastreabilidade, por exemplo, a equipe de desenvolvimento poderia não saber a extensão das mudanças e, com isso, não saber valorar as mudanças requeridas.

Além de que o crescimento do software faz com que o software fique muito mais caro para se desenvolver e pode dar início a muitos problemas em seu ciclo de vida. Neste contexto, conceitos como problema, anomalia, bug e falha são geralmente tratados indistintamente, embora possam ter semânticas ontológicas diferentes. Esse uso informal, por mais comum e prático que seja em nossas conversas cotidianas, pode ser fonte de problemas de ambiguidade e falsa concordância, uma vez que o conceito de anomalia é frequentemente sobrecarregado, portanto, referindo-se a entidades com naturezas ontológicas distintas. Em um ambiente mais formal, essa sobrecarga de construto pode levar a problemas e perdas de comunicação. Por isso é importante ter uma forma precisa de classificar os diferentes

tipos de anomalias de software (DUARTE et al., 2018).

O Guia de Conhecimento em Engenharia de Software (SWEBoK) (IRMAK, 2013) enfatiza, por exemplo, a necessidade de um consenso sobre a caracterização de anomalias e discute como uma classificação bem fundamentada pode ser usada em auditorias e análises de produtos. Além disso, o modelo CMMI (SEI/CMU, 2010) preconiza que as organizações devem criar ou reutilizar alguma forma de método de classificação de defeitos, erros e falhas. Também sugere o uso de um índice de densidade de defeito para muitos produtos de trabalho que fazem parte do processo de desenvolvimento de software.

Um esquema de classificação adequado pode permitir o desenvolvimento de diferentes tipos de perfis de anomalias, que podem ser produzidos como um indicador da qualidade do produto. Além disso, classificar sistematicamente anomalias de software que podem ocorrer em tempo de design ou tempo de execução é uma rica fonte de dados, que pode ser usada para melhorar processos e evitar a ocorrência de anomalias em projetos futuros (IEEE, 2009). Por fim, defeitos, erros e falhas têm um impacto negativo em aspectos importantes do software, como confiabilidade, eficiência, custo geral e, em última análise, vida útil. Portanto, uma melhor compreensão da natureza ontológica desses conceitos e como eles se relacionam com outros artefatos de software (por exemplo, requisitos, solicitações de mudança, relatórios e casos de teste) pode melhorar a forma como uma organização lida com esses problemas, reduzindo custos com atividades como gerenciamento de configuração e manutenção de software (DUARTE et al., 2018).

Embora existissem algumas propostas para classificar diferentes termos para anomalias de software, não existia um modelo de referência ou teoria que explicasse a natureza das diferentes anomalias de software. Para suprir essa lacuna, foi proposta uma Ontologia de referência de Defeitos, Erros e Falhas de Software (OSDEF) (DUARTE et al., 2021), que será mais tarde descrita na Seção 3.4. Sabendo da importância de analisar tais anomalias, em termos do risco que a sua presença acarreta para os sistemas de software e, em particular, para estes sistemas como portadores de valor para algum agente, foi elaborada uma Ontologia de Referência em Sistemas de Software (ROSS) (DUARTE et al., 2021), sobre a relação entre sistemas e outros artefatos de software em diferentes níveis de requisitos, que pode se conectar a OSDEF quando se leva em consideração que uma falha a nível de programa afeta os objetivos da organização, podendo inclusive inviabilizar a existência do software. A ROSS também será descrita posteriormente na Seção 3.3.

Essas ontologias são modelos, que servirão de referência para rastreabilidade de requisitos da aplicação utilizadas como exemplo, uma vez que na literatura propõe-se que para um funcionamento adequado de rastreabilidade é necessário que seja estabelecido um esquema formal de descrição das informações detalhadas, que facilite o raciocínio e que a aplicação de tecnologias semânticas promovam a utilização de metodologias e técnicas de rastreabilidade semântica, acarretando no gerenciamento de complexas relações entre

itens de dados (ALONSO-RORÍS et al., 2016). Além disso, Espinoza e Garbajosa (2011) sugerem que, para ter utilidade, a rastreabilidade necessariamente deve ser organizada em uma estrutura de modelagem adequada.

Como sugere o próprio significado, os chamados modelos de referência são *templates* criados sobre uma demanda específica com a finalidade de reduzir significativamente a tarefa de criar outros novos modelos e/ou sistemas restritivos para aplicações. O funcionamento consiste na parte onde o usuário final do modelo de referência seleciona o que é ou não relevante do modelo e as adapta ao problema que necessita da solução, configurando assim uma resposta a partir das partes adaptadas. Em outras palavras, modelos de referência podem ser compreendidos como uma abstração representativa de um conhecimento bem consolidado dentro de um domínio (RAMESH; JARKE, 2001).

1.2 Objetivos

O que o presente trabalho propõe é a criação de uma ferramenta gráfica baseada em tecnologia Java, na plataforma Web, que realize a rastreabilidade de requisitos utilizando como modelos de referência as ontologias ROSS e OSDEF (DUARTE et al., 2021), apresentando tal rastreabilidade de forma gráfica, para que seja visível de uma maneira mais amigável para o usuário final a relação entre requisitos.

Esse objetivo geral pode ser decomposto nos seguintes objetivos específicos.

- Compreensão dos requisitos do sistema e das ontologias ROSS e OSDEF;
- Capacitação para usar a plataforma de desenvolvimento do Jakarta EE (antigo Java EE) e seus vários recursos;
- Desenvolvimento do sistema, incluindo documentação da arquitetura e projeto.

1.3 Método de Pesquisa

Para atingir o objetivo geral apresentado na seção anterior, os seguintes passos foram realizados:

1. Revisão bibliográfica: leitura de materiais que forneçam uma revisão dos conceitos já estudados e materiais que apresentem conteúdos novos e necessários para o desenvolvimento do sistema;
2. Análise do problema proposto: leitura dos trabalhos anteriores, com ênfase nos trabalhos realizados por Duarte et al. (2021), onde consta a descrição das ontologias em que este trabalho se baseia, e levantamento de requisitos para a aplicação;

3. Estudo das tecnologias: leitura da documentação oficial da plataforma Jakarta EE, mas também de seus recursos como CDI, Maven e JSF. Além disso, uma pequena revisão sobre algumas tecnologias que seriam úteis ao projeto, a saber, SPARQL, Jena e PrimeFaces;

4. Implementação do sistema: codificação da implementação do sistema utilizando as tecnologias definidas.

1.4 Organização

O restante dessa monografia é organizado da seguinte forma:

O Capítulo 2 apresenta o estado da arte sobre Engenharia de Requisitos de software e a importância da rastreabilidade de requisitos.

O Capítulo 3 apresenta os fundamentos ontológicos adotados neste trabalho. Primeiramente, discutimos a importância e relevância das ontologias para este trabalho. Em segundo lugar, discutimos UFO (GUIZZARDI, 2005; GUIZZARDI, 2007; GUIZZARDI et al., 2013), a ontologia de fundamentação adotada para a construção de ROSS e OSDEF. Finalmente apresentamos ROSS e OSDEF, as duas ontologias de referência de domínio adotadas como base para a realização desse trabalho.

O Capítulo 4 apresenta e discute a ferramenta que está sendo proposta com relação a rastreabilidade de requisitos.

O Capítulo 5 apresenta os trabalhos relacionados, as conclusões desse trabalho, e as direções para trabalhos futuros.

2 Engenharia de Requisitos e Rastreabilidade

Este capítulo apresenta, na Seção 2.1, um resumo sobre Engenharia de Requisitos de Software. Em seguida, a Seção 2.2 foca no domínio rastreabilidade de requisitos de software e discute a importância da utilização de modelos de referência no domínio. Tais temas embasam a solução proposta neste trabalho.

2.1 Engenharia de Requisitos de Software

Sistemas de software são reconhecidamente importantes ativos estratégicos para diversas organizações. Com tal importância, é imprescindível que os sistemas funcionem de acordo com os requisitos estabelecidos. Sendo assim, um passo muito importante no desenvolvimento de software é a identificação e o entendimento dos requisitos dos negócios que os sistemas de software vão apoiar (AURUM; WOHLIN, 2005).

De acordo com Pressman e Maxim (2016) o amplo espectro de tarefas e técnicas que levam a um entendimento dos requisitos é chamado de Engenharia de Requisitos, que se inicia durante a atividade de comunicação e continua na fase de modelagem, devendo ser adaptada às necessidades do processo, do projeto e das pessoas que realizam o trabalho.

Os requisitos podem ser definidos como as descrições das restrições operacionais e dos serviços que devem ser providos pelo sistema (SOMMERVILLE, 2016). Nesse contexto, uma classificação que é amplamente aceita diz respeito ao tipo de informação documentada por um requisito. Requisitos funcionais são declarações de serviços que o sistema deve prover, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Já os requisitos não-funcionais, são restrições aos serviços ou funções oferecidos pelo sistema, incluindo restrições no processo de desenvolvimento e restrições impostas pelas normas (SOMMERVILLE, 2016).

Algo que pode ser útil na representação de requisitos é fazê-lo em níveis diferentes de descrição, pois os *stakeholders*[1] são muito interessados em requisitos, mas possuem expectativas distintas. Dessa forma, Sommerville (2016) propõe a descrição de dois níveis de requisitos:

- **Requisitos de usuário** são declarações, em uma linguagem natural com diagramas, de quais serviços o sistema deverá fornecer a seus usuários e as restrições com as quais este deve operar;

[1] *Stakeholders* são as partes interessadas no desenvolvimento dos sistemas de software. Exemplos comuns de *stakeholders* são os usuários finais do sistema e os diretores da organização que contrata a construção sistema de software.

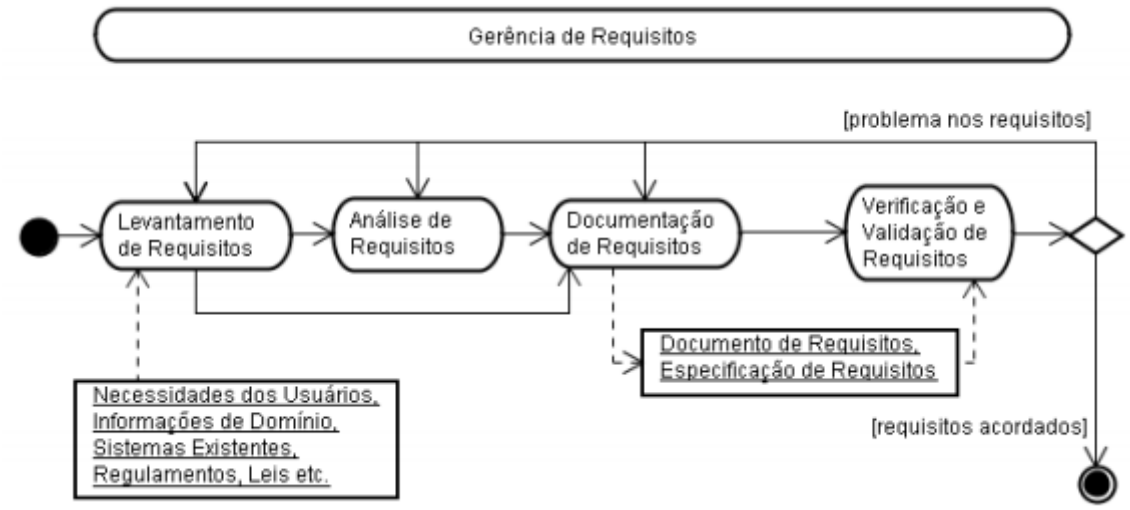

Figura 1 – Processo de Engenharia de Requisitos (KOTONYA; SOMMERVILLE, 1998)

- **Requisitos de sistema** são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software. O documento de requisitos do sistema (às vezes chamado especificação funcional) deve definir exatamente o que deve ser implementado. Pode ser parte do contrato entre o comprador do sistema e os desenvolvedores de software.

Mesmo que vários projetos requeiram processos com características específicas para contemplar suas peculiaridades, é possível estabelecer um conjunto de atividades básicas que deve ser considerado na definição de um processo de Engenharia de Requisitos. Tomando por base o processo proposto por Kotonya e Sommerville (1998), um processo de Engenharia de Requisitos padrão deve contemplar as atividades de levantamento de requisitos, análise de requisitos, documentação de requisitos, verificação e validação de requisitos e gerência de requisitos, como apresentado na Figura 1.

Levantamento de requisitos é a fase inicial do processo de Engenharia de Requisitos e envolve as atividades de descoberta dos requisitos. Nessa fase, um esforço conjunto de clientes, usuários e especialistas de domínio é necessário, com o objetivo de entender a organização, seus processos, necessidades, deficiências dos sistemas de software atuais, possibilidades de melhorias, bem como restrições existentes (KOTONYA; SOMMERVILLE, 1998).

Em seguida, inicia-se a fase de **análise de requisitos**, que serve como base da modelagem do sistema, usando como base os requisitos levantados.

Conforme discutido anteriormente, requisitos de usuário são descritos normalmente em linguagem natural, pois devem ser compreendidos por pessoas que não sejam especialistas técnicos. Entretanto, quanto mais detalhados os requisitos mais úteis eles serão para o desenvolvimento do sistema. Para isso, diversos tipos de modelos podem ser utilizados.

Esses modelos são representações gráficas que descrevem processos de negócio, o problema a ser resolvido e o sistema a ser desenvolvido. Por utilizarem representações gráficas, modelos são geralmente mais compreensíveis do que descrições detalhadas em linguagem natural (SOMMERVILLE, 2016).

Os modelos criados durante a atividade de análise de requisitos cumprem os papeis de: (i) ajudar o analista a entender a informação, a função e o comportamento do sistema, tornando a tarefa de análise de requisitos mais fácil e sistemática; e (ii) tornar-se o ponto focal para a revisão e, portanto, a chave para a determinação da consistência da especificação (PRESSMAN, 2009).

Após a análise dos requisitos levantados, inicia-se a etapa de **documentação de requisitos** que é uma atividade de registro e oficialização dos resultados da Engenharia de Requisitos.

Uma boa documentação apresenta vários benefícios, alguns deles são: tornar fácil a comunicação dos requisitos; reduzir esforço de desenvolvimento, pois sua preparação força *stakeholders* a considerar os requisitos atentamente, para evitar retrabalho posteriormente; prover uma base realista para estimativas, verificação e validação; facilitar a transferência do software para novos usuários e/ou máquinas, além de servir como base para futuras manutenções ou incremento de novas funcionalidades (NUSEIBEH; EASTERBROOK, 2000).

Quanto mais tarde defeitos e problemas são encontrados, maiores os custos associados à sua correção (ROCHA et al., 2001), por isso uma etapa importante e que deve ser iniciada o mais breve possível é a etapa de **verificação e validação de requisitos**. O objetivo da verificação é garantir que o software esteja sendo construído de forma correta, ou seja, deve-se verificar se os artefatos produzidos atendem aos requisitos estabelecidos e se os padrões organizacionais, de produto e processo, foram consistentemente aplicados. Já a etapa de validação tem por objetivo assegurar que o software que está sendo desenvolvido é o software correto, que significa assegurar que os requisitos, e o software derivado deles, atendem ao uso proposto e às expectativas dos *stakeholders* (ROCHA et al., 2001).

Analisando os documentos de requisitos, as etapas de validação e verificação devem garantir que um requisito, seja este uma regra de negócio, requisito funcional ou não funcional, deve ser (WIEGERS; BEATTY, 2013; PFLEEGER, 2004): completo, correto, consistente, realista, passível de ser priorizado, verificável e passível de confirmação e rastreável. Em outras palavras, devem garantir (PRESSMAN, 2009; KOTONYA; SOMMERVILLE, 1998; WIEGERS; BEATTY, 2013): a não ambiguidade de todos os requisitos, a detecção e correção de todas as inconsistências, conflitos, omissões e erros, a documentação em conformidade com os padrões estabelecidos e a real satisfação às necessidades dos clientes e usuários pelos requisitos.

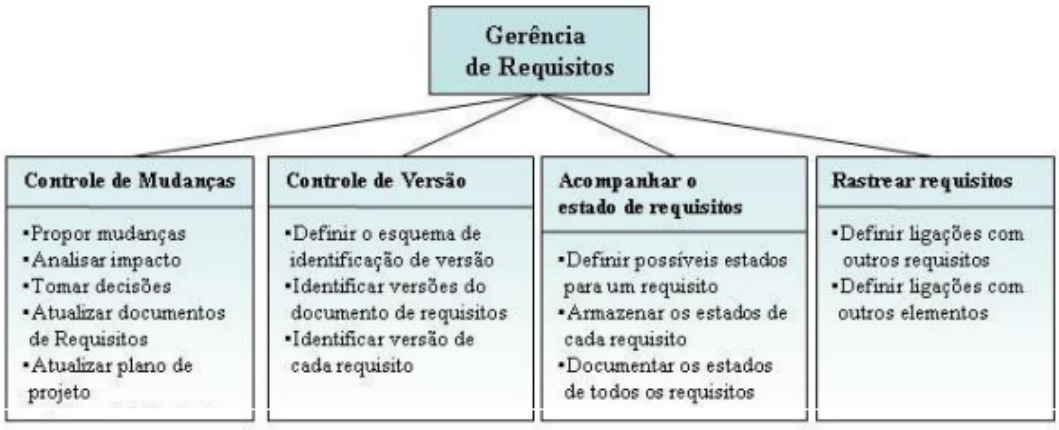

Figura 2 – Atividades de Gerência de Requisitos (WIEGERS; BEATTY, 2013)

Por fim, a **gerência de requisitos**, apresentada visualmente na Figura 2, é a atividade que atua na mudança dos requisitos. Tais mudanças ocorrem ao longo de todo o processo de software, desde o levantamento e análise de requisitos até depois do sistema entrar em operação. Essas mudanças decorrem de diversos fatores, dentre eles descoberta de erros, omissões, conflitos e inconsistências nos requisitos, melhor entendimento por parte dos usuários de suas necessidades, problemas técnicos, de cronograma ou de custo, mudança nas prioridades do cliente, mudanças no negócio, aparecimento de novos competidores, mudanças econômicas, mudanças na equipe, mudanças no ambiente onde o software será instalado e mudanças organizacionais ou legais. Para minimizar as dificuldades impostas por essas mudanças, é necessário gerenciar requisitos (TOGNERI, 2002).

A gerência de requisitos são atividades que ajudam a equipe de desenvolvimento a identificar, controlar e rastrear requisitos ao longo do ciclo de vida do software (KOTONYA; SOMMERVILLE, 1998; PRESSMAN, 2009). Os principais objetivos desse processo de acordo com Kotonya e Sommerville (1998) são a gerência de alteração de requisitos, a gerência de relacionamento de requisitos e a gerência de dependências entre requisitos e outros documentos produzidos no processo de software. Para isso o processo de gerência de requisitos deve incluir as atividades de controle de mudanças, controle de versão, controle de estado dos requisitos e rastreamento de requisitos (WIEGERS; BEATTY, 2013).

O controle de mudança define os procedimentos, processos e padrões que devem ser utilizados para gerenciar as alterações de requisitos, garantindo que toda proposta de mudança seja analisada conforme os critérios estabelecidos pela organização (KOTONYA; SOMMERVILLE, 1998). O controle de mudanças, em geral, envolve atividades como (KOTONYA; SOMMERVILLE, 1998; WIEGERS; BEATTY, 2013): verificar a validade da mudança, rastrear informações sobre que requisitos e artefatos são afetados pela mudança, estimar o custo e impacto da mudança, negociar as mudanças com os cliente e alterar os requisitos e documentos associados aos mesmos.

Ao se detectar a necessidade de mudança em um ou mais requisitos, uma solicitação deve ser registrada e passar por uma avaliação da equipe de projeto. Avaliação esta, que deverá verificar o impacto, que é uma parte crítica do controle de mudanças, e os valores de custo, esforço, tempo e viabilidade deverão ser passados para o solicitante da mudança. Além disso, para realizar essa mudança é necessário estabelecer uma rede de ligações de modo que um requisito e os elementos ligados a ele possam ser rastreados, identificado unicamente cada requisito e garantindo a possibilidade de saber quais requisitos dependem de que outros requisitos (FALBO, 2012). Surge então o conceito de **rastreabilidade**, que será abordado na próxima seção.

2.2 Rastreabilidade de Requisitos

A rastreabilidade pode ser definida como um grau relacional estabelecido entre duas ou mais entidades lógicas, especificamente as que possuem um predecessor-sucessor ou mestre-subordinado entre elas. Segundo Gotel e Finkelstein (1994), rastreabilidade de requisitos tem como definição a capacidade de seguir o ciclo de vida de um requisito, seja este requisitos de mesmo nível seja entre diferentes diferentes níveis no processo de Engenharia de Software.

Neste trabalho usamos como exemplo os requisitos de um software de caixa eletrônico elicitados por Bjork (2009), apresentados no Apêndice A. Como um exemplo de rastreabilidade, podemos identificar que o requisito de *stakeholder* STREQ001 — "O caixa eletrônico se comunicará com o computador do banco por meio de um link de comunicação apropriado" — impacta diretamente o requisitos de sistema SYSREQ001 — "O caixa eletrônico deve ter uma conexão de Internet adequada para se comunicar com o sistema do banco" —, ao mesmo tempo que o SYSREQ001 é também impactado por STREQ001, como pode ser visto na Tabela 6 no Apêndice A. Tal exemplo ilustra a rastreabilidade bidirecional dos requisitos.

Há quase 25 anos a rastreabilidade de requisitos vem sendo reconhecida nas principais literaturas científicas como uma atividade capaz de melhorar a qualidade de gestão de sistemas de softwares (GOTEL; FINKELSTEIN, 1994; RAMESH et al., 1995; RAMESH et al., 1997). Em outras palavras, a capacidade de rastrear de requisitos permite as partes interessadas acompanhar evolução do sistema e medir o impacto das mudanças que acontecem ao longo do ciclo de vida do software. Além disso, modelos de maturidade (SEI/CMU, 2010; BOURQUE; FAIRLEY et al., 2014) e padrões internacionais (ISO, IEC, 2017a; ISO, IEC, 2018) sugerem que a aptidão de rastrear requisitos mutáveis e relacioná-los a outros artefatos no ciclo de vida do software é indispensável para produtos de software de alta qualidade.

Requisitos são um artefato de alta prioridade e a fundação de todos os projetos

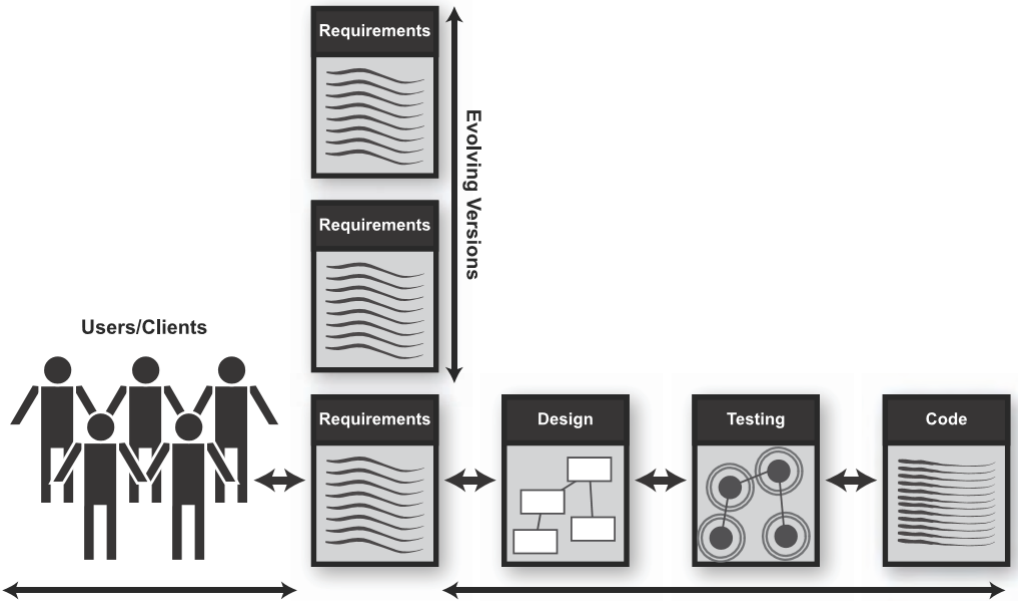

Figura 3 – Visão Geral de Rastreabilidade de Requisitos (KANNENBERG; SAIEDIAN, 2009)

de software; porém, muitos autores apontam a importância e relevância de outros muitos artefatos de software para o ciclo de vida do software. Esses artefatos estão diretamente relacionados aos requisitos de software por relações de dependência genérica ou mesmo existencial.[2] Tais relações, embora muitas vezes negligenciadas durante o ciclo de vida software, apresentam uma importante fonte de conhecimento sobre o domínio do sistema de software e facilitam sua gestão. Nesse contexto, uma das classificações mais aceitas na literatura propõe a separação entre rastreabilidade de requisitos horizontais e verticais (LINDVALL; SANDAHL, 1996).[3]

A rastreabilidade de requisitos horizontal está ligada à atividade de rastrear as relações existentes entre os elementos em diferentes modelos.[4] Já a rastreabilidade vertical se refere ao rastreio das relações entre os elementos no mesmo modelo.[5]

A Figura 3 apresenta uma visão geral do domínio de rastreabilidade de requisitos e descreve os conceitos de rastreabilidade horizontal e vertical que foram discutidos acima.

No entanto, construir e manter um mecanismo eficiente que forneça rastreabilidade entre requisitos é uma tarefa de alta complexidade. Diversos pesquisadores (GOTEL; FINKELSTEIN, 1994; KANNENBERG; SAIEDIAN, 2009; REGAN et al., 2012; TUFAIL

[2] Por exemplo, um caso de teste pode ser *existencialmente dependente* de um requisito funcional.

[3] Um exemplo comum de rastreabilidade de requisitos com outros artefatos dentro do ciclo de vida do software acontece na relação entre os casos de teste e os requisitos que são submetidos aos testes.

[4] Por exemplo, um programa PROG002 pode ser rastreado até um requisito de programa PROGREQ002 em um projeto de software

[5] Por exemplo, o PROGREQ002 usado no exemplo anterior também pode estar diretamente relacionado a outros requisitos de programa, como realmente o PROGREQ002 é relacionado ao PROGREQ014, como pode ser visto na tabela 8 no apêndice A

et al., 2017) apresentaram ao longo dos anos, estudos que citam os maiores e mais comuns problemas para estabelecer uma política de rastreabilidade de requisitos eficaz. Problemas como decaimento da rastreabilidade, complexidade dos sistemas, pouco suporte de ferramentas, comunicação ineficaz entre as partes interessadas, integração deficiente de dados e até mesmo a limitação na geração de dados necessários, são listados como principais obstáculos a serem superados no campo de estudo e aplicabilidade.

Dependendo do nível de maturidade da organização, os links de rastreabilidade são guardados de forma física em planilhas ou documentos baseados em texto, onde os links tendem a se deteriorar durante um projeto à medida que as atualizações, que são responsabilidades dos membros da equipe equilibrando-se em diversas outras tarefas operacionais, por vezes comprometem a atualização dos dados necessários para uma resposta eficaz na rastreabilidade dos dados.

Além disso, embora a rastreabilidade de requisitos seja constantemente referenciada em diversos padrões internacionais e modelos de maturidade, os requisitos específicos do processo ou diretrizes sobre como ela deve ser implementada raramente são fornecidos (KIROVA et al., 2008). Esse fenômeno acontece pois é sugerido que, como cada organização possui peculiaridades em seus processos, determinados fluxos de trabalho, produtos, recursos e serviços, elas necessitam da criação ou adaptação de um método viável e acessível para as demandas da organização.

Uma potencial solução para esses problemas é a utilização de ferramentas de gerenciamento de requisitos, como por exemplo, o IBM Rational RequisitePro, no entanto, tanto essa como outras ferramentas, em questão, tem frequentemente um custo muito elevado e muitas vezes acabam não sendo uma opção acessível para organizações de pequeno e médio porte. Por isso, infelizmente, muitas das organizações não priorizam a implementação das práticas de rastreabilidade, seja pelas dificuldades mencionadas ou porque sucumbem ao equívoco de que as práticas de rastreabilidade possuem apenas resultados de médio ou longo prazo para as organizações(CLELAND-HUANG et al., 2007).

Alonso-Rorís et al. (2016) propõem que para um funcionamento adequado de rastreabilidade é necessário que seja estabelecido um esquema formal de descrição das informações detalhadas que facilite o raciocínio e que a aplicação de tecnologias semânticas promovam a utilização de metodologias e técnicas de rastreabilidade semântica, acarretando no gerenciamento de complexas relações entre itens de dados. Segundo também mencionado por Espinoza e Garbajosa (2011), o uso da semântica como questão fundamental no que diz respeito à rastreabilidade é imprescindível. Ramesh e Jarke (2001) sugerem que, para ter utilidade, a rastreabilidade necessariamente deve ser organizada em uma estrutura de modelagem adequada.

Devido a essas características e obstáculos presentes no domínio, diversos pesquisadores têm empenhado esforços com relação ao desenvolvimento de modelos que

servem de referência para rastreabilidade de requisitos. Possibilitando enxergar uma janela para a realidade do que futuramente pode ser adotado em organizações com base em suas demandas individuais. Como sugere o próprio significado, os chamados modelos de referência são *templates* criados sobre uma demanda específica, com a finalidade de reduzir significativamente a tarefa de criar outros novos modelos e/ou sistemas restritivos para aplicações. O funcionamento consiste na parte onde o usuário final do modelo de referência seleciona o que é ou não relevante do modelo e as adapta ao problema que necessita da solução configurando assim uma resposta a partir das partes adaptadas. Em outras palavras, modelos de referência podem ser compreendidos como uma abstração representativa de um conhecimento bem consolidado dentro de um domínio (RAMESH; JARKE, 2001).[6]

O trabalho de Ramesh e Jarke (2001) é considerado um dos mais importantes e precursores do objeto de pesquisa referente à rastreabilidade que se baseia em modelos de referência. Eles propõem que a eficiência e a eficácia da rastreabilidade dependem fortemente do sistema de objetos e tipos de links de rastreabilidade que são utilizados. Além disso, os autores também indicam que os modelos referenciados podem ser desenvolvidos baseados nos diferentes aspectos do campo de rastreabilidade de requisitos. Por exemplo, o modelo de referência deles para rastreabilidade de requisitos está focado nos Objetos (Artefatos) que existem no processo de software, enquanto o Modelo Semântico proposto por Alonso-Rorís et al. (2016) está focado na caracterização do domínio do processo de rastreabilidade de tipos de rastros. Esse e muitos outros estudos podem ser facilmente encontrados na literatura com diferentes abordagens para o desenvolvimento de um mecanismo de rastreabilidade que, com base em modelos, tem o objetivo primário de fornecer resultados palpáveis e eficientes, além de suporte geral do processo para os usuários finais do recurso.

A abordagem proposta por Zhang et al. (2014) da ênfase nas dependências existentes entre os requisitos. Eles utilizam repetidamente o modelo de dependência de requisitos de Pohl e o modelo de dependência de requisitos (inter) de (DAHLSTEDT; PERSSON, 2005) para dessa forma elaborar um novo e utilizar como modelo de referência para uma rastreabilidade de requisitos em um estudo de caso de base industrial.

Por fim, a utilização de modelos de referência é frequentemente apontada dentro da literatura de rastreabilidade de requisitos como meio de representar os aspectos que mais importam no processo de rastreabilidade para uma organização. Conforme citado anteriormente nessa seção, a rastreabilidade de requisitos pode fornecer inúmeros benefícios para o processo de software de uma organização, porém, o processo tende a se tornar mais complexo e mais suscetível a erros conforme o software naturalmente avança por seu ciclo de

[6] De forma geral, domínios com elevado nível de padronização, como o domínio dos sistemas de software contemporâneos, podem ser beneficiados pela utilização de modelos de referência.

vida e sofre alterações. A utilização de modelos de referência é uma tentativa de organizar algumas das opções que podem ser realizadas, no sentido de que a organização necessita seguir diretrizes propostas no modelo, ao mesmo passo traz facilidade no entendimento dos *stakeholders* do processo de software.

3 Ontologias para Rastreabilidade de Software

Esse capítulo resume o trabalho de Duarte et al. (2021), que serviu como base para o desenvolvimento desta monografia. Começando pela fundação, a Seção 3.1 apresenta a definição de ontologia e as principais classificações adotadas na literatura, enquanto a Seção 3.2 apresenta UFO, a ontologia de fundamentação adotada para a construção de ROSS e OSDEF. Na sequência, as seções 3.3 e 3.4 apresentam as ontologias construídas por Duarte et al., respectivamente ROSS, a ontologia de referência sobre sistemas de software e OSDEF, a ontologia de defeitos, erros e falhas de software. ROSS e OSDEF foram as ontologias adotadas como base para a realização desse trabalho.

3.1 Ontologias

O termo "ontologia", segundo o dicionário Michaelis, é formado do grego *ontos* (ser) e *logia* (estudo), ou seja, a teoria ou ramo da filosofia cujo objeto é o estudo dos seres em geral, o estudo das propriedades mais gerais e comum a todos os seres; metafísica ontológica ou o estudo ou conhecimento dos seres e dos objetos enquanto eles mesmos, em oposição ao estudo de suas aparências e atributos.

Veio da filosofia como ramo geral da metafísica desde a época de Aristóteles (mas esta com "O" maiúsculo), porém o termo "ontologia" foi criado apenas no século 17, pelos filósofos Rudolf Gockel e Jacob Lorhad. No início do século 20, Edmund Husserl definiu a Ontologia Formal como um análogo da Lógica Formal, uma disciplina que visa desenvolver um sistema de conjunto de categorias geral e independente de domínio que pode ser usado no desenvolvimento de teorias científicas de domínio específico (SMITH, 1989).

Atualmente, uma das definições de ontologia mais aceitas, e usada como referência em (DUARTE et al., 2021), trabalho no qual este projeto é baseado, é a de Borst e Borst (1997) como uma especificação formal e explícita de uma conceituação compartilhada.

Embora o conceito de ontologia tenha uma natureza filosófica clara, tem sido reconhecido como um campo de pesquisa promissor nas áreas da Ciência da Computação, como Inteligência Artificial, Engenharia de Software, Dados Interligados, Design de Banco de Dados, Engenharia do Conhecimento e integração de informações (GUARINO, 1998), onde as ontologias têm sido reconhecidas como um instrumento conceitual bastante útil desde a década de 60. Na Engenharia de Software elas têm sido utilizadas para reduzir ambiguidades conceituais, tornar transparentes as estruturas de conhecimento, apoiar o compartilhamento de conhecimento e a interoperabilidade entre sistemas (JASPER;

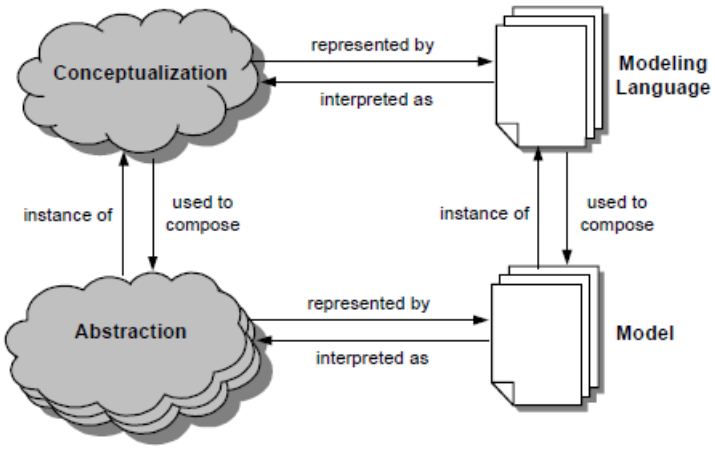

Figura 4 – Relações entre conceituação, linguagem de modelagem, abstração e modelo (GUIZZARDI, 2007)

USCHOLD et al., 1999), se destacando por sua usabilidade no ramo da Engenharia de Domínio, onde uma ontologia de domínio pode ser utilizada como um modelo de domínio, uma vez que ambos buscam fornecer um entendimento uniforme e não ambíguo de objetos e suas relações, provendo uma conceituação acerca de um determinado domínio (FALBO; GUIZZARDI; DUARTE, 2002).

A Figura 4 descreve as relações entre Conceituação, Linguagem de Modelagem e suas instâncias, Abstração e Modelo, respectivamente. Um dos principais fatores de sucesso por trás do uso de uma linguagem de modelagem está na capacidade da linguagem de fornecer aos usuários-alvo um conjunto de primitivas de modelagem que podem expressar diretamente conceitos de domínio relevantes, incluindo o que chamamos aqui de conceituação de domínio. Os elementos que constituem uma conceituação de um determinado domínio são usados para articular abstrações de um determinado estado de coisas na realidade. Chamamos as últimas abstrações de domínio. Conceituação e abstrações são representadas com nuvens porque são entidades imateriais que existem na mente de um usuário ou grupo de usuários de uma linguagem. Isso significa que para que uma comunicação seja precisa e inequívoca, é necessária uma linguagem que represente a mesma conceituação para os participantes da conversa (GUIZZARDI, 2007). Em (GUIZZARDI, 2005) se discute profundamente um conjunto de propriedades que devem ser garantidas para um mapeamento isomórfico entre uma Abstração e um Modelo.

3.1.1 Tipos de Ontologias

Existem múltiplas classificações ontológicas. Uma das mais citadas na literatura, é a proposta por Guarino (1998) e mostrada na Figura 5, que classifica ontologias nivelando-as por sua generalidade. A figura representa os tipos de ontologias e os relacionamentos entre elas.

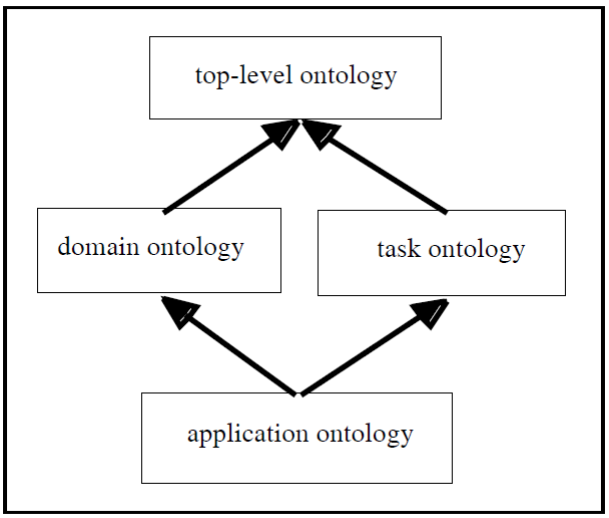

Figura 5 – Tipos de Ontologias de acordo com o seu nível de generalidade, apresentados em (GUARINO, 1998). As setas representam relacionamentos de especialização.

Uma Ontologia de Fundamentação, ou de nível superior são as ontologias que descrevem conceitos gerais independentes de um problema ou domínio particular, definidos a partir de estudos da Filosofia, Linguística e Psicologia Cognitiva, como por exemplo, o tempo, o espaço, a matéria, objetos, eventos e os tipos de relações fundamentais entre esses conceitos. Também de acordo com Guizzardi, Falbo e Guizzardi (2008) é um sistema de categorias filosoficamente bem fundamentado e independente de domínio. Uma ontologia de fundamentação deve considerar os princípios e conceitos gerais que envolvem diversos domínios da realidade, tratados pela disciplina Ontologia Formal, em Filosofia. Mais especificamente, devem ser consideradas as estruturas ontológicas formais, como a teoria todo-parte, tipos e instanciações, identidade, dependência e outros. Uma ontologia de fundamentação envolve essas estruturas ontológicas e é considerada um produto da Ontologia Formal.

Além de fornecerem a base para a construção de ontologias de domínio e de tarefa, ontologias de fundamentação também podem ser utilizadas para avaliar linguagens de modelagem conceitual e para desenvolver diretrizes para o uso dessas linguagens (WAND; STOREY; WEBER, 1999; EVERMANN; WAND, 2001; OPDAHL; HENDERSON-SELLERS, 2001; GUIZZARDI; HERRE; WAGNER, 2002), bem como para melhorar a qualidade de modelos conceituais, incluindo ontologias de domínio (NARDI; FALBO, 2008; GUIZZARDI; FALBO; GUIZZARDI, 2008; SILVA; FILHO; GONÇALVES, 2008). Por exemplo, UFO (Unified Foundational Ontology) (GUIZZARDI, 2005) tem sido utilizada para avaliar, reprojetar e dar semântica de mundo real a ontologias de domínio.

Ontologias de Domínio e de Tarefa descrevem, respectivamente, o vocabulário relacionado a domínios que descrevem conceitos relacionados a um domínio específico, como Engenharia de Requisitos ou Economia, ou a uma tarefa ou atividade específicos,

que representam ontologias que são criadas sobre tarefas ou atividades genéricas, que atravessam vários domínios ambos geralmente criadas pela especialização de conceitos de uma ontologia de nível superior pré-existente.

A construção de ontologias de domínio é uma das aplicações mais comuns das ontologias em Engenharia de Software, havendo inúmeros trabalhos desenvolvidos nesse sentido. Uma ontologia que representa uma conceituação para um domínio deve ser independente da aplicação e, para isso, conforme orienta Guarino (1998), seus conceitos devem ser mapeados em conceitos de uma ontologia de fundamentação.

As ontologias de Aplicação descrevem conceitos sobre um domínio específico e as tarefas e atividades particulares dele. Assim, as Ontologias de Aplicação geralmente se especializam em ontologias de domínio e de tarefa.

Outra classificação amplamente aceita é a proposta por Guizzardi (2007), que é ortogonal à apresentada por Guarino e que distingue entre ontologias como modelos conceituais, conhecidas como ontologias de referência, e ontologias como artefatos de codificação, chamadas de ontologias operacionais.

Ontologias de referência são modelos conceituais fortemente axiomatizados, criados para representar o domínio pretendido da melhor forma possível. Geralmente elaboradas com linguagens gráficas e com foco em expressividade e adequação ao domínio. Além disso, ontologias de referência são ferramentas utilizadas para serem entendidas por humanos, auxiliando-os em suas tarefas.

Por outro lado, as ontologias operacionais (ou ontologias leves) são baseadas em ontologias de referência, no entanto, são utilizadas em processamento de máquina, com linguagens que enfocam propriedades computacionais, como OWL ou RDF (GUIZZARDI, 2007). Ao contrário das ontologias de referência, ontologias operacionais são projetadas com o foco na garantia de propriedades computacionais desejáveis (FALBO et al., 2013).

Para este trabalho, adotamos ambas as classificações que foram discutidas. Usaremos a proposta de ontologia de referência de domínio sobre requisitos de sistemas baseados em software de Duarte et al. (2021). Esta ontologia específica, por sua vez, é baseada na ontologia de fundamentação UFO e também possui uma versão operacional implementada, baseada em gUFO. Nas seções que seguem introduziremos os conceitos de UFO e gUFO mencionados nesta seção.

3.2 UFO

A Ontologia de Fundamentação Unificada – UFO (GUIZZARDI, 2005; GUIZZARDI, 2006; GUIZZARDI, 2007; GUIZZARDI et al., 2013) é, como o próprio nome sugere, uma ontologia de fundamentação que foi desenvolvida com base em uma série de teorias

de Lógica Modal, Linguística, Filosofia, Ontologias Formais e Psicologia Cognitiva. É composta por três partes:

- UFO-A que é uma ontologia cujo objetivo é discutir objetos (*endurants*), seus tipos, composições e relações (GUIZZARDI, 2005); ela é o principal pilar de sustentação de UFO;
- UFO-B é uma ontologia de eventos (*perdurants*), suas composições, seus participantes e relações (GUIZZARDI et al., 2013; BENEVIDES et al., 2019). A principal diferença entre eventos (*perdurants*) e objetos (*endurants*) é que um objeto existe ou não existe no tempo, enquanto eventos são entidades temporais, complexas ou atômicas;
- Por fim UFO-C (GUIZZARDI, 2006; GUIZZARDI; FALBO; GUIZZARDI, 2008; BRINGUENTE; FALBO; GUIZZARDI, 2011b) é construída sobre os conceitos de UFO-A e UFO-B e é mais específica que ambas. UFO-C é uma ontologia de entidades sociais (*endurants* e *perdurants*) e discute os agentes, suas ações, objetivos, intenções e compromissos.

Por razões de brevidade, a Figura 6 mostra os conceitos de UFO que são reutilizados nesta tese.[1] Os conceitos apresentados são originalmente definidos nas três partes principais do UFO: UFO-A, UFO-B e UFO-C.

UFO foi escolhido como a ontologia fundamental para fundamentar as ontologias ROSS e OSDEF (DUARTE et al., 2021) por abordar muitos aspectos essenciais para a modelagem conceitual, fornecendo um conjunto completo de categorias para lidar com o domínio do software. Além de ser empregado, com sucesso, como ontologia fundamental para a criação de várias outras ontologias relacionadas a software (BRINGUENTE; FALBO; GUIZZARDI, 2011a; BARCELLOS; FALBO; MORO, 2010; DUARTE et al., 2018; SOUZA et al., 2013) e por ser a ontologia fundamental sugerida pelo SABiO, o método de engenharia de ontologias que foi usado por Duarte et al. (2021) para criar as ontologias utilizadas neste projeto.

3.2.1 UFO-A

Entidade (*Entity*) é o conceito do qual todos os demais conceitos de UFO são especializados, ou seja, é o conceito mais genérico em UFO. Uma entidade é definida como algo concebível ou perceptível. Em UFO-A, a primeira distinção que é realizada entre as especializações de entidade é a distinção fundamental entre as categorias de indivíduos e universais. Indivíduos (*Particulars*) são entidades que existem na realidade, possuindo uma identidade única. Uma casa, uma flor e uma pessoa são exemplos de indivíduos. Universais

[1] O modelo completo de UFO, combinando UFO-A UFO-B e UFO-C tem cerca de 100 conceitos, por isso torna-se impraticável representar todo o modelo neste formato.

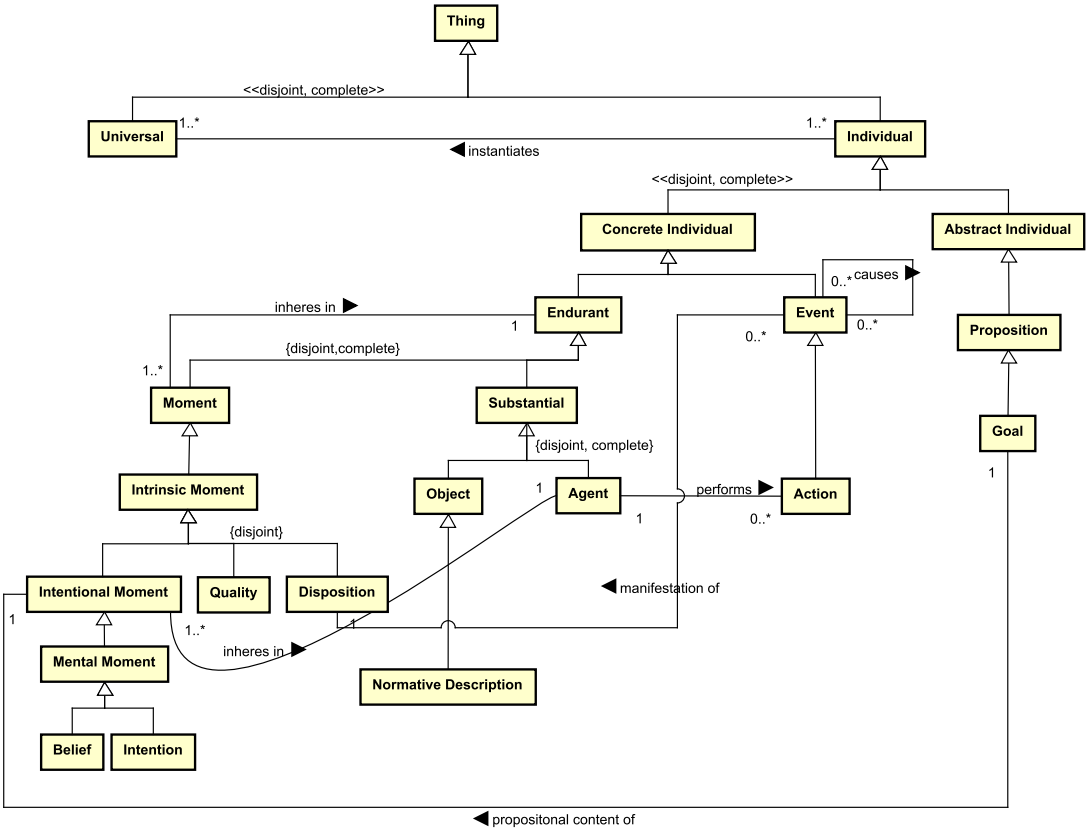

Figura 6 – Um fragmento de UFO apresentando os conceitos usados neste trabalho.

(*Universals*), por sua vez, são padrões de características que podem ser instanciados em um número de diferentes indivíduos. Pessoa, por exemplo, é um universal que descreve as características comuns aos indivíduos desse tipo como, por exemplo, nome e impressão digital.

Indivíduos podem ser existencialmente independentes ou dependentes. Substanciais (*Substantials*) são indivíduos existencialmente independentes, como por exemplo, uma pessoa, uma casa e outros objetos mesoscópicos do senso comum. Ao contrário, Momentos (*Moments*) são indivíduos que só podem existir em outros indivíduos (são existencialmente dependentes) e são ditos inerentes a esses indivíduos. Um momento denota a instanciação de uma propriedade. Uma cor, por exemplo, é um momento, ou seja, é uma propriedade que só pode existir em outro indivíduo.

A dependência existencial também pode ser utilizada para diferenciar dois tipos de momentos. Momentos Intrínsecos (*Intrinsic Moments*) são dependentes de um único indivíduo (por exemplo, uma cor depende de um único indivíduo, como a cor de uma maçã). Por outro lado, Momentos Relacionais (*Relators*) dependem de vários indivíduos (por exemplo, um emprego, que envolve um empregador e um empregado).

Relações (*Relations*) são entidades que aglutinam outras entidades. Considerando-se a base filosófica, duas categorias amplas de relações são consideradas em UFO-A:

relações formais e relações materiais. Relações formais (*Formal Relations*) acontecem entre duas ou mais entidades diretamente, sem que haja necessidade de que outro indivíduo intervenha. Em princípio, relações formais incluem as relações que formam a superestrutura matemática, como dependência existencial, parte-de, subconjunto-de, instanciação, dentre outras. Relações Materiais (*Material Relations*), por outro lado, possuem estrutura material própria e incluem exemplos como trabalhar em, estar matriculado em ou estar conectado a. Enquanto, por exemplo, a relação formal entre João e seu conhecimento x de Grego acontece diretamente, desde que João e x existam, para que aconteça uma relação material "ser tratado em" entre João e uma unidade médica UM, é necessário que exista uma outra entidade (nesse caso, um tratamento) para mediar João e UM.

UFO-A considera, ainda, a noção de situação. Situações (*Situations*) são entidades complexas constituídas possivelmente por vários objetos (incluindo outras situações), sendo tratadas como um sinônimo para o que a literatura chama de estado de coisas (*state of affairs*), ou seja, uma porção da realidade que pode ser compreendida como um todo. Um exemplo de situação é João está gripado e com febre.

Em UFO-A também há os conceitos de Universal de Primeira Ordem (*First Order Universal*) e Universal de Mais Alta Ordem (*High Order Universal*). Os universais de primeira ordem englobam universais cujas instâncias são indivíduos como, por exemplo, Pessoa. Já os universais de mais alta ordem representam os universais cujas instâncias são universais de primeira ordem. Exemplos de universais de mais alta ordem podem ser Espécies de Pássaros, com as instâncias Pinguim e Papagaio, que são universais de primeira ordem.

Por fim, segundo UFO-A, alguns substanciais universais são Sortais (*Sortal Universal*), ou seja, como substanciais universais, são padrões de características que podem ser realizadas em um número distinto de substanciais e, além disso, são sortais, isto é, proveem princípio de individualização, persistência e identidade. As classes dos animais (Mamíferos, Répteis, Aves etc.) são exemplos de sortais universais. Outros substanciais universais, chamados Mixins (*Mixin Universals*), não fornecem princípio de identidade, sendo meramente caracterizações. Entidades Racionais, ou seja, entidades com a propriedade de ser capaz de raciocinar, são exemplos de mixins universais.

Sortais rígidos podem ser Tipos (*Kinds*) ou Subtipos (*Subkinds*). Mamífero é um exemplo de tipo. Suas especializações, Mamífero Terrestre e Mamífero Aquático, são exemplos de subtipos. Sortais antirrígidos podem ser Fases (*Phases*) ou Papéis (Roles). O primeiro representa um possível estágio na história de um sortal universal, constituindo uma partição deste. As fases do ciclo de vida das plantas, das pessoas. O segundo representa um papel que um sortal universal pode desempenhar ao longo de sua história e que é demarcado por suas relações com outras entidades (é relacionalmente dependente). Mixins universais podem ser rígidos ou antirrígidos. Mixins Rígidos (*Rigid Mixins*) são mixins

cujos padrões de características são aplicados a todas as suas instâncias. Mixins rígidos que agregam propriedades essenciais e comuns a diferentes sortais universais são chamados de Categorias (*Categories*).

3.2.2 UFO-B

Eventos são indivíduos compostos por partes temporais e, ao contrário dos objetos, acontecem no tempo, no sentido de se estenderem no tempo acumulando partes temporais. Uma conversa, uma partida de futebol, a execução de uma sinfonia e um processo de negócio são exemplos de eventos. Eventos são, ainda, possíveis transformações de uma situação da realidade para outra, ou seja, eventos podem alterar o estado de coisas (conceito de situação em UFO-A) de um estado (pré-estado ou *pre-state*) para outro (pós-estado ou *post-state*). Eventos são entidades ontologicamente dependentes, ou seja, para existirem dependem de seus participantes. Por exemplo, seja o evento e: o ataque de Brutus a César. Nesse evento há participação de César, Brutus e da faca utilizada no ataque. Então, é composto pela participação individual de cada uma dessas entidades. Cada participação (*Participation*) é existencialmente dependente de um único substancial e pode ser por si só um Evento Atômico (*Atomic Eevent*) ou um Evento Complexo (*Complex Event*).

3.2.3 UFO-C

Segundo a UFO-C, uma Descrição Normativa (*Normative Description*) é um tipo de objeto social que define uma ou mais regras/normas reconhecidas por pelo menos um agente social e que pode definir entidades sociais como universais (por exemplo, tipos de comprometimentos sociais), outros objetos (por exemplo, a coroa do rei da Espanha) e papéis sociais (por exemplo, pedestre e presidente). São exemplos de descrições normativas a Constituição Brasileira e o regimento dos cursos de Pós-Graduação da COPPE/UFRJ.

Por essa razão, intenções levam agentes a executarem Ações (*Actions*). Ações são eventos intencionais, ou seja, eventos que instanciam um plano (Ação Universal ou *Action Universal*) com o propósito específico de satisfazer o conteúdo proposicional de alguma intenção.

Nem toda participação de um agente é considerada uma ação, mas somente as participações intencionais. Objetos, sendo entidades inanimadas, não realizam ações, mas podem participar delas (*Resource Participation*) como recurso (*resource*).

3.2.4 gUFO

A 'gentle' UFO (gUFO) (ALMEIDA et al., 2019) é uma versão operacional leve do UFO criada para fornecer uma implementação do UFO adequada para o desenvolvimento

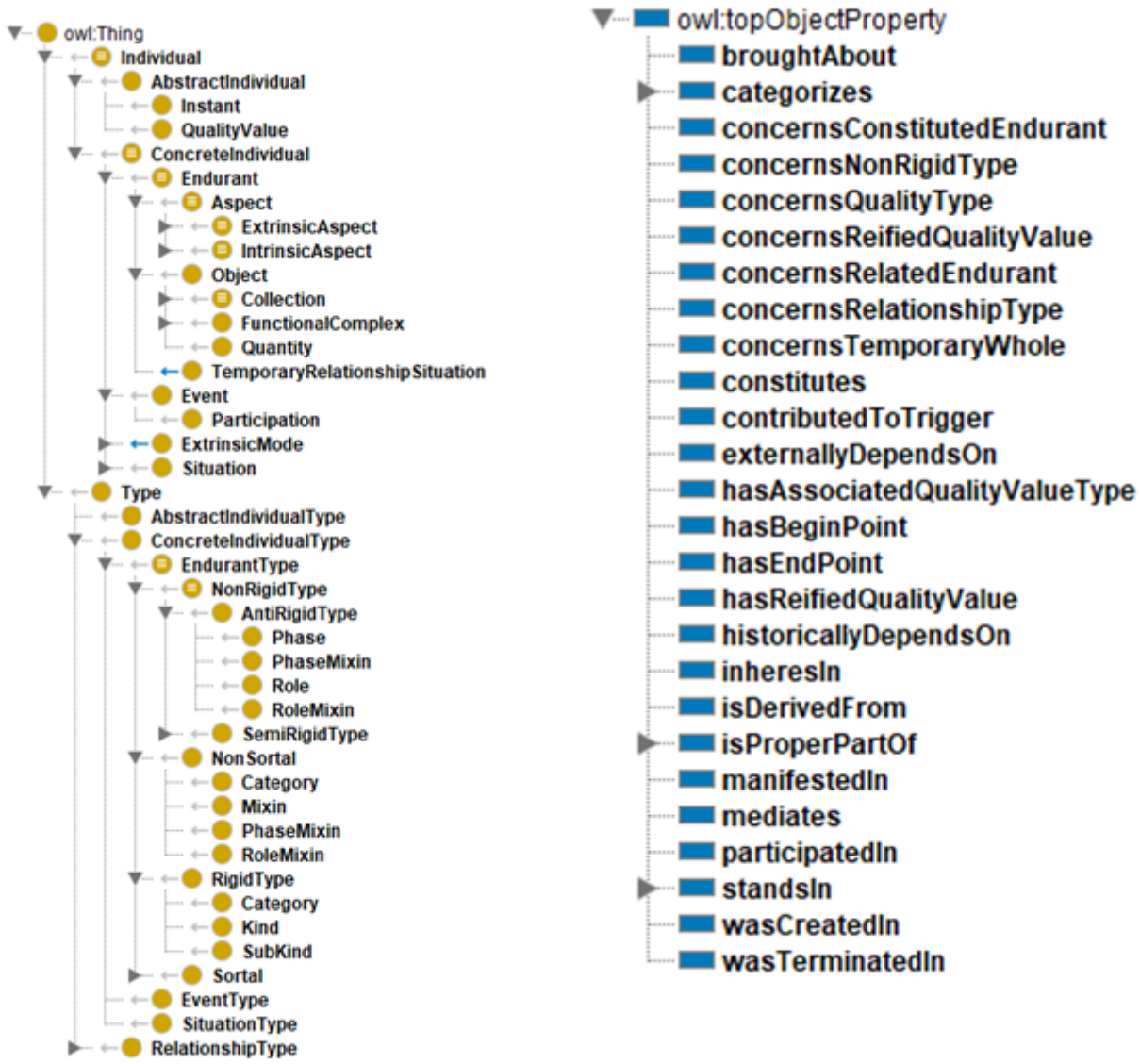

Figura 7 – Hierarquia de conceitos e visualização de relações do gUFO no Protégé.

de abordagens de dados interligados (*linked data*) e deve ser utilizada por projetistas de ontologias leves / operacionais como uma versão operacional do UFO.

Como uma implementação de UFO, a principal característica de gUFO é que inclui ambas as taxonomias originalmente apresentadas em UFO: a primeira com classes cujas instâncias são individuais (por exemplo, os conceitos de gufo:Object e gufo:Event) e a segunda com classes cujas instâncias são tipos / universais (por exemplo, os conceitos de gufo:Kind, gufo:Phase, gufo:Category). gUFO deve ser reutilizada por meio da especialização e instanciação de seus conceitos, facilitando a implementação de ontologias de referência originalmente desenvolvidas a partir de UFO.

A Figura 7 apresenta à esquerda a hierarquia dos conceitos definidos em gUFO e à direita as relações que são definidas entre esses conceitos, as quais são implementadas

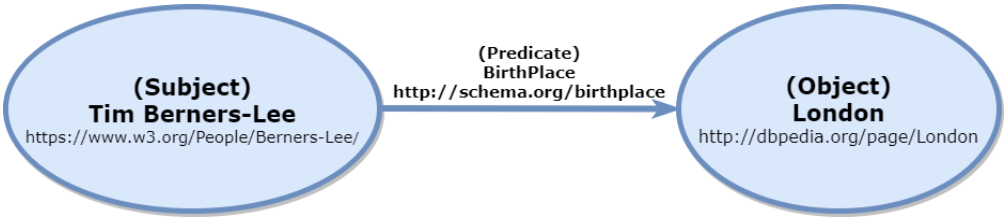

Figura 8 – Representação gráfica de uma tripla que representa as informações do local de nascimento de Sir Tim Berners-Lee. Adaptado de (W3C, 2004)

como propriedades dos conceitos.[2]

gUFO apresenta a implementação de uma parte do UFO-A (GUIZZARDI, 2005; GUIZZARDI; FALBO; GUIZZARDI, 2008), que define os conceitos de Objetos, Aspectos, Situações e a distinção entre Indivíduos e Tipos (*Universals*). gUFO também implementa a definição de Eventos e a noção de Participação do objeto em eventos, que é definida em UFO-B (GUIZZARDI et al., 2013).

Por último, embora gUFO não apresente a definição de conceitos de UFO-C, como Agentes e Ações, ela foi desenhada para que o usuário possa definir novos conceitos com menos esforço. Por exemplo, uma Ação pode ser definida como um subtipo de Evento, assim como é originalmente definido em UFO-C, e irá incorporar automaticamente todas as propriedades que já foram implementadas para o conceito de Evento. gUFO foi usada como base para o desenvolvimento das ontologias operacionais que serviram para avaliar os modelos de referência do ROSS e do OSDEF (DUARTE et al., 2021).

3.2.5 RDF e SPARQL

O Resource Description Framework (RDF) é uma linguagem de representação formal de informações na Web. O RDF é uma recomendação do W3C, desenvolvida para representar os dados de forma flexível, além de permitir o processamento automatizado desses dados.

Diferentemente da WWW, que foi criada para ser legível por humanos, a RDF foi criada para ser uma linguagem para a Web Semântica que fosse legível para máquinas (W3C, 2004). No entanto, essa não é a única diferença entre RDF e outras tecnologias da Web conhecidas. Enquanto HTML e XML foram criadas em uma estrutura de árvore os documentos RDF organizaram-se em modelos de dados de grafos (HITZLER; KROTZSCH; RUDOLPH, 2009). Este modelo de dados organizados em grafos, define a estrutura de um conjunto de triplas, que são compostos por um sujeito, um predicado ou propriedade, que denota uma relação, e um objeto.

A Figura 8 apresenta essa estrutura em um exemplo de grafo sobre o local de

[2] A descrição completa de todos os elementos gUFO e seu código-fonte podem ser encontrados em <https://nemo-ufes.github.io/gufo/>.

nascimento de Tim Berners-Lee, onde os nós do grafo são o sujeito (Berners-Lee) e o objeto (Londres), e o arco é o predicado (local de nascimento). Além disso, é importante notar que a direção do predicado no grafo é relevante, pois sempre apontará para o objeto. O RDF é considerado um dos alicerces da Web Semântica (HITZLER; KROTZSCH; RUDOLPH, 2009), pois é a base de muitas outras tecnologias como o RDF-Schema (W3C, 2014a), que é uma extensão do vocabulário RDF básico, Turtle (W3C, 2014b), uma sintaxe textual que permite que o grafo RDF seja escrito de uma forma mais amigável e compacta, adicionando a possibilidade de criação de abreviações para tipos de dados, e o SPARQL (PRUD'HOMMEAUX; SEABORNE, 2006).

SPARQL (PRUD'HOMMEAUX; SEABORNE, 2006) é uma linguagem de consulta RDF, seu nome é um acrônimo recursivo que significa Protocolo SPARQL e Linguagem de consulta RDF. Atualmente, o SPARQL é considerado uma recomendação do W3C, padronizado pelo RDF Data Access Working Group (DAWG) do World Wide Web Consortium e uma tecnologia chave para o desenvolvimento da Web Semântica.

Essencialmente, SPARQL é uma linguagem de consulta de correspondência de grafos. Dada uma fonte de dados D, uma consulta consiste em um padrão que é comparado com D, e os valores obtidos a partir dessa correspondência são processados para fornecer a resposta. A fonte de dados D a ser consultada pode ser composta por várias fontes. Uma consulta SPARQL consiste em três partes.

- A parte de *correspondência de padrões*, inclui vários recursos interessantes de correspondência de padrões de grafos, como partes opcionais, união de padrões, aninhamento, filtragem ou restrição de valores de correspondências possíveis e a possibilidade de escolher a fonte de dados a ser correspondida por um padrão.
- A parte dos *modificadores de solução*, que uma vez calculada a saída do padrão na forma de uma tabela de valores de variáveis, permitem modificar esses valores aplicando operadores clássicos como projeção, distinto, ordem, limite e deslocamento.
- e por fim, a *saída* (resultado) de uma consulta SPARQL pode ser de diferentes tipos: consultas sim e não, seleções de valores das variáveis que correspondem aos padrões, construção de novos triplas a partir desses valores e descrições de recursos.

Embora considerados um a um, os recursos do SPARQL são simples de descrever e entender, verifica-se que a combinação deles torna o SPARQL uma linguagem bastante complexa.

Como pode ser vista na Figura 9, a sintaxe das consultas lembra o SQL tradicional, com SELECT e WHERE, mas sem um FROM, pois a consulta não é feita em tabelas, mas sim, como descrito acima, consiste em um padrão que é comparado com a uma fonte de dados.

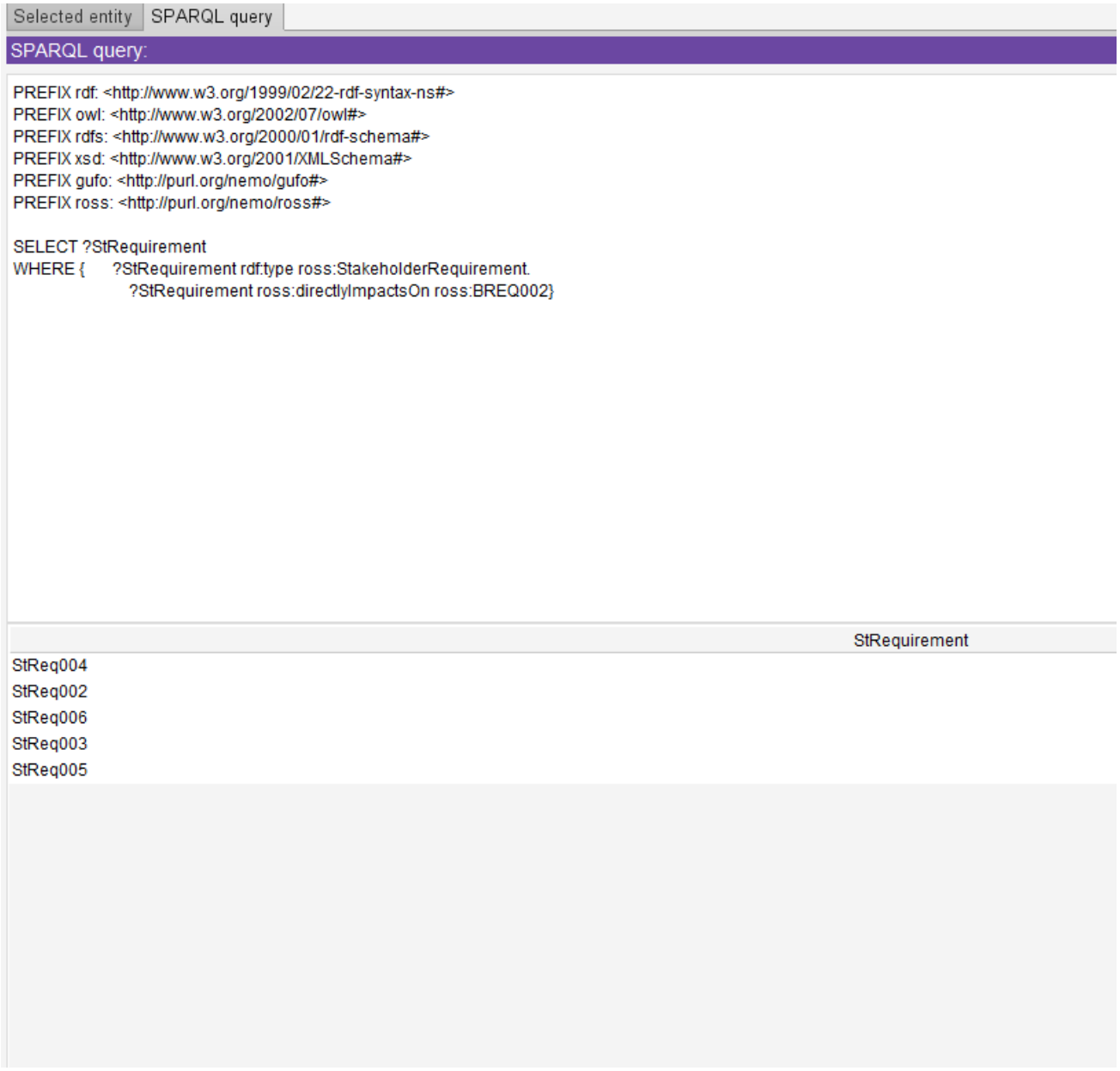

Figura 9 – Um exemplo de query SPARQL

3.2.6 Apache Jena

Apache Jena (MCBRIDE, 2001) é um *framework* Java de código aberto para a construção de aplicativos da Web Semântica, que utilizam dados interligados (*Linked Data*). A estrutura é composta de diferentes APIs interagindo para processar dados RDF podendo ser usada para pra criar e manipular grafos RDF, possui classes para representar grafos, recursos, propriedades e literais. Além de prover classes para outros esquemas bem conhecidos, bem como os próprios RDF e RDFS, Dublin Core e OWL.

O *framework* Jena inclui uma API RDF, apoio à leitura e escrita RDF em RDF/XML, N3 e N-triples, uma API OWL, armazenamento na memória e persistente, e um mecanismo de consulta SPARQL.

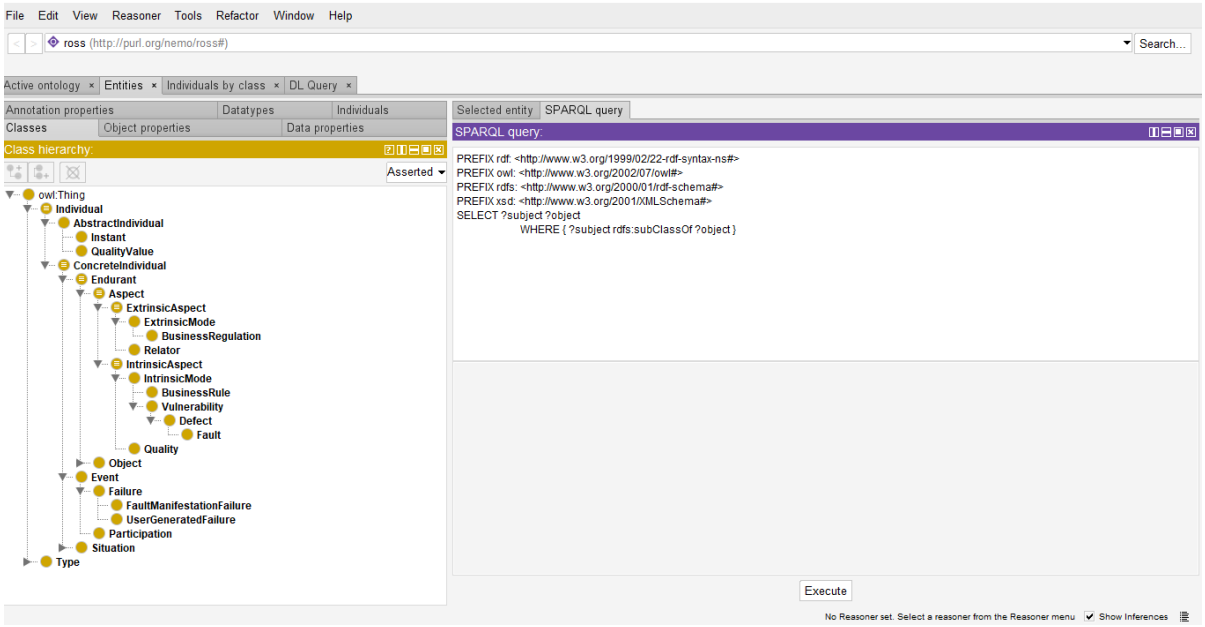

Figura 10 – Tela do Protégé com Hierarquia de classes da ROSS apresentada

3.2.7 Protégé

Protégé (NOY et al., 2003) é um editor de código aberto e sistema de aquisição de conhecimento. Fornece um conjunto de ferramentas para construir modelos de domínio e aplicativos baseados em conhecimento com ontologias. O Protégé implementa um amplo conjunto de estruturas de modelagem de conhecimento e ações de apoio para a criação, visualização e manipulação de ontologias em diversos formatos de representação. Ele pode ser personalizado para fornecer uma maneira amigável para entrada de dados e modelagem de conhecimento. Além disso, o Protégé pode ser estendido por meio de *plugins* e de sua API para a construção de ferramentas e aplicativos baseados em conhecimento. As ontologias são centrais para muitas aplicações, como portais de conhecimento científico, gerenciamento de informações e integração de sistemas, *e-commerce* e serviços da Web semântica.

A plataforma Protégé, apresentada na Figura 10, suporta duas formas principais de ontologias de modelo. Uma delas é o editor Protégé-Frames que permite aos usuários construir e preencher ontologias baseadas em frames, de acordo com o protocolo Open Knowledge Base Connectivity (OKBC). Nesse modelo, uma ontologia consiste em um conjunto de classes organizadas em uma hierarquia para representar os conceitos do domínio, um conjunto de slots associados às classes para descrever suas propriedades e relacionamentos e um conjunto dessas classes. Outra é o editor Protégé-OWL que permite aos usuários construir ontologias para a Web Semântica, em particular para o W3C, Web Ontology Language (OWL).

O Protégé foi desenvolvido pelo Centro de Pesquisa em Informática Biomédica de Stanford na Escola de Medicina da Universidade de Stanford. A versão 5.5.0 do Protégé

foi a adotada para a realização deste projeto.

3.3 ROSS – Ontologia de Referência em Sistemas de Software

Em sua pesquisa, Zave e Jackson (1997) discutem o que eles chamam de "os quatro cantos escuros da Engenharia de Requisitos". Ao fazer isso, eles esclarecem certos aspectos da natureza da Engenharia de Requisitos e demonstram a importância de certos itens de informação que são frequentemente negligenciados nessa disciplina. Nesse artigo, eles propuseram a seguinte (agora, bem conhecida) fórmula $(S, A \vdash R)$, que visa capturar que um conjunto de requisitos (R) é satisfeito por uma especificação (S) associada a um conjunto de suposições de domínio (A). Posteriormente em (GUNTER et al., 2000), a fórmula foi aprimorada para levar em consideração outros artefatos de software de relevância, como a Máquina (M), como a plataforma de programação , e o Programa (P), como unidade que se pretende implementar a especificação.

No entanto, este modelo de referência não leva em consideração o fato de que alguns desses artefatos de software podem existir em diferentes níveis de abstração através do processo de software. Por exemplo, no trabalho de Zave e Jackson, requisitos são considerados como entidades que existem apenas no ambiente externo, longe das noções de programa e máquina. Hoje em dia, porém, há um consenso claro em padrões modernos, como (ISO, IEC, 2018) e (ISO, 2017), e modelos de capacidades, como (SEI/CMU, 2010), que requisitos existem em muitos formatos durante o processo de software, sendo refinados de objetivos de alto nível para artefatos orientados para a solução. Em outro exemplo, o conceito de especificação é originalmente descrito como um artefato que existe na interface entre o mundo e a máquina. Portanto, se estamos assumindo que os requisitos existem em muitos níveis de abstração, o mesmo deve acontecer com suas especificações.

Em trabalho anterior, Duarte et al. (2018) adotou as contribuições de Zave e Jackson, juntamente com uma ontologia de Artefatos de Software (WANG et al., 2014a; WANG et al., 2014b) para desenvolver a Software Ontology (SwO) e a Ontologia de Referência de Requisitos de Software (RSRO). SwO e RSRO são duas ontologias de referência criadas com o propósito de serem reutilizadas para o desenvolvimento de uma ontologia mais específica, a *Runtime Requirements Ontology* (RRO) (DUARTE et al., 2018; DUARTE et al., 2016). A RRO, por outro lado, serve de referência conceitual para a criação e interoperabilidade de requisitos em tempo de execução. Porém, conforme prescrito no SABiO (FALBO, 2014) (que foi utilizado em seu desenvolvimento), a SwO e a RSRO foram projetadas de maneira geral, contendo apenas os conceitos necessários para suportar extensões como RRO. Por isso, elas não contemplam aspectos sociotécnicos de sistemas de software, nem alguns de seus aspectos como entidades multi-artefatos (WANG et al., 2014a). No entanto, esses aspectos são centrais para explicar como defeitos, erros e

falhas afetam os sistemas de software de uma perspectiva de análise de risco, em particular, devido à conexão entre riscos (sociais) e valores. Por exemplo, eles são necessários para caracterizar o contexto no qual os sistemas de software existem e operam, bem como o impacto causado por eventos de falha.

Nesse contexto, foi desenvolvida a Ontologia de Referência de Sistemas de Software (ROSS), um modelo de referência de domínio e ferramenta de representação de conhecimento para sistemas de software que reutiliza e complementa esses trabalhos anteriores. ROSS é baseada em padrões internacionais amplamente aceitos, como ISO 29148 (ISO, IEC, 2018), ISO 12207 (ISO, 2017) e SWEBoK (BOURQUE; FAIRLEY et al., 2014), mas também no trabalho seminal de Zave e Jackson sobre a natureza dos requisitos de software (ZAVE; JACKSON, 1997; GUNTER et al., 2000).

A apresentação da ontologia, adota as diretrizes do SABiO sobre modularização de ontologias e ROSS foi dividido em três módulos, com o objetivo de representar o próprio domínio dos sistemas de software, que pode ser dividido em três camadas:

- A camada de negócios, que pode ser vista na Figura 11, na qual agentes, como organizações e seus membros, formulam objetivos e requisitos de alto nível para atingir esses objetivos através de sistemas de software;
- A camada de sistema de software, que pode ser vista na Figura 12, e tem por objetivo fornecer requisitos para funções capazes de satisfazer os objetivos dessas entidades de negócios, bem como conectar os conceitos e entidades que existem na camada de negócios com a camada de máquina;
- A camada de máquina, que pode ser vista na Figura 13, na qual a máquina é capaz de executar uma tradução de uma especificação de programa carregada em memória para realizar sua execução, visando assim alcançar os objetivos definidos na camada de negócio.

A primeira parte da ontologia, demonstrada na Figura 11, representa o ambiente de negócio/organização onde o software existe. Requisitos de negócio são objetivos de alto nível que a organização quer que o sistema apresente. Representando o principal motivo do porquê o projeto foi iniciado, o que o projeto almeja alcançar, que métricas podem ser usadas para medir o sucesso ou falha do projeto (ISO, IEC, 2018). Para representar a relação entre metas e agentes na ROSS, foi criada a relação “tem meta”. Em UFO, objetivos são proposições, em particular, eles são o conteúdo proposicional de uma intenção inerente a um agente (no domínio do ROSS, uma organização ou um *stakeholder*). Além disso, como objetivos de um agente, os requisitos de negócios são geralmente descritos como algum tipo de artefato a ser usado, rastreado e mantido adequadamente pela organização. Este item de informação é denominado especificação de requisitos de negócios e, como

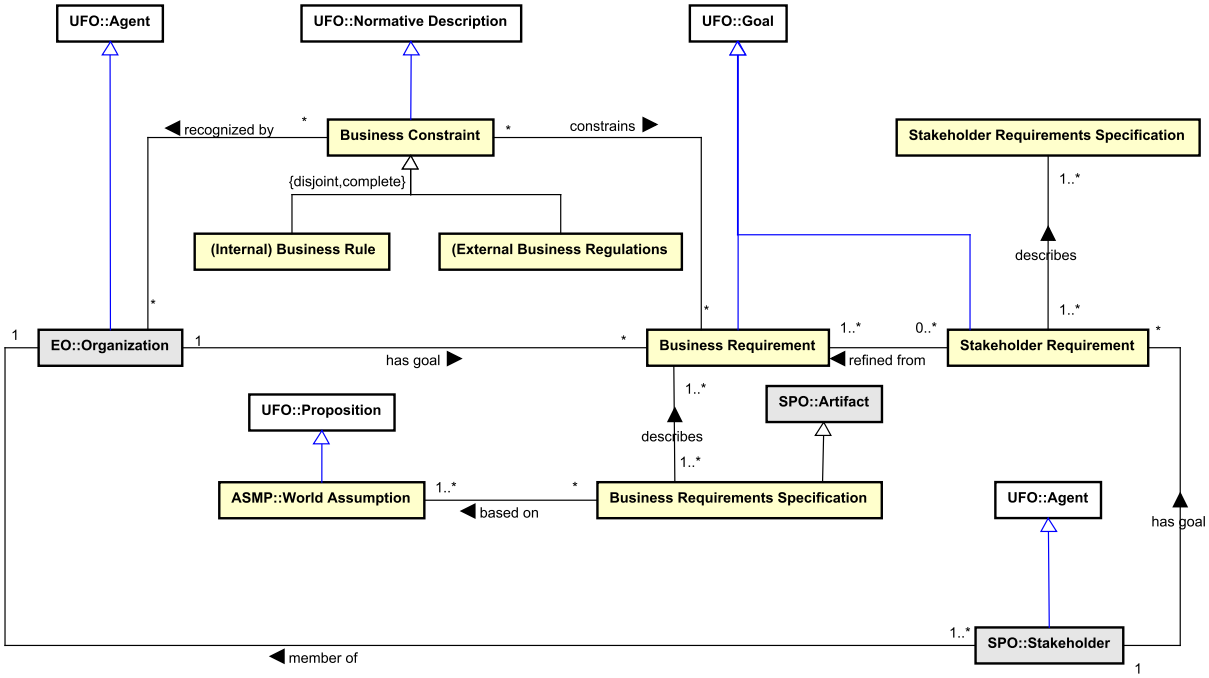

Figura 11 – Modelo Conceitual da camada de negócios de ROSS (DUARTE et al., 2021)

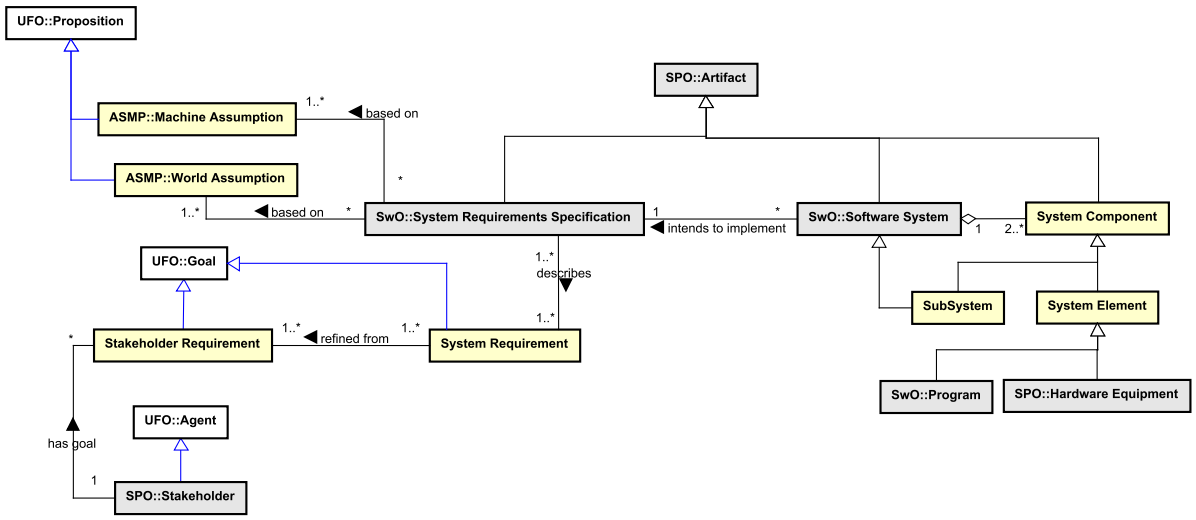

Figura 12 – Modelo Conceitual da camada de sistema de ROSS (DUARTE et al., 2021)

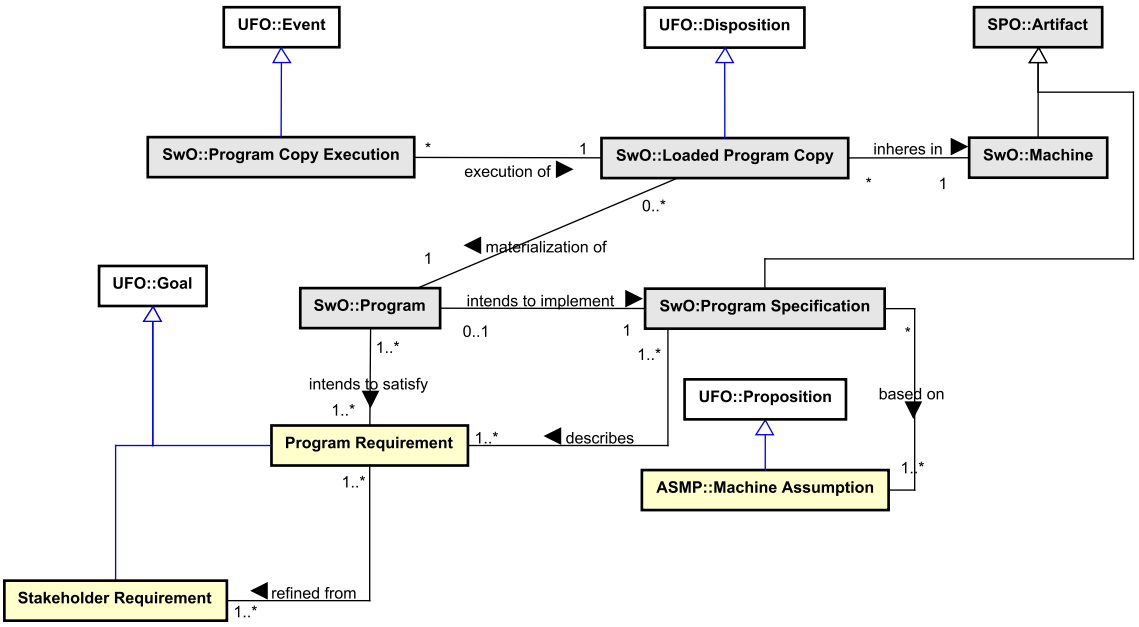

Figura 13 – Modelo Conceitual da camada de maquina de ROSS (DUARTE et al., 2021)

um produto do processo de desenvolvimento do sistema, é criado muito cedo e existirá durante todo o ciclo de vida do sistema.

Zave e Jackson (1997) defendem que as suposições/premissas (*assumptions*) devem ser tratadas como cidadãos de primeira classe em todo projeto de sistema de software, sendo documentadas e gerenciadas como qualquer outro item de configuração. Com base nisso, Wang et al. (2016) propõem uma pequena ontologia de suposições, que está sendo reutilizada pela ROSS com o prefixo ASMP. Em outras palavras, uma especificação é uma descrição possível dos requisitos com base em um conjunto de *assumptions*, ou seja, diferentes *assumptions* usadas resultarão em especificações diferentes e, se as *assumptions* usadas estiverem incorretas ou incompletas, a especificação não será capaz de descrever adequadamente o requisitos. Este fenômeno também é verdadeiro quando os requisitos estão em seus níveis mais baixos de abstração, pois são devidamente refinados para a solução do problema que deu origem ao processo de software

A segunda parte da ontologia, ilustrada na Figura 12, é centrada em torno do conceito de sistema de software, que atua como uma interface entre a máquina e o ambiente. Em seu trabalho, Zave e Jackson (1997) definem brevemente um sistema de software como um artefato geral com componentes manuais, automáticos e até abstratos (dados), separando-o do conceito de máquina. Em uma definição mais geral, sistema de software é definido pela ISO 24765 (ISO, IEC, 2017b) como uma combinação de elementos de interação organizados para atingir um ou mais objetivos declarados. O SWEBoK estende essa definição ao explicar o conceito de sistema de software como um artefato complexo e heterogêneo, pois pode ser composto por diversos elementos do sistema, como software,

hardware, firmware, pessoas, dados e até outros sistemas.

Como um tipo de artefato de software (FALBO; BERTOLLO, 2009), os sistemas de software também são desenvolvidos com base em um conjunto de requisitos. Requisitos são metas orientadas à solução para o sistema de interesse, que são baseadas em informações básicas sobre os objetivos de alto nível a serem alcançados por uma solução (ISO, IEC, 2018). Requisitos de sistema são diferentes de requisitos de negócios e requisitos das partes interessadas, uma vez que existem na perspectiva da solução, enquanto os requisitos das partes interessadas e do negócio existem na perspectiva do problema. No entanto, os requisitos do sistema são derivados dos requisitos das partes interessadas. Essa relação entre os dois tipos de requisitos fornece a conexão entre a camada de negócios e a camada de sistema. Além disso, da mesma forma que em suas contrapartes de nível superior, os requisitos do sistema são descritos em um documento chamado especificação de requisitos do sistema (ISO, IEC, 2018). Além disso, como um sistema de software pode ser composto por elementos de sistema distintos, a especificação de requisitos do sistema compila, a nível técnico, requisitos, capacidades e restrições do sistema de interesse como um todo. Por isso, depende de dois tipos de suposições, a saber, a suposição do mundo, que foram apresentadas anteriormente e a suposição da máquina, ou seja, uma suposição sobre as operações internas da máquina, que só são visíveis para a máquina. Ou seja, para que a especificação de requisitos do sistema seja criada, depende de suposições sobre o ambiente e sobre a máquina.

Por fim, a última parte da ontologia, apresentada na Figura 13, representa as partes de um sistema de software que existem dentro de uma máquina e tem como foco o conceito de programa. Wang et al. (2014b) promovem uma ampla discussão e uma ontologia de referência de artefatos de software. Com base neste trabalho, e para capturar a natureza complexa do software, em SwO (DUARTE et al., 2018), defende-se que um programa é definido como um artefato produzido durante um processo de software e que tem a finalidade de gerar um resultado no ambiente, por meio de sua execução em uma máquina (WANG et al., 2016). Além disso, programas são artefatos constituídos por código-fonte, embora não sejam idênticos ao código. O código-fonte, como uma sequência de símbolos, pode ser alterado sem alterar a identidade do programa. Neste contexto, os programas são elementos do sistema relacionados à máquina. Eles só podem cumprir sua função quando carregados (como cópia de programa carregada) e executados como eventos, chamados de execução de cópia de programa, que ocorrem dentro de uma máquina. Além disso, o objetivo do programa está diretamente relacionado à sua identidade e não a sua implementação.

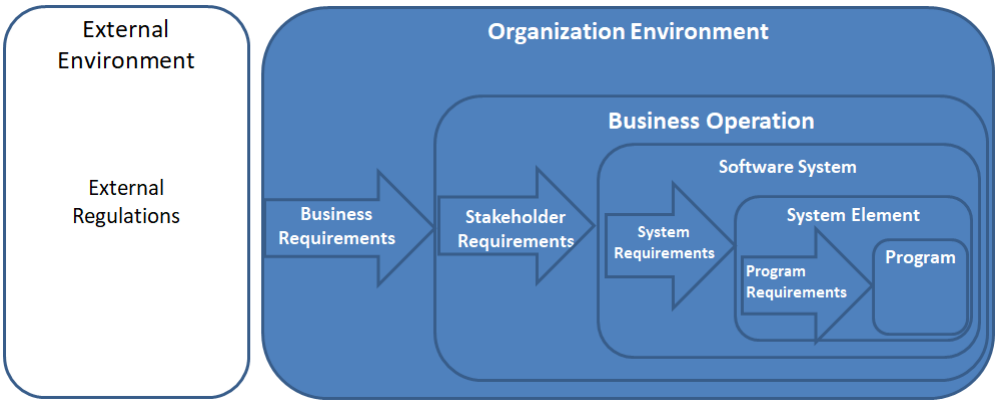

Figura 14 – Adaptação de um exemplo de escopo de requisitos em um contexto de negócios. Figura originalmente apresentada na ISO 29148 (ISO, IEC, 2018).

3.4 OSDEF - Ontologia de referência de Defeitos, Erros e Falhas de Software

O Guia de Conhecimento em Engenharia de Software (SWEBoK) (IRMAK, 2013) enfatiza, por exemplo, a necessidade de um consenso sobre a caracterização de anomalias e discute como uma classificação bem fundamentada pode ser usada em auditorias e análises de produtos. Além disso, o modelo CMMI (SEI/CMU, 2010) preconiza que as organizações devem criar ou reutilizar alguma forma de método de classificação de defeitos e falhas. Também sugere o uso de um índice de densidade de defeito para muitos produtos de trabalho que fazem parte do processo de desenvolvimento de software. Um esquema de classificação adequado pode permitir o desenvolvimento de diferentes tipos de perfis de anomalias que podem ser produzidos como um indicador da qualidade do produto. Além disso, classificar sistematicamente anomalias de software que podem ocorrer em tempo de design ou tempo de execução é uma rica fonte de dados que pode ser usada para melhorar processos e evitar a ocorrência de anomalias em projetos futuros (IEEE, 2009). Por fim, defeitos, erros e falhas têm um impacto negativo em aspectos importantes do software, como confiabilidade, eficiência, custo geral e, em última análise, vida útil. Portanto, uma melhor compreensão da natureza ontológica desses conceitos e como eles se relacionam com outros artefatos de software (por exemplo, requisitos, solicitações de mudança, relatórios e casos de teste) pode melhorar a forma como uma organização lida com esses problemas, reduzindo custos com atividades como gerenciamento de configuração e manutenção de software.

Embora existissem algumas propostas para classificar diferentes termos para anomalias de software, não existia um modelo de referência ou teoria que explicasse a natureza das diferentes anomalias de software. Em outras palavras, não existia uma ontologia de referência adequada (GUIZZARDI, 2007) voltada para a representação de defeitos, erros e falhas de software. Para suprir essa lacuna, foi proposta uma Ontologia de referência

de Defeitos, Erros e Falhas de Software (OSDEF) (DUARTE et al., 2018; DUARTE et al., 2021). Esta ontologia leva em consideração diferentes tipos de anomalias que podem existir em artefatos relacionados a software e que são mencionadas de forma recorrente no conjunto dos padrões mais relevantes da área. OSDEF é então analisada a partir desta perspectiva de análise de risco, alavancando na Ontologia Comum de Valor e Risco (COVR) (SALES et al., 2018).

Conforme mencionado anteriormente, o termo anomalia é comumente usado para se referir a uma variedade de noções de natureza ontológica distinta. Para direcionar este problema, OSDEF fornece uma conceituação ontológica dos diferentes tipos de anomalias de software que existem ao longo do ciclo de vida do software.

A Figura 15 mostra o modelo conceitual do OSDEF. O conceito central dessa ontologia é a falha, uma vez que é a ocorrência de uma falha que normalmente é percebida por um agente que opera o sistema de software. Conforme definido nas normas (ISO, IEC, 2017b; IEEE, 2009; IEEE, 2016) e empregado em geral na literatura científica (IEEE, 2016), falhas são perdurantes (eventos). Nesse sentido, a base conceitual fornecida por UFO pode nos ajudar a entender como as falhas ocorrem como eventos durante a execução do software. Em um contexto de software, uma falha é definida como um evento em que um programa não funciona como deveria, ou seja, um evento que impacta negativamente esses objetivos relevantes das partes interessadas que motivaram a criação desse software (GUIZZARDI et al., 2013). Como eventos, as falhas podem causar outras falhas em uma cadeia de eventos (por exemplo, uma falha grave em um servidor web como o Apache httpd pode fazer com que todos os seus aplicativos hospedados sofram falhas subsequentes). Conforme definido em UFO (GUIZZARDI et al., 2013), a causalidade é uma relação de ordem parcial estrita e, portanto, as falhas não podem ser suas próprias causas ou as causas de suas causas, mas as falhas podem (talvez, indiretamente) desencadear outras falhas em uma cadeia de causalidade.

Os defeitos podem existir ao longo de todo o ciclo de vida do software (CHILLAREGE, 1996). Conforme mencionado anteriormente, alguns defeitos podem (contingentemente) evitar se manifestarem nas execuções de software. Quando um defeito se manifesta, chamamos esse defeito de falha (defeito em tempo de execução). Uma falha, portanto, pode ser vista como um papel desempenhado por um defeito em relação a uma falha. Além disso, OSDEF considera a participação de agentes como entidades que participam da ocorrência de falha, seja de forma não-objetiva, através de uma ação errada ou seja de forma objetiva, onde o agente tinha a intenção de ocasionar a falha.

Conforme discutido em (FRICKER; SCHNEIDER, 2015), eventos (incluindo falhas) são entidades poligênicas que podem resultar da interação de múltiplas disposições. Por exemplo, consideramos que uma falha gerada pelo usuário pode ser causada por uma combinação de certas disposições de um sistema de software combinadas com certos

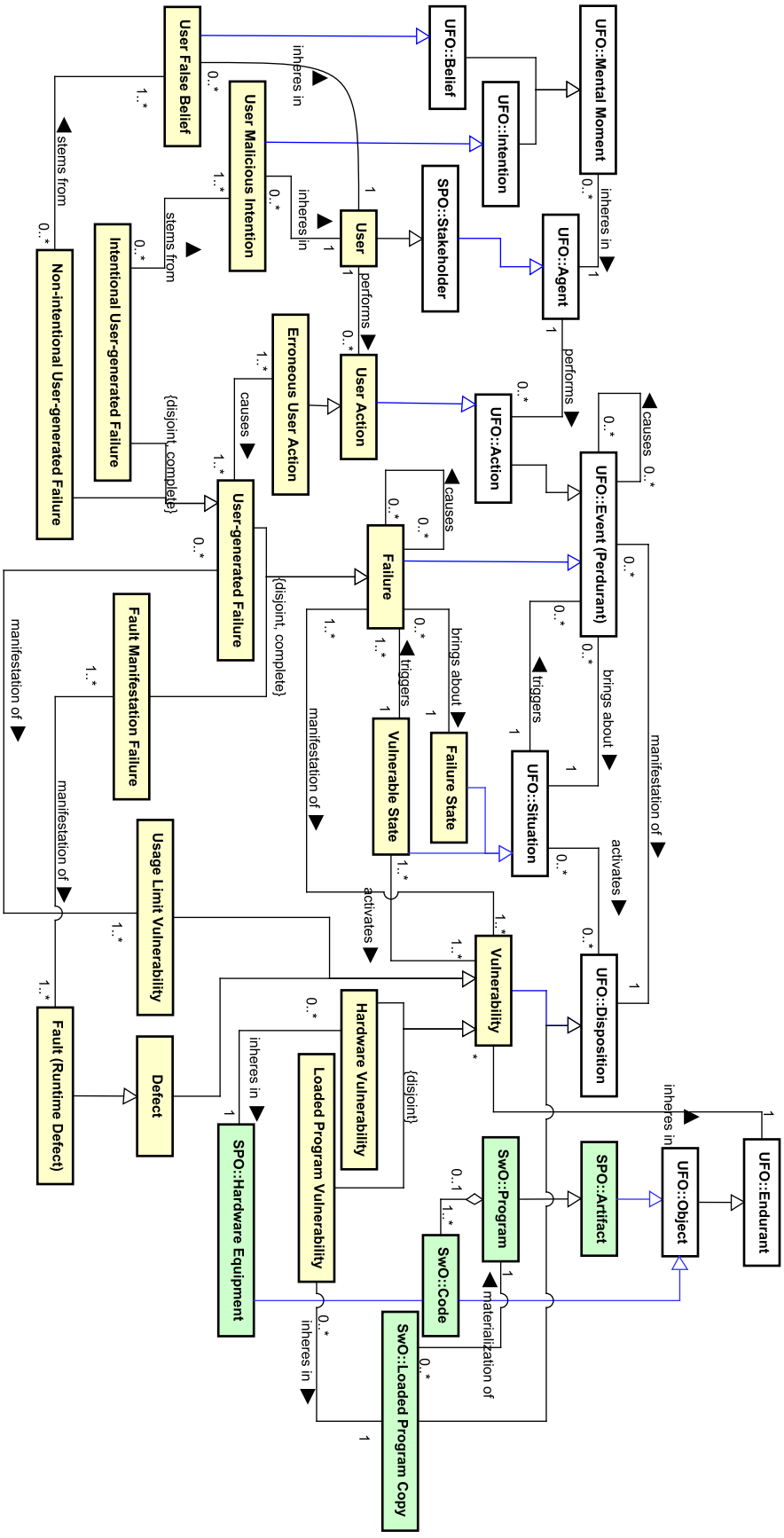

Figura 15 – Modelo conceitual de OSDEF (DUARTE et al., 2018)

momentos mentais de agentes. Esses momentos mentais incluem crenças (incluindo falsas crenças do usuário sobre suposições de domínio), bem como intenções (incluindo intenções maliciosas do usuário). Um caso particular de falha gerada pelo usuário é aquele em que esta vulnerabilidade de limite de uso é explorada de maneira intencionalmente mal-intencionada, no que é denominado um ataque (por exemplo, um usuário com intenções maliciosas pode fazer um servidor Web falhar com uma negação distribuída de ataque de serviço). Nesse caso, o servidor que está sendo atacado não tem defeito (e, portanto, nenhuma falha). Este servidor tem apenas um número limitado de solicitações que pode atender em um determinado período (uma capacidade, que é um tipo de disposição). Se esse número for excedido por um longo período, todos os recursos do sistema serão consumidos e o servidor experimentará uma falha intencional gerada pelo usuário. Essa falha pode ser tão simples quanto uma negação de serviço devido à falta de recursos ou tão crítica quanto uma falha total do sistema. Em um cenário diferente, uma falha não intencional gerada pelo usuário pode resultar da falsa crença do usuário de um coletivo de usuários acessando simultaneamente o sistema (por exemplo, como no site da Prefeitura de Vitória durante o agendamento da vacinação contra o Covid-19 em 2021).

Assim, como discutido na Seção 2.2 a rastreabilidade tem sua importância na atividade na Gerência de Requisitos (BOURQUE; FAIRLEY et al., 2014). Ramesh e Jarke (2001) sugerem que, para ter utilidade, a rastreabilidade necessariamente deve ser organizada em uma estrutura de modelagem adequada, pois modelos de referência podem ser compreendidos como uma abstração representativa de um conhecimento bem consolidado dentro de um domínio. Nesse contexto, utilizaremos as ontologias ROSS e OSDEF descritas acima, como modelos de referência para a rastreabilidade de requisitos. Para isso, utilizaremos das versões operacionais de ROSS e OSDEF, a linguagem SPARQL e o *framework* Jena para rastrear e apresentar de forma gráfica, através de uma aplicação Web, as relações, diretas e indiretas, que existem entre os diversos artefatos que compõem o ciclo de vida de um sistema de software, provendo assim uma ferramenta para visualização da rastreabilidade desses artefatos.

4 Proposta de Aplicação Web para Apresentação amigável de rastreabilidade de requisitos

Neste capítulo dissertaremos sobre algumas tecnologias e ferramentas que foram utilizadas durante a realização deste projeto.

Esse capitulo é dividido da seguinte forma: A Seção 4.1 apresenta as tecnologias empregadas na realização desse trabalho. A Seção 4.2 apresenta e discute a contribuição principal do trabalho, como as tecnologias foram utilizadas para implementá-lo e uma elucidação sobre a sua importância.

4.1 Tecnologias Aplicadas

A ferramenta desenvolvida neste projeto foi implementado utilizando a linguagem de programação Java, na plataforma Java EE 8 (Java Enterprise Edition 8). Utilizamos a plataforma em conjunto com vários *frameworks* gratuitos e de código aberto que foram desenvolvidos para auxiliar no desenvolvimento de uma aplicação Web utilizando a linguagem Java.

A Tabela 1 descreve as principais tecnologias utilizadas na implementação da aplicação.

Tabela 1 – Tecnologias utilizadas para o desenvolvimento da ferramenta.

Tecnologia	**Versão**	**Descrição**	**Propósito**
Java EE[1]	8	Especificação da plataforma Java para a criação de aplicações distribuídas, incluindo a API de Servlets para o desenvolvimento Web.	Linguagem de programação utilizada na implementação da ferramenta proposta
WidlFly[2]	22.0	Servidor de aplicações	Fornece todo o ambiente para a implantação e manutenção aplicação

[1] Para mais detalhes acesse: https://www.oracle.com/java/technologies/java-ee-8.html
[2] Para mais detalhes acesse: https://www.wildfly.org/

Tabela 1 – Tecnologias utilizadas para o desenvolvimento da ferramenta.

Tecnologia	**Versão**	**Descrição**	**Propósito**
XHTML[3]	1.0	Família de tipos de documentos e módulos que reproduzem e estendem o HTML	Utilizados para o apresentação das telas da aplicação.
Maven[4]	3.8.2	Ferramenta de automação de compilação	Dinamicamente faz o download de bibliotecas Java.
Jena[5]	4.0.0	*Framework* Java gratuito e de código aberto para a construção de aplicativos da Web Semântica, que utilizam dados interligados.	Usado para carregar o modelo operacional de ontologias de Duarte et al. (2021) e realização de consultas SPARQL nos mesmos.
CDI[6]	2.0	É uma especificação Java, responsável por fazer o controle da injeção de dependências nas aplicações.	Realizar a injeção de dependências entre as classes do sistema afim de reduzir a quantidade de código utilizado.
JSF[7]	2.3	*Framework* que permite a elaboração de interfaces com o usuário na plataforma Web.	Criar a interface com o usuário das páginas.
PrimeFaces[8]	10.0.0	Conjunto de componentes visuais para criação de interfaces gráficas com o usuário em JSF.	Criar a interface com o usuário das páginas.

4.2 Discussão do trabalho

Nesta seção apresentamos os requisitos, o projeto arquitetural e o desenvolvimento do sistema produzido neste trabalho.

[3] Para mais detalhes acesse: https://www.w3.org/TR/xhtml1/
[4] Para mais detalhes acesse: https://maven.apache.org/
[5] Para mais detalhes acesse: https://jena.apache.org/
[6] Para mais detalhes acesse: https://www.oracle.com/java/technologies/java-ee-8.html
[7] Para mais detalhes acesse: https://www.oracle.com/java/technologies/java-ee-8.html
[8] Para mais detalhes acesse: https://www.primefaces.org/

4.2.1 Levantamento de requisitos

Como mencionado no Capitulo 2, a fase de levantamento é a fase inicial do processo de Engenharia de Requisitos e envolve as atividades de descoberta dos requisitos. Para este projeto levantamos alguns requisitos funcionais e não funcionais listados nas tabelas 2 e 3 respectivamente.

Tabela 2 – Requisitos funcionais da ferramenta.

Requisito Funcional	**Descrição**
REQ001	A tela deve prover a rastreabilidade entre os artefatos das ontologias operacionais de ROSS e OSDEF.
REQ002	O sistema deve apresentar na tela as opções de seleção de artefato para a consulta.
REQ003	O sistema deve apresentar na tela um menu de seleção da tela com a consulta que se deseja fazer.

Tabela 3 – Requisitos não-funcionais da ferramenta.

Requisito Não-Funcional	**Descrição**
RNF001	O sistema deve usar as ontologias ROSS e OSDEF como modelo de referência do sistema.
RNF002	O sistema deve utilizar uma biblioteca de componentes visuais amigáveis.
RNF003	As opções de seleção da tela devem ser feitas em um dropdown.
RNF004	O sistema deve apresentar na tela somente as opções possíveis de seleção de artefato para a consulta.

4.2.2 Design de Projeto

Foi definido como o padrão de arquitetura do projeto a arquitetura do tipo MVC (*Model-View-Controller*), que por definição é um padrão arquitetural que divide os elementos da aplicação em 3 camadas. A camada de modelo é responsável por representar os dados do sistema, a camada de visualização é responsável por apresentar na tela que é a interface de comunicação com o usuário, por onde ocorre a interação do usuário com o sistema e o Controller é responsável por receber as requisições do usuário, tratá-las e respondê-las adequadamente para que sejam entregues novamente na visualização. Na Figura 16 vemos a arquitetura do projeto proposto e as relações entre as camadas, com a visão consultando o estado do modelo e o modelo fazendo notificações a camada de visão dentre outros que podem ser observados.

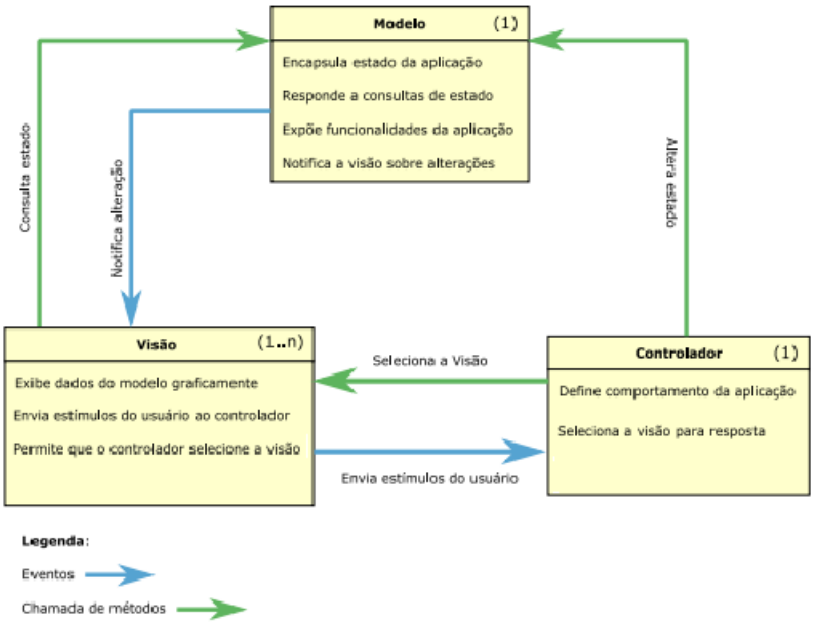

Figura 16 – Funcionamento do Padrão MVC

Podemos ver na Figura 17 o diagrama de classes desenhado para o projeto da ferramenta, que inclui a página onde serão feitas as buscas da rastreabilidade, os controladores DropdownView e MindmapView que recebem as requisições da tela e fazem as devolutivas e o ModeloOntológico que recebe a entrada da tela e retorna a lista de requisitos que são rastreados a partir do dado de entrada.

Podemos ver também na Figura 18 o único caso de uso da aplicação que consiste em um usuário acessar o sistema para a visualização da rastreabilidade do artefato selecionado.

4.2.3 Desenvolvimento

O sistema desenvolvido para a realização desse trabalho foi feito seguindo um padrão de projeto MVC, baseado em JSF e utilizando a biblioteca de visão PrimeFaces, além das várias tecnologias apresentadas na Seção 4.1, na IDE Eclipse for Java EE Developers. A arquitetura de software proposta pelo JSF para a implementação do padrão MVC tem como meta aprimorar a implementação de interfaces com o usuário, separando as representações internas de informação da camada com que o usuário da aplicação tem contato, o que diminui a necessidade de escrita de código e faz com que a camada de visão fique reduzida. O funcionamento do padrão MVC é apresentado na Figura 16. O Controlador recebe a entrada do usuário por meio de um estímulo vindo da Visão, manipula o Modelo e, se necessário, seleciona um elemento de Visão para exibir o resultado. O Modelo notifica à Visão as alterações que ocorreram em sua estrutura. A seguir dissertaremos um pouco sobre cada uma das camadas de projeto além de apresentar algumas telas do sistema.

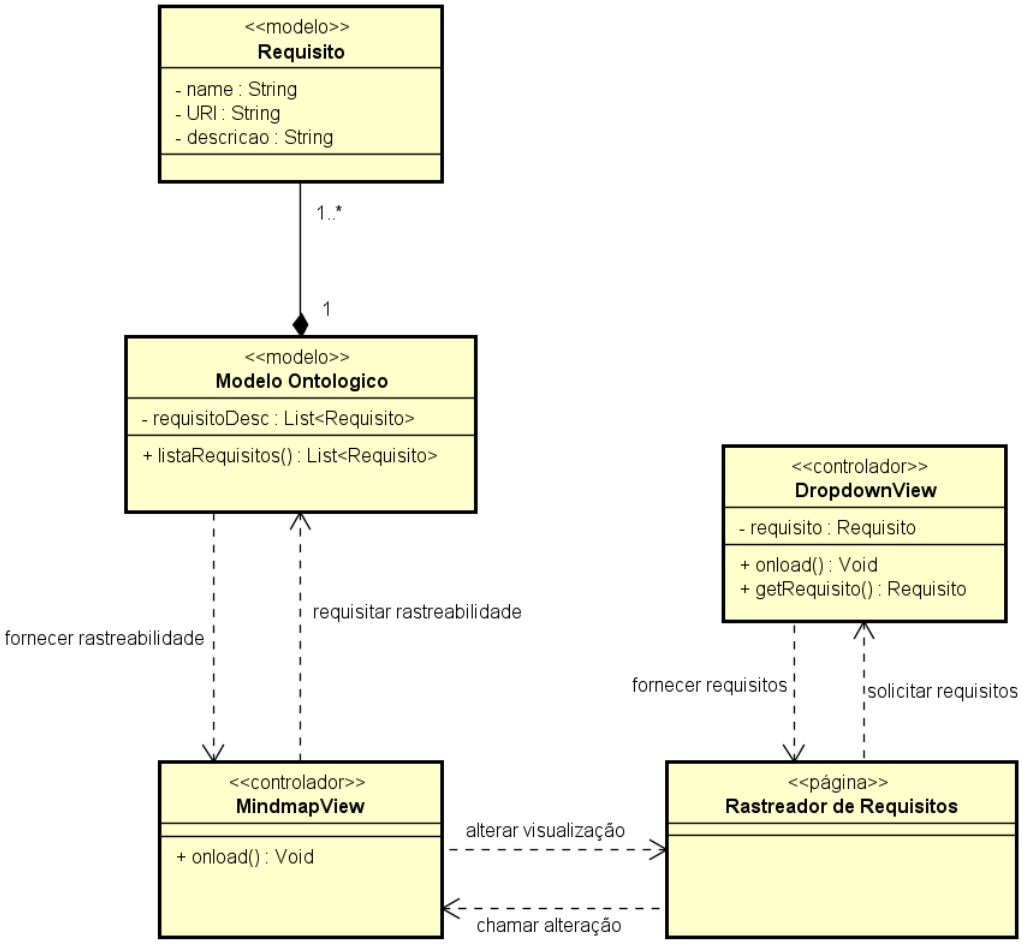

Figura 17 – Diagrama de classes da ferramenta

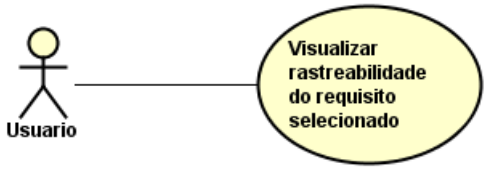

Figura 18 – Caso de uso da ferramenta

```java
package modelo;

import java.io.Serializable;

@Named
@SessionScoped
public class Requisito implements Serializable{

    private static final long serialVersionUID = 1L;
    private String name;
    private String URI;
    private String descricao;

    public Requisito(String name, String URI, String descricao) {
        super();
        this.name = name;
        this.URI = URI;
        this.descricao = descricao;
    }

    public Requisito() {}

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getURI() {
        return URI;
    }
    public void setURI(String uRI) {
        URI = uRI;
    }
    public String getDescricao() {
        return descricao;
    }
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

}
```

Figura 19 – Tela apresentando a classe Requisito

4.2.3.1 Camada de Modelo (Model)

A Camada de Modelo é responsável por representar o domínio em que estão os modelos do sistema, que no caso deste trabalho possui apenas dois, sendo estes, o *Requisito* e o *ModeloOntologico*.

No JSF, é possível que entre as camadas haja uma mistura de responsabilidades, o que significa que é possível implementar métodos de negócio em JavaBeans. Para alguns, isso pode levar a implementações menos elegantes no que diz respeito ao padrão MVC. No caso deste trabalho, temos o JavaBeans Requisitos da camada de modelo, que é puramente um modelo. Como pode ser visto na Figura 19, em que é possível ver que a classe apresenta

```
String queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\r\n"
        + "PREFIX owl: <http://www.w3.org/2002/07/owl#>\r\n"
        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n"
        + "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\r\n"
        + "PREFIX gufo: <http://purl.org/nemo/gufo#>\r\n"
        + "PREFIX ross: <http://purl.org/nemo/ross#>\r\n"
        + "\r\n"
        + "SELECT ?StRequirement\r\n"
        + "WHERE {  ?StRequirement rdf:type ross:StakeholderRequirement.\r\n"
        + "      ?StRequirement ross:directlyImpactsOn ross:"+requisito+"}";
```

Figura 20 – Consulta que retorna que Stakeholder Requirement é diretamente impactado pelo Business Requirement selecionado

apenas três atributos, sendo estes, o nome (denominado name), o atributo URI e a descrição descricao, todos strings simples com visibilidade private, o que só as torna acessível pelos *getters* e *setters*.

A Camada de Modelo ainda é composta pelos chamados beans gerenciados (ou *Managed Beans*, ou ainda *Backing Beans*), que são os responsáveis por manipular os dados que transitarão pela camada de visualização da aplicação, fazendo então a comunicação entre a camada *view* ou até outras. É o caso deste projeto, onde o *managed bean* ModeloOntologico faz comunicação com o modelo (classe Requisito) e com os dados(que no caso deste projeto é um arquivo com extensão .ttl). Esses *beans* gerenciados são *JavaBeans* comuns, normalmente utilizados na camada de modelo das aplicações Java, mas que aplicados sobre o *framework* JSF servem como modelos para componentes de interface e, por isso, podem ser acessados pela página JSF.

Neste projeto a classe ModeloOntologico é um bean gerenciado para o qual foi escolhido o escopo de sessão. Foi assim decidido pois o ModeloOntologico carrega a ontologia no sistema através da biblioteca do Jena e não muda em instante nenhum. A classe ModeloOntologico funciona para este projeto como uma classe DAO que extrai os dados de uma base de dados, neste caso, um arquivo Turtle (extensão .ttl) e é chamado pelo *managed bean* MindMap para apresentação de dados extraídos via SPARQL.

Três consultas foram escolhidas para implementação e são apresentadas nas figuras 20, 21 e 22.

A consulta da Figura 20, faz uma consulta simples em toda ontologia ROSS procurando por Stakehoder Requirements que são diretamente impactados por um requisito. Já a consulta da Figura 21 associa defeitos que foram registrados nos programas que implementam o requisito de programa selecionado como entrada para a *query*. Essa consulta é interessante pois permite a um *stakeholder* visualizar quais Programas e Requisitos de Programas estão mais associados com a ocorrência de defeitos no software.

E por fim a consulta da Figura 22 faz uma busca mais a fundo no grafo usando o conceito de subconsulta para navegar no grafo que corresponde a ontologia ROSS e

```
String queryString = "PREFIX owl: <http://www.w3.org/2002/07/owl#>\r\n"
        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n"
        + "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\r\n"
        + "PREFIX osdef: <http://purl.org/nemo/osdef#>\r\n"
        + "PREFIX ross: <http://purl.org/nemo/ross#>\r\n"
        + "PREFIX gufo: <http://purl.org/nemo/gufo#>\r\n"
        + "\r\n"
        + "SELECT ?defect \r\n"
        + "WHERE {  ?defect a osdef:Defect.\r\n"
        + " ?program a ross:Program.\r\n"
        + " ?defect gufo:inheresIn ?program.\r\n"
        + " ?progreq a ross:ProgramRequirement.\r\n"
        + " ?program  ross:implements ross:"+requisito+"\r\n"
        + "  }";
```

Figura 21 – Consulta que retorna a relação de Defeitos causados por um Requisito de programa selecionado

```
String queryString = " PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\r\n"
        + "PREFIX owl: <http://www.w3.org/2002/07/owl#>\r\n"
        + "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n"
        + "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\r\n"
        + "PREFIX gufo: <http://purl.org/nemo/gufo#>\r\n"
        + "PREFIX ross: <http://purl.org/nemo/ross#>\r\n"
        + "\r\n"
        + "SELECT ?Breq\r\n"
        + "WHERE{ ?StReq ross:directlyImpactsOn ?Breq \r\n"
        + "\r\n"
        + "{\r\n"
        + "\r\n"
        + "SELECT ?StReq\r\n"
        + "WHERE { ?ProgReq  ross:isRefinedFrom ?StReq\r\n"
        + "\r\n"
        + "{\r\n"
        + "\r\n"
        + "SELECT ?ProgReq\r\n"
        + "WHERE { ross:"+requisito+" ross:implements ?ProgReq}\r\n"
        + "\r\n"
        + "}\r\n"
        + "}\r\n"
        + "}\r\n"
        + "}\r\n"
        + "";
```

Figura 22 – Consulta que retorna que Business Requirement é implementado pelo Programa selecionado

```
<f:metadata>
    <f:viewAction action="#{dropdownView.onload()}" />
    <f:viewAction action="#{mindmapView.onload()}" />
</f:metadata>
```

Figura 23 – Tela apresentando a chamada dos metodos onLoad dos ManagedBeans no XHTML

```
<p:mindmap id="mindMapRequisitos" value="#{mindmapView.root}"
    style="width: 100%;height: 600px;border: 1px solid black;">
    <p:ajax event="select" listener="#{mindmapView.onNodeSelect}" />
    <p:ajax event="dblselect" listener="#{mindmapView.onNodeDblselect}"
        update="output" oncomplete="PF('details').show();" />
</p:mindmap>
```

Figura 24 – Tela apresentando mindmap sendo chamado no XHTML

recuperar cada parte das informações necessárias. A parte interna da consulta, que sempre é executada primeiro, recupera o Requisito do Programa que é implementado pelo programa selecionado na tela como entrada para a consulta. Essa informação é usada como dado de entrada para a consulta do meio, que apresenta o Requisito do Stakeholder relacionado ao programa selecionado e, finalmente, a consulta externa usa o resultado do meio como entrada para fornecer a resposta final. Essa consulta é a que melhor demonstra os conceitos de rastreabilidade, pois ela filtra vários elementos fazendo associações de conceitos que não estão ligados diretamente nos modelos de referência (ROSS e OSDEF) como os de requisitos do negócio e requisitos de programa.

4.2.3.2 Camada de Apresentação (View)

A Camada de apresentação, como descrito acima, foi baseada em JSF utilizando a biblioteca do PrimeFaces, em especial três componentes: o DropDown, o Menu e o MindMap para apresentação dos requisitos. Para montar as telas foram utilizados documentos XHTML para três páginas que fazem três tipos de consultas que serão descritas na sequência.

As telas apresentam um Menu para navegação entre elas. Também um componente DropDown, onde são carregados apenas os indivíduos possíveis de entrar naquela consulta, para que seja pesquisada a consulta da página selecionada relacionada a este indivíduo. Na Figura 23 podemos ver a chamada dos metodos onLoad dos *managed beans* dropdownView e mindmapView, que carregam tanto o DropDown com os valores corretos para a tela, quanto o MindMap após a seleção do individuo. E por fim na Figura 24, temos a chamada do Mindmap na tela de apresentação, cujos dados também são gerados em um *managed bean*, o MindMapView, que será descrito posteriormente.

Um dos recursos mais interessantes do JSF é que as interações entre os JavaBeans e a camada de visão da aplicação são implícitas, ou seja, não precisam ser implementadas. Isso é viabilizado através de um padrão de projeto intitulado Inversão de Controle, que faz com que o *container* seja responsável pelo mapeamento entre o JavaBean e sua camada de visão correspondente. A inversão de controle neste projeto é aplicada por meio da injeção de dependência utilizando o CDI.

A camada de visão será baseada em Facelets, cujo ciclo de vida dos componentes é mais complexo, pois sua instanciação e exibição ocorre em separado e em ordem definida, o que melhora a sua performance.

A camada de visão deste trabalho tem três telas:

- Uma tela para rastreabilidade de Stakeholder Requirements diretamente impactados por Business Requirements;
- Uma tela para rastreabilidade de Defeitos causados por um Program Requirement;
- E uma tela para rastreabilidade de Business Requirements implementados em um Programa especifico.

4.2.3.3 Camada de Controle (Control)

Existem neste projeto mais dois Managed Beans, a saber, *MindmapView*, *DropdownView*. Estes Managed Beans representam a camada de controle da aplicação. Numa aplicação baseada em JSF, utilizando a arquitetura MVC, as requisições são atendidas por controladores. Estes se responsabiliza por todas as respostas da aplicação, recebendo entradas dos usuários. Além de validar dados, preencher objetos da camada de modelo e renderizar as respostas.

O MindmapView, cuja classe é apresentada na Figura 25, é o *managed bean* responsável pela implementação do MindMap. Para ele, foi escolhido o escopo de sessão, definido por meio da anotação @SessionScoped. Isso determina que o ciclo de vida do *bean* se inicia com a primeira requisição HTTP e se encerra somente ao fim da sessão, ou seja, quando o usuário termina a navegação. Isso para que o MindMap esteja sempre a mostra independente de ter algum parâmetro selecionado ou não no DropDown. Assim foi feito, para simplificação devido a questão do tempo. Esta classe foi inteiramente extraída do site do PrimeFaces e ajustada para o uso dentro da aplicação do projeto sendo apresentado.

O DropdownView, cuja classe é apresentada na Figura 26, é o *managed bean* responsável pela implementação do DropDown. Para este também foi escolhido o escopo de sessão. Foi decidido assim, pois o DropDown continua o mesmo durante todo o uso da aplicação apenas alterando os valores dos dados dentro dele. E assim como a classe já

```java
@Named
@SessionScoped
public class MindmapView implements Serializable {

    private static final long serialVersionUID = 1L;
    private String entrada = null;
    @Inject
    private ModeloOntologico mo;
    private MindmapNode ross;
    private MindmapNode selectedNode;

    public void onload() throws FileNotFoundException {

        HttpServletRequest request = (HttpServletRequest) FacesContext.getCurrentInstance().getExternalContext()
                .getRequest();

        String pagina = request.getRequestURI().toString();

        FacesMessage msg;
        msg = new FacesMessage("PÁGINA: " + pagina);
        FacesContext.getCurrentInstance().addMessage("messages", msg);

        if (pagina.equals("/Graduacao/index.xhtml")) {

            ross = new DefaultMindmapNode(entrada, entrada, "c6a100", true);

            for (Requisito requisito : mo.listaSTRequisitos(entrada)) {
                ross.addNode(new DefaultMindmapNode(requisito.getName(), requisito.getURI(), "6e9ebf"));
            }
        } else if (pagina.equals("/Graduacao/indexREQ.xhtml")) {

            ross = new DefaultMindmapNode(entrada, entrada, "c6a100", true);

            ArrayList<Requisito> requisitos = (ArrayList<Requisito>) mo.listaRequisitos(entrada);
            if(!requisitos.isEmpty()) {
                System.out.println(requisitos.get(0).getName().toString());
                ross.addNode(new DefaultMindmapNode(requisitos.get(0).getName(), requisitos.get(0).getURI(), "6e9ebf"));
            }
        }

        else if (pagina.equals("/Graduacao/indexBREQ.xhtml")) {
            ross = new DefaultMindmapNode(entrada, entrada, "c6a100", true);

            for (Requisito requisito : mo.listaBRequisitos(entrada)) {
                ross.addNode(new DefaultMindmapNode(requisito.getName(), requisito.getURI(), "6e9ebf"));
            }

        } else {
            ross = new DefaultMindmapNode("Vazio", "Vazio", "c6a100", true);
        }

    }
```

Figura 25 – Classe do Mindmap

apresentada, esta classe foi inteiramente extraída do site do PrimeFaces e ajustada para o uso dentro da aplicação do projeto sendo apresentado.

4.3 Apresentação das telas do trabalho

A Seguir são discutidas as telas do trabalho em execução, utilizando o exemplo de caixa eletrônico apresentado por Bjork (2009) cujos requisitos são listado no Apêndice A.

Na Figura 27 já é possível ver a utilização da tela do primeiro item do menu que é a tela default. Nela fizemos uma consulta e o retorno é apresentado na forma do MindMap, com todas as ligações retornadas pela consulta SPARQL relacionada. Na figura temos, por exemplo, se eu necessito melhorar o requisito de negócio “Melhorar a confiabilidade do processo por meio de automação” (BREQ004) inserindo uma autenticação por biometria, necessito identificar quais os requisitos do *stakeholder* que serão impactados, e podemos

```java
@Named
@SessionScoped
public class DropdownView implements Serializable {

    private static final long serialVersionUID = 1L;

    private Map<String, Map<String, String>> data = new HashMap<String, Map<String, String>>();
    private String requisito;
    private Map<String, String> dropdownEntradas;


    public void onload() {

        HttpServletRequest request = (HttpServletRequest) FacesContext.getCurrentInstance().getExternalContext()
                .getRequest();

        String pagina = request.getRequestURI().toString();

        FacesMessage msg;
        msg = new FacesMessage("PÁGINA: " + pagina);
        FacesContext.getCurrentInstance().addMessage("messages", msg);


        if (pagina.equals("/Graduacao/index.xhtml")) {

            dropdownEntradas = new HashMap<String, String>();

            for (int i = 1; i < 5; i++) {
                dropdownEntradas.put("BREQ00" + i, "BREQ00" + i);
            }
        }

        else if (pagina.equals("/Graduacao/indexREQ.xhtml")) {

            dropdownEntradas = new HashMap<String, String>();

            for (int i = 1; i < 12; i++) {
                if(i<10)
                    dropdownEntradas.put("ProgReq00" + i, "ProgReq00" + i);
                if(i>9)
                    dropdownEntradas.put("ProgReq0" + i, "ProgReq0" + i);
            }

        }

        else if (pagina.equals("/Graduacao/indexBREQ.xhtml")) {

            dropdownEntradas = new HashMap<String, String>();

            for (int i = 1; i < 10; i++) {
                dropdownEntradas.put("Prog00" + i, "Prog00" + i);
            }
            dropdownEntradas.put("Prog010", "Prog010");

        } else {
            dropdownEntradas = new HashMap<String, String>();
        }
    }
```

Figura 26 – Classe do DropDown

ver que impactaria 11 requisitos, o que nos leva a um entendimento que a tarefa não é tão simples e deverá ter seus custos calculados de maneira apropriada.

A Figura 28 apresenta a segunda tela selecionada pela respectiva opção do Menu, selecionado o valor no DropDown é possível ver o resultado apresentado no componente MindMap. Nesta figura já lidamos com o conceito de defeitos no sistema. Pudemos ver que o requisito de programa “Uma operação para dispensar dinheiro deve ser implementada no software. A quantidade de dinheiro dispensado deve ser verificada pelo periférico e persistida no sistema” (PROGREQ005), causou o defeito “A Tela de Saque não verifica duas vezes a quantidade de dinheiro a ser retirada pelo cliente” (DEFECT001) de alguma forma, essa identificação leva a melhor manutenibilidade do sistema do exemplo, visto que

Figura 27 – Apresentação da primeira Tela do menu

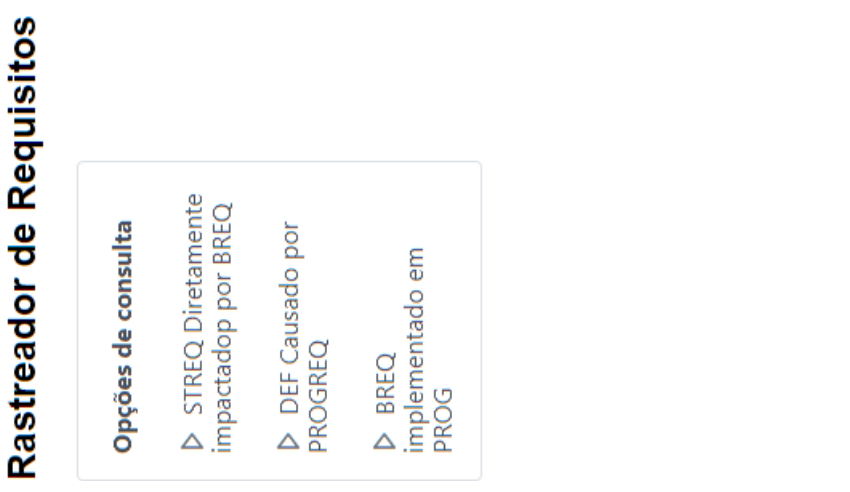

Figura 28 – Apresentação da segunda Tela do menu

Figura 29 – Apresentação da terceira Tela do menu

os desenvolvedores podem ir diretamente ao requisito causador do problema e entender e consertar o defeito.

Por fim, na Figura 29 temos a terceira tela associada ao terceiro item do menu, que, como as outras, apresenta no MindMap o resultado da busca da página associada a opção selecionada no DropDown. Na figura podemos ver que a "Conexão do sistema do banco" (PROG001), tem em sua implementação o *business requirement* "Melhorar a confiabilidade do processo por meio de automação" (BREQ004).

Nesse capítulo, então, foram apresentadas as tecnologias necessárias para a implementação da ferramenta de rastreabilidade desenvolvida. Explicando desde as ferramentas utilizadas, como a IDE Eclipse e o editor Protégé, até os *frameworks* integrados ao trabalho. Elas foram muito úteis para o desenvolvimento do sistema de rastreabilidade de requisitos, utilizando o exemplo de caixa eletrônico apresentado por Bjork (2009). Apresentamos também as 3 consultas utilizadas no projeto, elas demonstram as relações que existem entre diversos artefatos do ciclo de vida do software, que foram baseadas nas versões operacionais de ROSS e OSDEF, ontologias de domínio utilizadas como modelo de referência para implementação da ferramenta que foram detalhadas no Capitulo 3.

5 Conclusões

Esse trabalho apresenta uma ferramenta Web, que foi desenvolvida como projeto de graduação após um estudo da literatura e de diversas ferramentas disponíveis no mercado, objetivando tornar possível a visualização da rastreabilidade dos requisitos do sistema de caixa eletrônico (BJORK, 2009), utilizando as ontologias de domínio ROSS e OSDEF (DUARTE et al., 2021) discutidas no Capítulo 3 e as tecnologias java citadas no Capítulo 4.

Ao longo de todo o desenvolvimento desta monografia, apresentamos as etapas que culminaram no desenvolvimento de uma aplicação Web, baseada na plataforma Java EE e no *framework* Jena.

Tomando por base o processo proposto por Kotonya e Sommerville (1998), um processo de Engenharia de Requisitos padrão deve contemplar várias atividades, dentre elas a gerência de requisitos, a atividade que atua na mudança dos requisitos, que vimos de maneira mais detalhada na Seção 2.1.

A gerência de requisitos compreende as atividades que ajudam a equipe de desenvolvimento a identificar, controlar e rastrear requisitos ao longo do ciclo de vida do software (KOTONYA; SOMMERVILLE, 1998; PRESSMAN, 2009). Dentre os principais objetivos desse processo temos, a gerência de alteração de requisitos, a gerência de relacionamento de requisitos e a gerência de dependências entre requisitos e outros documentos produzidos no processo de software. Para isso, o processo de gerência de requisitos deve incluir várias atividades uma delas, a de rastreamento de requisitos (WIEGERS; BEATTY, 2013).

Na literatura vimos que a rastreabilidade tem sua importância e Ramesh e Jarke (2001) sugerem que, para ter utilidade, a rastreabilidade necessariamente deveria ser organizada em uma estrutura de modelagem adequada, pois modelos de referência podem ser compreendidos como uma abstração representativa de um conhecimento bem consolidado dentro de um domínio.

Tendo isso como base utilizamos como modelo de referencia as ontologias ROSS e OSDEF (DUARTE et al., 2021) que são descritas no Capitulo 3.

Por fim, acredito ter alcançado os objetivos propostos para o trabalho, visto que, foi possível desenvolver um protótipo de uma ferramenta Web utilizando as ontologias de domínio. ROSS e OSDEF foram utilizadas como modelos de referência para a criação da ferramenta web, de forma que a mesma foi capaz de demonstrar a rastreabilidade dos requisitos de um sistema de caixa eletrônico, através dos relacionamentos existentes nas

ontologias e da linguagem SPARQL.

5.1 Limitações

Uma das grandes limitações do trabalho é que as consultas foram adicionadas diretamente no código, o que limitou a ferramenta a apenas 3 consultas.

Outra limitação foi o uso de concatenação de strings ao invés de um QueryFactory, o que dificulta a manutenção e pode ser entrada para falhas de segurança.

5.2 Trabalhos Futuros

Os trabalhos futuros com relação ao tema estudado podem ser:

- Pensando na ampliação do projeto podemos pensar na criação dinâmica de consultas e armazenamento das mesmas em uma base de dados que possa ser acessada por outros usuários;
- Tendo como foco a gerência e a rastreabilidade de requisitos, outro bom ponto é a criação de perfis para que usuários possam compartilhar consultas e criação compartilhada de consultas;
- Utilização de um QueryFactory ao invés da concatenação de strings nas consultas;
- Criação de uma matriz de rastreabilidade dinâmica que mostra todos os conceitos e ao clicar em um mostra os requisitos conectados;
- Comprovar a usabilidade da ferramenta.

Referências

ALMEIDA, J. P. A. et al. *gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO)*. 2019. <http://purl.org/nemo/doc/gufo>. Accessed: 2021-08-22. Citado na página 35.

ALONSO-RORÍS, V. M. et al. Towards a cost-effective and reusable traceability system. a semantic approach. *Computers in Industry*, Elsevier, v. 83, p. 1–11, 2016. Citado 3 vezes nas páginas 17, 25 e 26.

AURUM, A.; WOHLIN, C. *Engineering and managing software requirements*. [S.l.]: Springer, 2005. v. 1. Citado na página 19.

BARCELLOS, M. P.; FALBO, R. de A.; MORO, R. D. A well-founded software measurement ontology. In: *Proc. of the 6th International Conference on Formal Ontology in Information Systems (FOIS 2010)*. [S.l.: s.n.], 2010. p. 213–226. Citado na página 32.

BENEVIDES, A. B. et al. Representing a reference foundational ontology of events in sroiq. *Applied Ontology*, IOS Press, v. 14, n. 3, p. 293–334, 2019. Citado na página 32.

BJORK, R. C. *ATM Simulation*. 2009. <http://www.cs.gordon.edu/courses/cs211/ATMExample/>. Accessed: 2021-08-14. Citado 5 vezes nas páginas 23, 60, 65, 66 e 74.

BORST, W. N.; BORST, W. Construction of engineering ontologies for knowledge sharing and reuse. 1997. Citado na página 28.

BOURQUE, P.; FAIRLEY, R. E. et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. [S.l.]: IEEE Computer Society Press, 2014. Citado 4 vezes nas páginas 15, 23, 42 e 49.

BRINGUENTE, A. C. de O.; FALBO, R. de A.; GUIZZARDI, G. Using a foundational ontology for reengineering a software process ontology. *Journal of Information and Data Management*, v. 2, n. 3, p. 511, 2011. Citado na página 32.

BRINGUENTE, A. C. O.; FALBO, R. A.; GUIZZARDI, G. Using a Foundational Ontology for Reengineering a Software Process Ontology. *Journal of Information and Data Management*, Brazilian Computer Society, v. 2, n. 3, p. 511–526, 2011. Citado na página 32.

CHENG, B. H.; ATLEE, J. M. Research directions in requirements engineering. In: IEEE COMPUTER SOCIETY. *2007 Future of Software Engineering*. [S.l.], 2007. p. 285–303. Citado na página 15.

CHILLAREGE, R. Orthogonal defect classification. *Handbook of Software Reliability Engineering*, IEEE CS Press, p. 359–399, 1996. Citado na página 47.

CLELAND-HUANG, J. et al. Best practices for automated traceability. *Computer*, IEEE, v. 40, n. 6, 2007. Citado na página 25.

DAHLSTEDT, Å. G.; PERSSON, A. Requirements interdependencies: state of the art and future challenges. In: *Engineering and managing software requirements*. [S.l.]: Springer, 2005. p. 95–116. Citado na página 26.

DUARTE, B. B. *An Ontology-based Reference Model for the Software Systems Domain with a focus on Requirements Traceability*. Tese (PhD Thesis) — Universidade Federal do Espírito Santo, Brasil, 2021. Em construção. Citado na página 74.

DUARTE, B. B. et al. Towards an ontology of software defects, errors and failures. In: SPRINGER. *International Conference on Conceptual Modeling*. [S.l.], 2018. p. 349–362. Citado 4 vezes nas páginas 9, 16, 47 e 48.

DUARTE, B. B. et al. An ontological analysis of software system anomalies and their associated risks. *Data & Knowledge Engineering*, Elsevier, v. 134, p. 101892, 2021. Citado 12 vezes nas páginas 9, 16, 17, 28, 31, 32, 37, 43, 44, 47, 51 e 66.

DUARTE, B. B. et al. Towards an ontology of requirements at runtime. In: IOS PRESS. *Proc. of the 9th International Conference on Formal Ontology in Information Systems (FOIS 2016)*. [S.l.], 2016. v. 283, p. 255. Citado na página 41.

DUARTE, B. B. et al. Ontological foundations for software requirements with a focus on requirements at runtime. *Applied Ontology*, p. to appear, 2018. Citado 3 vezes nas páginas 32, 41 e 45.

ESPINOZA, A.; GARBAJOSA, J. A study to support agile methods more effectively through traceability. *Innovations in Systems and Software Engineering*, Springer, v. 7, n. 1, p. 53–69, 2011. Citado 2 vezes nas páginas 17 e 25.

EVERMANN, J.; WAND, Y. An ontological examination of object interaction in conceptual modeling. In: *Proceedings of the Workshop on Information Technologies and Systems WITS*. [S.l.: s.n.], 2001. v. 1, p. 15–16. Citado na página 30.

FALBO, R. A. SABiO: Systematic Approach for Building Ontologies. In: GUIZZARDI, G. et al. (Ed.). *Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering*. Rio de Janeiro, RJ, Brasil: CEUR, 2014. Citado na página 41.

FALBO, R. d. A. et al. Organizing ontology design patterns as ontology pattern languages. In: SPRINGER. *Extended Semantic Web Conference*. [S.l.], 2013. p. 61–75. Citado na página 31.

FALBO, R. D. A.; BERTOLLO, G. A software process ontology as a common vocabulary about software processes. *International Journal of Business Process Integration and Management*, Inderscience Publishers, v. 4, n. 4, p. 239–250, 2009. Citado na página 45.

FALBO, R. D. A.; GUIZZARDI, G.; DUARTE, K. C. An ontological approach to domain engineering. In: *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. [S.l.: s.n.], 2002. p. 351–358. Citado na página 29.

FALBO, R. de A. Engenharia de requisitos. 2012. Citado na página 23.

FRICKER, S. A.; SCHNEIDER, K. (Ed.). *Requirements Engineering: Foundation for Software Quality - 21st International Working Conference, REFSQ 2015, Essen, Germany, March 23-26, 2015. Proceedings*, v. 9013 de *Lecture Notes in Computer Science*, (Lecture Notes in Computer Science, v. 9013). [S.l.]: Springer, 2015. ISBN 978-3-319-16100-6. Citado na página 47.

GOTEL, O. C.; FINKELSTEIN, C. An analysis of the requirements traceability problem. In: IEEE. *Requirements Engineering, 1994., Proceedings of the First International Conference on*. [S.l.], 1994. p. 94–101. Citado 3 vezes nas páginas 23, 24 e 25.

GUARINO, N. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. [S.l.]: IOS press, 1998. v. 46. Citado 5 vezes nas páginas 9, 28, 29, 30 e 31.

GUIZZARDI, G. *Ontological Foundations for Structural Conceptual Models*. Tese (PhD Thesis) — University of Twente, The Netherlands, 2005. Citado 6 vezes nas páginas 18, 29, 30, 31, 32 e 37.

GUIZZARDI, G. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, IOS Press, v. 155, p. 18, 2007. Citado 5 vezes nas páginas 9, 18, 29, 31 e 46.

GUIZZARDI, G.; FALBO, R. de A.; GUIZZARDI, R. S. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. In: *Proc. of the 11th Iberoamerican Conference on Software Engineering (CIbSE)*. [S.l.: s.n.], 2008. p. 127–140. Citado 3 vezes nas páginas 30, 32 e 37.

GUIZZARDI, G.; HERRE, H.; WAGNER, G. Towards ontological foundations for uml conceptual models. In: SPRINGER. *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*. [S.l.], 2002. p. 1100–1117. Citado na página 30.

GUIZZARDI, G. et al. Towards Ontological Foundations for the Conceptual Modeling of Events. In: *Proc. of the 32th International Conference on Conceptual Modeling*. [S.l.]: Springer, 2013. p. 327–341. Citado 5 vezes nas páginas 18, 31, 32, 37 e 47.

GUIZZARDI, R. S. S. *Agent-oriented Constructivist Knowledge Management*. Tese (PhD Thesis) — University of Twente, The Netherlands, 2006. Citado 2 vezes nas páginas 31 e 32.

GUIZZARDI, R. S. S. et al. Ontological distinctions between means-end and contribution links in the i* framework. In: *Proceedings of the 32th International Conference on Conceptual Modeling, ER*. Hong-Kong, China: [s.n.], 2013. p. 463–470. Citado na página 47.

GUNTER, C. A. et al. A del for requirements and specifications. *IEEE Software*, IEEE, v. 17, n. 3, p. 37–43, 2000. Citado 2 vezes nas páginas 41 e 42.

HITZLER, P.; KROTZSCH, M.; RUDOLPH, S. *Foundations of semantic web technologies*. [S.l.]: CRC press, 2009. Citado 2 vezes nas páginas 37 e 38.

IEEE. *IEEE 1044: Standard Classification for Software Anomalies*. [S.l.], 2009. Citado 3 vezes nas páginas 16, 46 e 47.

IEEE. *IEEE 1012: Standard for System, Software, and Hardware Verification and Validation*. [S.l.], 2016. Citado na página 47.

IRMAK, N. Software is an abstract artifact. *Grazer Philosophische Studien*, Rodopi, v. 86, n. 1, p. 55–72, 2013. Citado 3 vezes nas páginas 15, 16 e 46.

ISO, I. *ISO/IEC International Standard - Systems and software engineering – Software life cycle process*. [S.l.], 2017. 1-157 p. Citado 2 vezes nas páginas 41 e 42.

ISO, IEC. *ISO/IEC International Standard - Software and systems engineering – Methods and tools for variability traceability in software and systems product line*. [S.l.], 2017. 35 p. Citado na página 23.

ISO, IEC. *ISO/IEC International Standard - Systems and software engineering – Vocabulary*. [S.l.], 2017. 1-541 p. Citado 3 vezes nas páginas 15, 44 e 47.

ISO, IEC. *IEEE. 29148: 2018-Systems and software engineering-Requirements engineering*. [S.l.], 2018. Citado 6 vezes nas páginas 9, 23, 41, 42, 45 e 46.

JASPER, R.; USCHOLD, M. et al. A framework for understanding and classifying ontology applications. In: CITESEER. *Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW*. [S.l.], 1999. v. 99, p. 16–21. Citado na página 29.

KANNENBERG, A.; SAIEDIAN, H. Why software requirements traceability remains a challenge. *CrossTalk The Journal of Defense Software Engineering*, v. 22, n. 5, p. 14–19, 2009. Citado 3 vezes nas páginas 9, 24 e 25.

KIROVA, V. et al. Effective requirements traceability: Models, tools, and practices. *Bell Labs Technical Journal*, Wiley Online Library, v. 12, n. 4, p. 143–157, 2008. Citado na página 25.

KOTONYA, G.; SOMMERVILLE, I. *Requirements engineering: processes and techniques*. [S.l.]: Wiley Publishing, 1998. Citado 5 vezes nas páginas 9, 20, 21, 22 e 66.

LINDVALL, M.; SANDAHL, K. Practical implications of traceability. *Software: Practice and Experience*, Wiley Online Library, v. 26, n. 10, p. 1161–1180, 1996. Citado na página 24.

MCBRIDE, B. Jena: Implementing the rdf model and syntax specification. In: *SemWeb*. [S.l.: s.n.], 2001. v. 40, p. 23–28. Citado na página 39.

NARDI, J. C.; FALBO, R. A. Evolving a Software Requirements Ontology. In: *Proc. of the 34th Conferencia Latinoamericana de Informatica (CLEI 08)*. Santa Fe, Argentina: [s.n.], 2008. Citado na página 30.

NOY, N. F. et al. Protégé-2000: an open-source ontology-development and knowledge-acquisition environment. In: *AMIA... Annual Symposium proceedings. AMIA Symposium*. [S.l.: s.n.], 2003. p. 953–953. Citado na página 40.

NUSEIBEH, B.; EASTERBROOK, S. Requirements engineering: a roadmap. In: *Proceedings of the Conference on the Future of Software Engineering*. [S.l.: s.n.], 2000. p. 35–46. Citado na página 21.

OPDAHL, A. L.; HENDERSON-SELLERS, B. Grounding the oml metamodel in ontology. *Journal of Systems and Software*, Elsevier, v. 57, n. 2, p. 119–143, 2001. Citado na página 30.

PFLEEGER, S. L. *Engenharia de software: teoria e prática*. [S.l.]: Prentice Hall, 2004. Citado na página 21.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software: Uma abordagem profissional*. [S.l.]: McGraw Hill Brasil, 2016. Citado na página 19.

PRESSMAN, R. S. *Engenharia de Software-7*. [S.l.]: Amgh Editora, 2009. Citado 3 vezes nas páginas 21, 22 e 66.

PRUD'HOMMEAUX, E.; SEABORNE, A. *SPARQL Query Language for RDF. W3C Candidate Rec. 6 April 2006*. 2006. Citado na página 38.

RAMESH, B.; JARKE, M. Toward reference models for requirements traceability. *IEEE transactions on software engineering*, IEEE, n. 1, p. 58–93, 2001. Citado 5 vezes nas páginas 17, 25, 26, 49 e 66.

RAMESH, B. et al. Implementing requirements traceability: a case study. In: IEEE. *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*. [S.l.], 1995. p. 89–95. Citado na página 23.

RAMESH, B. et al. Requirements traceability: Theory and practice. *Annals of software engineering*, Springer, v. 3, n. 1, p. 397–415, 1997. Citado na página 23.

REGAN, G. et al. The barriers to traceability and their potential solutions: Towards a reference framework. In: IEEE. *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. [S.l.], 2012. p. 319–322. Citado 2 vezes nas páginas 24 e 25.

ROCHA, A. R. C. d. et al. *Qualidade de software: teoria e prática*. [S.l.]: São Paulo: Prentice Hall, 2001. Citado na página 21.

SALES, T. P. et al. The common ontology of value and risk. In: SPRINGER. *International Conference on Conceptual Modeling*. [S.l.], 2018. p. 121–135. Citado na página 47.

SEI/CMU. CMMI® for Development, Version 1.3, Improving processes for developing better products and services. *no. CMU/SEI-2010-TR-033. Software Engineering Institute*, 2010. Citado 4 vezes nas páginas 16, 23, 41 e 46.

SILVA, E. D. O. D.; FILHO, J. L.; GONÇALVES, G. Improving analysis patterns in the geographic domain using ontological meta-properties. In: *ICEIS (3-1)*. [S.l.: s.n.], 2008. p. 256–261. Citado na página 30.

SMITH, B. Logic and formal ontology. 1989. Citado na página 28.

SOMMERVILLE, I. *Software engineering 10th edition*. [S.l.]: Pearson Education, 2016. ISBN 1-292-09613-6. Citado 2 vezes nas páginas 19 e 21.

SOUZA, V. E. S. et al. Requirements-driven software evolution. *Computer Science - Research and Development*, Springer, v. 28, n. 4, p. 311–329, nov 2013. Citado na página 32.

TOGNERI, D. Apoio automatizado à engenharia de requisitos cooperativa. *Universidade Federal do Espírito Santo (UFES). Vitória*, 2002. Citado na página 22.

TUFAIL, H. et al. A systematic review of requirement traceability techniques and tools. In: IEEE. *System Reliability and Safety (ICSRS), 2017 2nd International Conference on*. [S.l.], 2017. p. 450–454. Citado 2 vezes nas páginas 24 e 25.

W3C. *Resource Description Framework*. 2004. <https://www.w3.org/TR/rdf-concepts/>. Accessed: 2021-08-15. Citado 2 vezes nas páginas 9 e 37.

W3C. *RDF Schema*. 2014. <https://www.w3.org/TR/rdf-schema/>. Accessed: 2021-08-14. Citado na página 38.

W3C. *Terse RDF Triple Language*. 2014. <https://www.w3.org/TR/turtle/>. Accessed: 2021-08-14. Citado na página 38.

WAND, Y.; STOREY, V. C.; WEBER, R. An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems (TODS)*, ACM New York, NY, USA, v. 24, n. 4, p. 494–528, 1999. Citado na página 30.

WANG, X. et al. Software as a social artifact: a management and evolution perspective. In: *International Conference on Conceptual Modeling*. [S.l.]: Springer, 2014. p. 321–334. Citado 2 vezes nas páginas 15 e 41.

WANG, X. et al. Towards an Ontology of Software: a Requirements Engineering Perspective. In: *Proceedings of the 8th International Conference on Formal Ontology in Information Systems*. Rio de Janeiro, RJ, Brasil: IOS Press, 2014. v. 267, p. 317–329. Citado 3 vezes nas páginas 15, 41 e 45.

WANG, X. et al. How software changes the world: The role of assumptions. In: *Tenth IEEE International Conference on Research Challenges in Information Science, RCIS*. Grenoble, France: [s.n.], 2016. p. 1–12. Citado 2 vezes nas páginas 44 e 45.

WIEGERS, K.; BEATTY, J. *Software requirements*. [S.l.]: Pearson Education, 2013. Citado 4 vezes nas páginas 9, 21, 22 e 66.

ZAVE, P.; JACKSON, M. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, v. 6, n. 1, p. 1–30, jan 1997. Citado 3 vezes nas páginas 41, 42 e 44.

ZHANG, H. et al. Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*, Elsevier, v. 56, n. 1, p. 40–53, 2014. Citado na página 26.

A Apêndice

Esse apêndice apresenta o conjunto de requisitos, programas e defeitos utilizados como exemplo durante o desenvolvimento deste projeto. Os dados apresentados nas tabelas representam os elementos e características centrais de um protótipo de sistema de caixa eletrônico originalmente fornecida por (BJORK, 2009). Em (DUARTE, 2021) esse conjunto foi analisado e categorizado nos diferentes conceitos da ROSS, inferindo novos requisitos para obter um conjunto completo. Sobre eles desenvolvemos o protótipo de aplicação Web apresentada neste trabalho.

Tabela 4 – Requisitos de Stakeholder para o Sistema de Caixa eletrônico (ATM).

Requisitos	**Descrição**
STREQ001	O caixa eletrônico se comunicará com o computador do banco por meio de um link de comunicação apropriado.
STREQ002	O caixa eletrônico atenderá um cliente por vez. Ao cliente será solicitado a inserção de um cartão na ATM e inserir um número de identificação pessoal (PIN) - ambos os quais serão enviados ao banco para validação como parte de cada transação. O cliente poderá então realizar uma ou mais transações. O cartão ficará retido na máquina até que o cliente indique que não deseja mais realizar nenhuma transação, então o cartão será devolvido - exceto conforme indicado abaixo.
STREQ003	O cliente deve ser capaz de fazer um saque em dinheiro de qualquer conta vinculada ao cartão, em múltiplos de $ 20,00. Antes que o dinheiro seja dispensado é necessária a aprovação do banco.
STREQ004	O cliente deve poder fazer um depósito em qualquer conta vinculada ao cartão, consistindo em dinheiro e / ou cheques em envelope. O cliente irá inserir o valor do depósito no caixa eletrônico, sujeito a verificação manual quando o envelope for retirado da máquina por um operador. A aprovação deve ser obtida do banco antes de aceitar fisicamente o envelope.
STREQ005	O cliente deve ser capaz de fazer uma transferência de dinheiro entre quaisquer duas contas vinculadas ao cartão.
STREQ006	O cliente deve ser capaz de fazer uma consulta de saldo de qualquer conta vinculada ao cartão.
STREQ007	Um cliente deve ser capaz de abortar uma transação em andamento pressionando a tecla Cancelar em vez de responder a uma solicitação da máquina.

STREQ008	O caixa eletrônico comunicará cada transação ao banco e obterá a verificação de que foi permitida pelo banco. Normalmente, uma transação será considerada concluída pelo banco depois de aprovada.
STREQ009	Se o banco determinar que o PIN do cliente é inválido, o cliente precisará inserir o PIN novamente antes que a transação possa prosseguir. Se o cliente não conseguir inserir o PIN com sucesso após três tentativas, o cartão será retido permanentemente pela máquina e o cliente terá que entrar em contato com o banco para recuperá-lo.
STREQ010	Se uma transação falhar por qualquer motivo que não seja um PIN inválido, o caixa eletrônico exibirá uma explicação do problema e, em seguida, perguntará ao cliente se ele deseja fazer outra transação.
STREQ011	O caixa eletrônico fornecerá ao cliente um recibo impresso para cada transação bem-sucedida, mostrando a data, hora, localização da máquina, tipo de transação, conta(s), montante, saldo final e saldo(s) disponível (s) da conta afetada (para contabilizar as transferências).
STREQ012	O caixa eletrônico também manterá um registro interno de transações para facilitar a resolução de ambiguidades decorrentes de uma falha de hardware no meio de uma transação. As inscrições serão feitas no registro quando o caixa eletrônico for iniciado e desligado, para cada mensagem enviada ao Banco(juntamente com a resposta do mesmo, caso seja prevista) para a dispensa de dinheiro e para a recepção de envelope. As entradas de registro podem conter números de cartão e valores em dólares, mas por segurança nunca conterão um PIN

Tabela 7 – Requisitos de Programa para o Sistema de Caixa eletrônico (ATM).

Requisitos	**Descrição**	**Derivado de**
PROGREQ001	Seguindo o padrão da organização, a classe *Network-ToBank* deve implementar métodos para abrir e fechar uma conexão com o sistema bancário.	STREQ001
PROGREQ002	As operações de leitura, ejeção e retenção de um cartão devem ser implementadas no sistema.	STREQ002

Tabela 5 – Requisitos de Negócio para o Sistema de Caixa eletrônico (ATM).

Requisitos	**Descrição**
BREQ001	Reduzir o tempo de serviço para o cliente.
BREQ002	Redução de custos com empregados.
BREQ003	Permitir ao cliente atendimento fora de horário comercial.
BREQ004	Melhorar a confiabilidade do processo por meio de automação.

Tabela 7 – Requisitos de Programa para o Sistema de Caixa eletrônico (ATM).

Requisitos	**Descrição**	**Derivado de**
PROGREQ003	A tela de retirada deve confirmar com o cliente a quantidade de dinheiro a ser retirada.	STREQ003
PROGREQ004	Antes de iniciar uma transação de saque, o Sistema do caixa eletrônico deve confirmar se o cliente possui fundos (dinheiro mais limites de conta) para realizar o saque. Se o cliente não tiver fundos suficientes, uma mensagem será exibida para o cliente: "Não há fundos suficientes para realizar esta operação". Um registro deve ser feito.	STREQ003
PROGREQ005	Uma operação para dispensar dinheiro deve ser implementada no software. A quantidade de dinheiro dispensado deve ser verificada pelo periférico e persistida no sistema.	STREQ003
PROGREQ006	O caixa eletrônico deve confirmar se o envelope foi devidamente depositado lendo um código de barras impresso no envelope. Uma mensagem será exibida para o cliente: " O depósito agora está sujeito à análise do banco ". Se o cliente deixar de depositar o envelope dentro do tempo limite, ou pressionar cancelar, a transação será considerada cancelada e uma mensagem será exibida para o cliente: " Depósito não concluído ". Um registro deve ser feito.	STREQ004
PROGREQ007	Antes do início de uma transação de transferência, o Sistema do caixa eletrônico deve confirmar com o sistema do banco se o cliente possui fundos (dinheiro mais limites da conta) para realizar a transferência. Se o cliente não tiver fundos suficientes, uma mensagem será exibida para o cliente: " Não há fundos suficientes para realizar esta operação ". Um registro deve ser feito.	STREQ005
PROGREQ008	A tela de transferência deve solicitar ao cliente a confirmação de ambas as contas (número, proprietário e agência), antes do início da transação.	STREQ005

Tabela 7 – Requisitos de Programa para o Sistema de Caixa eletrônico (ATM).

Requisitos	**Descrição**	**Derivado de**
PROGREQ09	A consulta de saldo deve ser apresentada principalmente na tela. A tela de saldo também deverá apresentar um botão onde o cliente poderá optar por imprimir o saldo.	STREQ006
PROGREQ010	Uma restrição deve ser implementada para evitar que o cliente imprima o saldo mais de 2 vezes no mesmo dia.	STREQ006
PROGREQ011	O botão Cancelar presente no console do cliente deve lançar uma exceção que precisa ser capturada para que a seção cancele a transação em andamento	STREQ007
PROGREQ012	Todo serviço importante prestado pelo caixa eletrônico (saque, transferência, depósito e consulta de saldo) é considerado uma transação e deve ser comunicado ao banco. As implementações destes serviços deverão verificar se a transação a ser realizada está à disposição do cliente.	STREQ008
PROGREQ013	Deverá ser implementado um Terminal de Operação que apresentará as funcionalidades de: liberação do cartão retido e consulta/impressão dos registros do sistema.	STREQ009
PROGREQ014	Uma opção para usar outro serviço deve estar disponível para o cliente após a conclusão ou cancelamento da transação em andamento.	STREQ010
PROGREQ015	Os serviços de depósito, transferência e saque devem imprimir um recibo para o cliente após a conclusão da transação. O recibo deve indicar o tipo de operação realizada, o dia, o valor da operação e as contas envolvidas.	STREQ011
PROGREQ016	As transações, o envelope, o terminal e os registos de erros devem ser implementados e acessíveis apenas através do terminal do operador.	STREQ012

Tabela 8 – Programas de sistema de software de caixas eletrônicos (ATM).

Programa	**Descrição**	**Requisitos de Programa**
PROG001	Conexão do sistema do banco	PROGREQ001
PROG002	Session de caixa eletrônico	PROGREQ002, PROGREQ014

Tabela 8 – Programas de sistema de software de caixas eletrônicos (ATM).

Programa	**Descrição**	**Requisitos de Programa**
PROG003	Saque	PROGREQ003, PROGREQ004, PROGREQ005
PROG004	Depósito	PROGREQ006
PROG005	Transferência	PROGREQ007, PROGREQ008
PROG006	Consulta de saldo	PROGREQ009, PROGREQ010
PROG007	Transação Geral	PROGREQ011,PROGREQ012
PROG008	Terminal do Operador	PROGREQ013
PROG009	Impressão de recibo	PROGREQ015
PROG010	Criação de Registro	PROGREQ016

Tabela 6 – Requisitos de Sistema para o Sistema de Caixa eletrônico (ATM).

Requisitos	**Descrição**	**Derivado de**
SYSREQ001	O caixa eletrônico deve ter uma conexão de Internet adequada para se comunicar com o sistema do banco.	STREQ01
SYSREQ002	A conexão com a Internet deve ser criptografada com criptografia AES o tempo todo.	STREQ01
SYSREQ003	O sistema deve ser implementado com tecnologias Java EE, a fim de manter a consistência com outros sistemas do banco.	STREQ01
SYSREQ004	O caixa eletrônico deve ter um leitor de cartão instalado. O driver do leitor de cartão deve ser nativamente compatível com o sistema operacional instalado na máquina do caixa eletrônico.	STREQ02
SYSREQ005	O caixa eletrônico deve ter um periférico dispensador de dinheiro instalado. O driver do dispensador de dinheiro deve ser nativamente compatível com o sistema operacional instalado na máquina do caixa eletrônico.	STREQ04
SYSREQ006	O caixa eletrônico deve ter um periférico para aceitar envelopes instalado. O driver periférico deve ser nativamente compatível com o sistema operacional instalado na máquina do caixa eletrônico.	STREQ05
SYSREQ07	O caixa eletrônico deve ter uma impressora instalada. O driver da impressora deve ser nativamente compatível com o sistema operacional instalado na máquina do caixa eletrônico.	STREQ12

Tabela 9 – Lista de defeitos no sistema de software de caixas eletrônicos e nos programas em que estão inseridos.

Defeito	**Descrição**	**Programa**
DEFECT001	A Tela de Saque não verifica duas vezes a quantidade de dinheiro a ser retirada pelo cliente.	PROG003
DEFECT002	A restrição que impede o cliente de imprimir mais de um recibo de consulta de saldo na mesma sessão bancária não está funcionando.	PROG006
DEFECT003	O recibo de transferência não está sendo impresso corretamente.	PROG009
DEFECT004	O caixa eletrônico está fechando a sessão após imprimir um recibo, sem perguntar à cliente se ela deseja realizar outra operação.	PROG002
DEFECT005	O caixa eletrônico não mantém um registro com informações dos cartões retidos.	PROG010