UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Marcio Tadeu de Oliveira Júnior

Recomendação de Desenvolvedores Externos para Projetos de Software Baseada na Análise de Contribuições Prévias

Juiz de Fora
2019

Marcio Tadeu de Oliveira Júnior

Recomendação de Desenvolvedores Externos para Projetos de Software Baseada na Análise de Contribuições Prévias

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profa. D.Sc. Regina Maria Maciel Braga Villela

Coorientador: Prof D.Sc. Gleiph Ghiotto Lima de Menezes

Juiz de Fora

2019

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

de Oliveira Júnior, Marcio Tadeu.
Recomendação de Desenvolvedores Externos para Projetos de Software Baseada na Análise de Contribuições Prévias / Marcio Tadeu de Oliveira Júnior. -- 2019.
100 p. : il.

Orientadora: Regina Maria Maciel Braga Villela
Coorientador: Gleiph Ghiotto Lima de Menezes
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós Graduação em Ciência da Computação, 2019.

1. Ecossistemas de Software. 2. Sistemas de Recomendação. 3. Expertise Retrieval. 4. Recomendação de Desenvolverdores. I. Villela, Regina Maria Maciel Braga, orient. II. de Menezes, Gleiph Ghiotto Lima, coorient. III. Título.

Márcio Tadeu de Oliveira Júnior

"Recomendação de Desenvolvedores Externos para Projetos de Software Baseada na Análise de Contribuições Prévias"

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre em Ciência da Computação.

Aprovada em 04 de dezembro de 2019.

BANCA EXAMINADORA

Profª. Dra. Regina Maria Maciel Braga Villela – Orientadora
Universidade Federal de Juiz de Fora

Prof. Dr. Gleiph Ghiotto Lima de Menezes – Coorientador
Universidade Federal de Juiz de Fora

Prof. Dr. José Maria Nazar David
Universidade Federal de Juiz de Fora

Profª. Dra. Gabriella Castro Barbosa Costa Dalpra
Centro Federal de Educação Tecnológica de Minas Gerais

Dedico este trabalho às mulheres da minha vida, por terem me apoiado em minhas decisões, e compartilhado as vitórias.

AGRADECIMENTOS

Agradeço à minha mãe Tânia e irmã Mariana, por todo apoio e compreensão ao longo desta jornada, pela amizade, amor, ensinamentos, momentos, mudanças, puxões de orelha, e tudo mais que o amor nos proporcionou.

Ao meu amor, Juliana, pelo amor, paciência e companheirismo.

À minha família, por compreenderem minhas ausências, dos momentos deliciosos de reunião, pelas velhas dores de cabeça e tudo aquilo que forma este recorte que chamamos de família. Em especial aos meus avós, primos, tios e meu pai.

À minha orientadora, Regina, por toda paciência, dedicação e zelo. Por me fazer acreditar em mim mesmo nos momentos em que me esqueci de fazê-lo. Pelo apoio que foi além do mestrado.

Ao meu coorientador, Gleiph, por todo *insight* que me tirou do estado de inércia em que me encontrei ao longo do meu mestrado. Agradeço pela renovação e retomada de meu trabalho.

Aos membros da banca, Gabriella e José Maria, pelas críticas que permitiram a melhora de meu trabalho, e por aceitarem participar nesta última etapa.

Aos professores do PGCC pelos seus ensinamentos dentro e fora das salas de aula.

Aos todos os meus amigos por compreenderem minha ausência, e compartilharem os bons momentos.

Aos amigos do PGCC: Phillipe, Iuri, Jade, Míria, Malú, Pedro Ivo, Lenita, Hugo, Lucas, Marcelo. Pelos bons momentos do #teamNEnC e da Banda Olar.

Aos amigos com os quais dividi meus dias: Vó Maria, Tio Du, João Marcos, Márcio, Breno, Esther e Mariana.

Àqueles que partiram sem testemunhar este momento em minha vida, em especial à Vó Carmita por sempre acreditar em mim, e ao meu amigo Betinho, o qual fazia da sua, minha segunda casa.

Às minhas meninas por sempre me aguardarem ansiosamente e me proporcionar a alegria de enfrentar os dias difíceis.

"*The chief proof of man's real greatness lies in his perception of his own smallness.*"

Sir Arthur Conan Doyle

RESUMO

A indústria de desenvolvimento de software evoluiu nos últimos anos e novos desafios surgiram. Dentre estas mudanças surgiram os ecossistemas de software, um novo paradigma de desenvolvimento, onde colaboradores externos apoiam a produção de software ao disponibilizar soluções que complementam uma plataforma comum a estes desenvolvedores. Devido à grande diversidade de tecnologias, *frameworks* e domínios que um ecossistema pode abrigar, a todo momento surgem colaboradores com variados tópicos de conhecimento e habilidades. Entretanto, recrutar colaboradores com as características desejadas se torna um trabalho complexo devido aos diferentes graus de conhecimento e habilidades que cada colaborador tem em suas diversas competências. Diante disso, apresenta-se uma arquitetura de um sistema de recomendação (SR) apoiado por uma ontologia capaz de recomendar colaboradores que tenham mostrado *expertise* nos tópicos de interesse. Para tanto, o SR utiliza técnicas da área de *expertise retrieval* para pontuar o grau de aderência dos colaboradores sobre os tópicos de conhecimento representados em uma *query*. A arquitetura é então capaz de fornecer as informações de contexto da recomendação, ou seja, uma visualização sobre onde pode-se encontrar os tópicos de conhecimento que levaram à recomendação de cada colaborador. Provas de conceito foram realizadas sobre dois ecossistemas de software para verificar a viabilidade da arquitetura, as quais mostraram indícios de que a arquitetura é capaz de realizar recomendações, e ainda oferece informações de contexto que são importantes à tomada de decisão sobre as recomendações realizadas.

Palavras-chave: Ecossistemas de Software, Sistema de Recomendação, Ontologia, *Expertise Retrieval*, e-*Science*, Colaboração.

ABSTRACT

The software development industry has evolved in the recent years and new challenges have emerged. Among these changes came Software Ecosystems, a new development paradigm, where external contributors support software production by providing solutions that complement a common platform for these developers. Due to the large number of technologies, frameworks and domains that an ecosystem can host, an equally large number of contributors acquainted with varied topics of their knowledge and skills have also emerged. However, recruiting collaborators with desired characteristics becomes a complex task due to the varying degrees of knowledge and skill that each developer has in their various competencies. Given this, we present a architecture of a recommendation system (RS) supported by an ontology capable of recommending collaborators who have shown expertise in the topics of interest. In order to do so, the RS uses retrieval expertise techniques to score the developers´ level of knowledge about topics represented in a query. The architecture is then able to provide the contextual information of the recommendation, i.e., a visualization of where one can find the knowledge topics that led to the recommendation of each contributor. Proof of Concepts were conducted on two software ecosystems to demonstrate feasibility of the architecture, which have shown evidence that the architecture is able to perform recommendations and still offers context information, important to the decision-making process over the recommendations made.

Keywords: Software Ecosystems, Recommendation System, Ontology, Expertise Retrieval, e-Science, Collaboration.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANTLR	ANorther Tool for Language Recognition
API	Application Programming Interface
DDS	Desenvolvimento Distribuído de Software
ECOS	Ecossistema de Software
ECOSs	Ecossistemas de Software
FLOSS	Free/Libre Open Source Software
HTTP	Hyper Text Transfer Protocol
JSON	JavaScript Object Notation
LPS	Linha de Produto de Software
NEnC	Núcleo de Engenharia do Conhecimento
OWL	Web Ontology Language
P2P	Peer-to-Peer
RDF	Resource Description Framework
SEON	Software Engineering Ontology Network
SPARQL	SPARQL Protocol and RDF Query Language
SR	Sistema de Recomendação
UFJF	Universidade Federal de Juiz de Fora
URI	Uniform Resource Identifier

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto relacionado ao problema abordado nesta dissertação, as motivações para seu estudo, os objetivos, bem como sua organização.

1.1 CONTEXTUALIZAÇÃO

A indústria de desenvolvimento de software tem evoluído nos últimos anos. Essa evolução trouxe novas maneiras de desenvolver software, desde a adoção de novos *frameworks* e linguagens até o surgimento de novos paradigmas de desenvolvimento. Essa evolução vem, em parte, pela necessidade de redução no custo e no tempo de desenvolvimento (BOSCH, 2009).

Neste sentido, novas abordagens de desenvolvimento de software foram surgindo ao longo dos anos, como destaque para Linhas de Produto de Software (LPS) (CLEMENTS; NORTHROP, 2002), Desenvolvimento Distribuído de Software (DDS) (AUDY; PRIKLADNICKI, 2007) e Ecossistemas de Software (ECOSs) (BOSCH, 2009; JANSEN; FINKELSTEIN; BRINKKEMPER, 2009).

Linhas de Produto de Software se baseiam em construir soluções de software a partir de uma plataforma central, que contém as funcionalidades básicas do domínio, e com o passar do tempo, funcionalidades específicas a aplicações do domínio são adicionadas a esta plataforma (CLEMENTS; NORTHROP, 2002). Já o paradigma de Desenvolvimento Distribuído de Software (DDS) é adequado quando, neste contexto, há a necessidade de se recorrer a desenvolvedores externos para atender todos os requisitos de desenvolvimento de determinado software, desta forma, recorre-se a acordos com desenvolvedores externos que sejam capazes de desenvolver tais funcionalidades através de algum modelo de colaboração (e.g. colaboração *open-source, outsourcing*) (AUDY; PRIKLADNICKI, 2007). Ecossistemas de Software, por sua vez, englobam a ideia de separação de uma arquitetura principal, comum a LPS, da alta demanda por funcionalidades que levam à abertura da plataforma para que desenvolvedores externos contribuam com novas funcionalidades, sendo o desenvolvimento por membros externos, uma característica comum a DDS (AUDY; PRIKLADNICKI, 2007).

Tanto LPS, quanto DDS e ECOSs, são paradigmas que buscam reduzir os custos e tempo de desenvolvimento de software através de fatores como o reuso de artefatos desenvolvidos por terceiros, distribuição da carga de trabalho, e compartilhamento de conhecimento, entre outros. Neste contexto, surge uma maior necessidade de se encontrar

candidatos capacitados para serem introduzidos aos projetos que estão sendo desenvolvidos, de forma a colaborarem e manterem a continuidades destes.

Neste sentido, foi identificada a preocupação da comunidade em apoiar a colaboração no desenvolvimento de software no contexto de um ECOS (TEIXEIRA, ROBLES e GONZÁLEZ-BARAHONA, 2015). A partir de um mapeamento da literatura, foram identificados poucos trabalhos que focam na colaboração entre desenvolvedores a nível individual em um ECOS, tendo a maior parte da literatura focado em governança de ecossistemas, enquanto a inclusão e aderência dos indivíduos a projetos em um ECOS tem sido descrita superficialmente.

1.2 MOTIVAÇÃO

Considerando-se um ECOS como a evolução de LPS e DDS (BOSCH, 2009), podemos citar algumas vantagens que podem ser observadas ao se estabelecer um ecossistema ao redor de uma plataforma de software: (i) aumento do valor das funcionalidades oferecidas (ii) aumento na atratividade a novos usuários (iii) aumento da aderência à plataforma (iv) aceleração da inovação através de coinovação (v) colaboração com parceiros para dividir os custos de inovação (vi) união das soluções desenvolvidas por parceiros à plataforma (vii) redução do custo de aquisição e manutenção de funcionalidades desenvolvidas por parceiros (BOSCH, 2009). Ao se observar estas vantagens, verifica-se que um ECOS se torna cada vez mais atrativo quando novos colaboradores passam a fazer parte do ecossistema, melhorando ainda mais a coinovação e, consequentemente, a disponibilidade de soluções de software.

Assim, a aderência de novos desenvolvedores no ecossistema gera um ciclo virtuoso. Portanto, é importante promover a entrada de desenvolvedores em projetos de software relacionados ao ECOS, levando, idealmente, a um aumento da sua longevidade e saúde. No entanto, a literatura relata que existe uma tendência dos desenvolvedores a somente colaborarem em projetos nos quais eles já fazem parte, existindo, no entanto, uma menor parcela de desenvolvedores que colabora em outros projetos que não o seu próprio (SYED; JANSEN, 2013). Estes últimos, assim que começam a colaborar em projetos externos, tendem a criar ciclos virtuosos de colaboração (TEIXEIRA, ROBLES e GONZÁLEZ-BARAHONA, 2015), que levam mais desenvolvedores a colaborar em mais projetos. Esta tendência pode ser vista principalmente, em projetos de software aberto, onde a colaboração nestes projetos gera status e projeção.

1.3 PROBLEMA

Considerando este novo contexto de desenvolvimento em ECOSs, observa-se que a literatura trata de forma subliminar à colaboração dos indivíduos. Isso ocorre uma vez que, trabalhos que apoiam a qualidade da colaboração (VALENÇA; ALVES, 2017), focam principalmente nas relações de poder entre o desenvolvedor da plataforma e colaboradores, considerando questões de governança do ecossistema (QIU; HANN; GOPAL, 2013; RICKMANN; WENZEL; FISCHBACH, 2014; SMEDLUND; FAGHANKHANI, 2015; WAREHAM; FOX; CANO GINER, 2013). Portanto, verifica-se que a colaboração dos indivíduos em ecossistemas de software é um tópico ainda a ser explorado, apesar de já existirem algumas iniciativas pontuais (SYED; JANSEN, 2013; TEIXEIRA; ROBLES; GONZÁLEZ-BARAHONA, 2015). Também é possível verificar que mesmo entre os poucos trabalhos preocupados em estabelecer a posição de indivíduos dentro de seus ECOSs, nenhum se preocupou em diretamente promover a colaboração dentro dos ECOSs através da busca de colaboradores em outros projetos.

1.4 METODOLOGIA

A partir da identificação do problema, esta dissertação busca colaborar na identificação de soluções que ajudem a melhorar a colaboração de desenvolvedores externos em novos projetos, considerando principalmente o contexto de um ECOS. Para isso, este trabalho segue as diretrizes de *Design Science* (HEVNER *et al.*, 2008), apresentando uma arquitetura para recomendação de colaboradores externos para projetos de software, denominada Collab-RS (criação de artefatos), com o objetivo de automatizar a recomendação através do uso de modelos ontológicos e técnicas de mineração de repositórios, auxiliando a atividade de recrutamento de colaboradores (domínio do problema). Um mapeamento sistemático foi realizado para verificar o atual cenário da literatura sobre o problema abordado. Dois estudos foram realizados para se verificar a viabilidade da solução, um inicial que serviu para verificar a viabilidade do protótipo em um contexto relacionado a DDS, e um segundo no contexto de um ECOS científico, para verificar a extensibilidade da solução.

1.5 QUESTÃO DE PESQUISA

A seguinte questão de pesquisa é investigada neste trabalho: *Como a arquitetura Collab-RS pode recomendar candidatos externos para projetos de software relacionados a um ECOS?*

1.6 OBJETIVO

O objetivo geral desta dissertação é propor a arquitetura Collab-RS, sendo ela capaz de apoiar a colaboração em um ECOS a partir de técnicas de recomendação apoiadas por ontologia.

Como objetivos específicos podemos citar:

I. Estudo da literatura para verificar o estado da arte considerando a colaboração de indivíduos em projetos de desenvolvimento de software externos ao seu contexto;
II. Desenvolvimento de uma ontologia para a representação de colaboração de indivíduos em um ECOS;
III. Desenvolvimento de uma arquitetura para a recomendação de colaboradores em ECOSs;
IV. Condução de um estudo de viabilidade relacionado ao uso da arquitetura no contexto do ECOS Node.js.
V. Condução de Prova de Conceito para verificar a viabilidade de uso da arquitetura no contexto do ECOS E-SECO;

1.7 ORGANIZAÇÃO

Este trabalho foi dividido em cinco capítulos. O Capítulo 2 apresenta os pressupostos teóricos e o resultado de um mapeamento sistemático da literatura, que identificou lacunas a serem investigadas por essa dissertação. O Capítulo 3 descreve a solução proposta, que inclui a especificação da arquitetura Collab-RS, bem como a ontologia SECOn, desenvolvida como parte da solução, além de apresentar um estudo de viabilidade sobre os artefatos gerados, considerando o ECOS Node.js. O Capítulo 4 detalha uma Prova de Conceito, considerando o uso da arquitetura Collab-RS no contexto do ECOS E-SECO. O Capítulo 5 destaca as

conclusões deste trabalho, junto às contribuições da pesquisa, suas limitações e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta alguns conceitos necessários para o entendimento e desenvolvimento deste trabalho. Serão introduzidos os conceitos de Ecossistemas de Software, Sistemas de Recomendação e *Expertise Retrieval*. Além disso, foi realizado um mapeamento sistemático para verificar como os relacionamentos que levam à colaboração são estudados e apoiados. O mapeamento ajudou na identificação dos conceitos fundamentais detalhados na ontologia SECOn, apresentada no Capítulo 3, bem como a identificação de como a colaboração entre desenvolvedores vem sendo tratada.

2.1 ECOSSISTEMAS DE SOFTWARE

O Desenvolvimento Distribuído de Software (DDS) é um fenômeno que mudou os paradigmas de desenvolvimento de software ao expandi-lo além das barreiras das empresas, tornando-se um modelo amplamente adotado (AUDY; PRIKLADNICKI, 2007).

Através da adoção das ideias de DDS, empresas passaram a deixar parte da implementação de seus produtos a terceiros por diversos fatores, entre eles: especialidade de terceiros, custo/benefício de treinamento desvantajoso, falta de pessoal capacitado, especialistas trabalhando em outros projetos, entre outros.

Neste contexto, surge o conceito de Ecossistema de Software, que pode ser visto como uma evolução às Linhas de Produto de Software (LPS), onde estas extrapolam as fronteiras das empresas e são tratadas no contexto de desenvolvimento distribuído, aberto ou intermediado por parcerias (BOSCH, 2009).

Em um ECOS, desenvolvedores externos são atraídos para trabalhar em uma plataforma devido ao valor oferecido por esta, diferenciando-se de DDS tradicional, onde empresas externas são chamadas a trabalhar com o produto sendo desenvolvido através de contratos de *outsourcing*, ou seja, em DDS espera-se mais que desenvolvedores externos trabalhem em projetos por serem contratados, enquanto em ECOSs espera-se que os desenvolvedores sejam atraídos pelas vantagens de se desenvolver complementos para o projeto que está sendo conduzido (BOSCH, 2009).

Os primeiros relatos de ECOSs foram apresentados por MESSERSCHMITT; SZYPERSKI (2003), mas a primeira definição apareceu na evolução do trabalho do mesmo autor (MESSERSCHMITT; SZYPERSKI, 2005), como: “Tradicionalmente, um ecossistema

de software se refere a uma coleção de produtos de software que têm algum grau de relacionamento simbiótico.”

Já JANSEN, FINKELSTEIN e BRINKKEMPER (2009) definem ecossistemas de software com uma visão mais centrada nos negócios: “[…] um conjunto de empresas funcionando como uma unidade e interagindo com um mercado compartilhado de software e serviços, juntamente com as relações entre eles. Esses relacionamentos são frequentemente subjugados por uma plataforma tecnológica comum ou pelo mercado e operam através da troca de informações, recursos e artefatos.”.

BOSCH (2009) apresentou uma outra definição para ECOSs, com uma visão mais centrada na plataforma: “Um ecossistema de software consiste no conjunto de soluções de software que permitem, suportam e automatizam as atividades e transações dos atores no ecossistema social ou comercial associado e nas organizações que fornecem essas soluções.”.

MANIKAS e HANSEN (2013), realizando uma revisão sistemática da literatura sobre ECOSs, compilam essas e outras definições, e como resultado, os autores apresentam uma nova definição: “[...] a interação de um conjunto de atores sobre uma plataforma tecnológica comum que resulta em várias soluções ou serviços de software. Cada ator é motivado por um conjunto de interesses ou modelos de negócios e conectado ao resto dos atores e ao ecossistema como um todo com relações simbióticas, enquanto a plataforma tecnológica é estruturada de forma a permitir o envolvimento e a contribuição dos diferentes atores.”.

Esta última definição abrange as três definições iniciais, expondo a presença das interações simbióticas entre os atores e a presença de uma plataforma comum. Uma definição diferente é dada por LUNGU, ROBBES e LANZA (2010), observando ECOSs a partir de uma perspectiva de projetos interativos: “Um ecossistema de software é uma coleção de projetos de software que são desenvolvidos e evoluem juntos no mesmo ambiente”.

De forma detalhada e mais técnica, ECOSs podem também ser vistos como uma evolução natural de LPS, cujos produtos tradicionais tiveram uma demanda significativa por novos recursos e personalização por parte dos usuários, ao ponto de os desenvolvedores internos (ou desenvolvedores *keystone*) não conseguirem lidar sozinhos com o desenvolvimento destes recursos e personalizações devido a limitações de pesquisa e desenvolvimento de tecnologias (BOSCH, 2009). As LPS contam com uma arquitetura geral da solução de software e diversos componentes modularizados. Estes componentes oferecem ao software criado através de uma LPS maior variabilidade e, portanto, maior grau de personalização do software (CLEMENTS; NORTHROP, 2002). A evolução ocorre quando a arquitetura se torna aberta, assim permitindo que desenvolvedores externos possam colaborar

através da criação destes componentes modularizados. Assim, aumenta-se a disponibilidade de soluções personalizadas para tal arquitetura, agora chamada plataforma, no contexto de ECOSs.

Neste trabalho, usamos a definição de MANIKAS e HANSEN (2013), apresentada anteriormente.

Levando em consideração as definições anteriores, os ecossistemas de software são um conjunto de soluções e serviços que compartilham um mesmo mercado, recursos e outras soluções de software, e de atores que compartilham e aproveitam informações na rede social formada pelos mesmos. Portanto, ECOSs são ambientes que permitem uma maior gama de soluções de software que atendam aos usuários. Além das vantagens de atender às demandas dos usuários, outras vantagens podem ser obtidas estabelecendo um ECOS em torno de uma plataforma de software (BOSCH, 2009):

- Maior valor dos recursos oferecidos;
- Maior atratividade para novos usuários;
- Maior aderência à plataforma;
- Acelerar a inovação através da coinovação;
- Colaboração com parceiros para compartilhar os custos de inovação;
- Oferecer soluções criadas por desenvolvedores externos para a plataforma; e
- Oferecer custo reduzido de aquisição e manutenção de recursos desenvolvidos por colaboradores externos.

Um ecossistema de software pode ser observado de diferentes perspectivas. CAMPBELL e AHMED, (2010) estudam ECOSs a partir de uma perspectiva tridimensional, considerando as seguintes dimensões:

- Dimensão arquitetural que foca na plataforma e sua arquitetura. Esta dimensão é a que se preocupa com modelos arquiteturais, modelos de variabilidade, interoperabilidade, processo de engenharia de domínio e outros tópicos relacionados à arquitetura da plataforma.
- Dimensão transacional, ou dimensão de negócios, se concentra nas transações entre os atores por meio de negócios, projetos, planejamento estratégico, aquisição de soluções e outros tópicos relacionados a transações entre os atores.
- Dimensão social que destaca as relações sociais entre atores, promoção da plataforma e atores, compartilhamento e aquisição de conhecimento.

O conhecimento destas três dimensões é importante para se compreender as diversas relações e transações que ocorrem dentro de um ECOS. No entanto, SANTOS e WERNER, (2012) propõem ainda, uma nova dimensão, a dimensão de Engenharia e gerenciamento de ECOSs, a qual mescla as outras três dimensões através das relações entre estas. Esta dimensão se concentra no desenvolvimento, evolução, estabelecimento e valor da plataforma, os quais são observados através das interações que ocorrem entre as outras três dimensões.

As diferentes interações observadas em todas as dimensões formam as redes sociotécnicas, nas quais artefatos, componentes, serviços, modelos, documentos e atores interagem entre si (DOS SANTOS *et al.*, 2014). As redes sociotécnicas reúnem então uma grande gama de informação sobre os ECOSs e podem ser vistas como fonte de entendimento sobre um dado ecossistema quando analisadas.

2.1.1 Redes Sociotécnicas

Redes sociotécnicas relacionam aspectos arquiteturais, sociais e de negócios entre desenvolvedores e software (DOS SANTOS *et al.*, 2014). Elas são formadas pelos relacionamentos dentro de ECOSs, incluindo compartilhamento de conhecimento, promoção de soluções de software e habilidades de atores, entre outros. O conceito de rede sociotécnica vem da fusão dos conceitos de redes sociais e redes técnicas (CAMPBELL; AHMED, 2010).

Em uma rede social, muitas informações podem ser trocadas, aumentando o conhecimento médio no ecossistema, agregando assim valor ao mesmo. Um ECOS com uma rede social ativa e colaborativa pode ser mais atraente para novos desenvolvedores e usuários, já que eles podem ter maior chance de encontrar suporte. Portanto, as redes sociais no contexto de ECOSs são importantes para manutenção e atração de atores para a comunidade do ecossistema, além de aumentar o conhecimento compartilhado sobre a plataforma e as soluções que as cercam.

As redes técnicas, por outro lado, estão preocupadas com as relações comerciais e técnicas que envolvem soluções de software em ECOSs (CAMPBELL; AHMED, 2010). Elas geralmente são definidas pelas relações transacionais, onde desenvolvedores fornecem soluções a seus usuários, sejam estes usuários finais ou outros desenvolvedores. Portanto, as redes técnicas são importantes para ECOSs pelo seu papel fundamental na definição das relações simbióticas entre os atores, permitindo troca e reutilização das soluções.

Ao compreender a importância de ambas as redes é possível compreender melhor o papel essencial das redes sociotécnicas em manter os atores em um ambiente simbiótico onde a coinovação, colaboração e até competição sustentam o equilíbrio do ecossistema.

Um mapeamento sistemático é conduzido na Seção 2.4 com a intenção de se reconhecer quais são os pontos importantes das redes sociotécnicas a fim de organizá-los para melhor modelarmos tais redes através de modelos semânticos, como ontologias (GUARINO; OBERLE; STAAB, 2009), e delas extrair informações sobre quais tópicos são de conhecimento de cada desenvolvedor. Para se reconhecer o grau de *expertise* atribuído aos desenvolvedores em relação aos tópicos de conhecimento destes, presentes nas redes sociotécnicas, podem ser utilizados sistemas de recomendação, capazes de calcular esse expertise baseado no conteúdo produzido pelos desenvolvedores na rede sóciotécnica.. Neste sentido, detalhamos a seguir alguns destes conceitos importantes a este trabalho.

2.2 SISTEMAS DE RECOMENDAÇÃO

Sistemas de recomendação são softwares capazes de ranquear objetos de acordo com uma função de utilidade. Uma função de utilidade é o grau que indica a capacidade um item em satisfazer uma necessidade. Portanto, quanto maior a função de utilidade para um objeto, maior o grau de recomendação deste objeto (ADOMAVICIUS; TUZHILIN, 2005).

As funções de utilidade são definidas de acordo com o contexto da recomendação, e principalmente pelo método de filtragem utilizado. Cinco métodos de filtragem são destacados na literatura: filtragem baseada em conteúdo, filtragem colaborativa, filtragem híbrida, filtragem demográfica e filtragem social (ADOMAVICIUS; TUZHILIN, 2005) (BOBADILLA *et al.*, 2013). Métodos de filtragem buscam separar itens que são recomendáveis aos usuários, cada um considerando o tipo de informação disponível. A seguir detalhamos as principais características de cada um dos métodos.

A filtragem baseada em conteúdo compara a similaridade entre objetos a serem recomendados com as características do alvo da recomendação (ADOMAVICIUS; TUZHILIN, 2005). Por exemplo, ao se recomendar um desenvolvedor para colaborar em um projeto, pode-se observar características do candidato que são similares em termos de utilidade ao projeto, como linguagens de programação exigidas pelo projeto e as linguagens de programação conhecidas pelos desenvolvedores. Este método é útil por ser simples, e cabível quando a recomendação pode se basear em características dos próprios objetos recomendados.

A filtragem colaborativa consulta as avaliações dadas aos objetos recomendados por usuários com preferências similares ao usuário alvo da recomendação (ADOMAVICIUS; TUZHILIN, 2005). Um exemplo disso, são os repositórios recomendados na seção de descoberta do GitHub (Seção 3.4), os quais são recomendados por terem maior quantidade de seguidores, onde seguir é a forma de demonstrar alguma preferência por um repositório.

A filtragem social se utiliza da afinidade de indivíduos em redes sociais para a recomendação. Os objetos são então recomendados baseados nas interações dentro de redes sociais realizadas pelo indivíduo alvo da recomendação (BOBADILLA *et al.*, 2013). Um exemplo são as recomendações de candidatos para se seguir no Github, onde são recomendados usuários que o sistema de recomendação considera fazer parte do mesmo círculo social do usuário alvo.

A filtragem demográfica considera um perfil demográfico traçado para cada indivíduo, baseado em alguma de suas características que o classifica como pertencente a um grupo, estas características são definidas de acordo com os critérios que são importantes à recomendação. Os grupos demográficos então são usados como os indivíduos das recomendações (BOBADILLA *et al.*, 2013). Como exemplo, temos as recomendações de repositórios que fazem parte de um mesmo ecossistema. Quando um indivíduo é classificado como pertencente a um ecossistema, repositórios do ecossistema passam a ser recomendados a este.

A filtragem híbrida, por sua vez, combina ao menos dois dos outros métodos de filtragem vistos anteriormente (BOBADILLA *et al.*, 2013).

Neste trabalho, desenvolve-se uma arquitetura que se utiliza de filtragem híbrida, baseada em filtragem social e filtragem baseada em conteúdo, que será detalhada na Seção 3.3.O uso da filtragem híbrida se justifica na utilidade oferecida pelo uso de modelos semânticos para representar a rede sociotécnica, criando-se um modelo de rede social a partir da qual as relações dos usuários podem ser utilizadas para se filtrar quais usuários farão parte da recomendação. Além disso, o uso de um modelo semântico pode também permitir o uso de informações a partir das quais pode-se extrair tópicos de conhecimento do candidato que podem ser utilizados para a filtragem baseada em conteúdo. A ausência da filtragem colaborativa se justifica pela ausência de sistemas de pontuação dos desenvolvedores, suas mensagens e código-fonte nos repositórios analisados. Enquanto a ausência da filtragem demográfica é justificada pela complexidade de se agrupar os desenvolvedores por tópicos de conhecimentos comuns, sem se usar da *query.* O que levaria o agrupamento em si a ser uma

forma de filtragem baseada em conteúdo. O uso de filtragem híbrida será apresentada no Capítulo 3 desta dissertação.

2.3 EXPERTISE RETRIEVAL

Expertise retrieval é uma área que tem como objetivo estimar o grau de associação entre pessoas e tópicos com os quais elas estão familiarizadas (BALOG *et al.*, 2012). Esta área tem como problema o fato de, diferente da recomendação de objetos, pessoas não podem ser decompostas em tópicos simples, portanto faz-se necessária a tarefa de se criar uma representação do conhecimento dos indivíduos baseando-se nos documentos aos quais estes estão associados. É importante ressaltar que o "documento", neste contexto, é qualquer artefato textualmente compreensível que carrega informações sobre os tópicos tratados.

Existem diversas abordagens em *expertise retrieval* para se encontrar *experts* em um conjunto de tópicos. Destas abordagens, podemos citar: modelos probabilísticos generativos, modelos probabilísticos discriminativos, modelos de votação e modelos baseados em grafos (BALOG *et al.*, 2012).

- Os modelos probabilísticos generativos ranqueiam candidatos de acordo com a probabilidade do candidato ser *expert* em uma *query*. Existem duas classes principais de modelos generativos: modelos de geração de candidatos e modelos de geração de tópicos.
- Os modelos probabilísticos descritivos têm maior enfoque nos documentos, utilizando diversas métricas sobre tais documentos para então se investigar as ligações entre autores e documentos. Estes métodos exigem um treinamento sobre os dados para mostrar bons resultados.
- Os modelos de votação se baseiam em ranquear os documentos de acordo com a *query* e estes documentos, por sua vez, são responsáveis por votarem em seus autores. Quanto maior a posição de um documento dentro do ranqueamento, mais peso ele dá a seus autores.
- Modelos baseados em grafos buscam pelos documentos de maior relevância para a *query* e a partir destes, cria uma rede de citações para identificar mais candidatos nas suas imediações.

A abordagem apresentada nesta dissertação faz uso dos modelos probabilísticos generativos. O uso se justifica pelo custo computacional reduzido em comparação aos demais e pela forma que recrutamentos ocorrem de fato. Os outros três modelos (probabilísticos

descritivos, votação, baseados em grafos) inicialmente analisam um conjunto limitado de documentos, e a partir deste conjunto, extraem-se os autores. Recrutamentos por sua vez se iniciam restringindo quais desenvolvedores podem ser selecionados, e estes então são filtrados baseados em suas contribuições (respectivamente filtragem social e filtragem baseada em conteúdo). A seguir apresentamos as classes e métodos de modelos probabilísticos generativos.

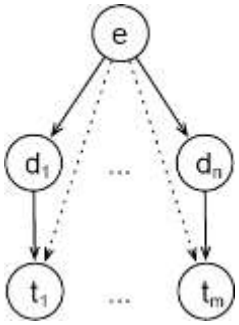
2.3.1 Modelos de geração de candidatos

A classe de modelos de geração de candidatos se baseia em pontuar a relevância dos documentos de acordo com uma *query* com os termos de interesse e, em seguida, ponderar este valor com um modelo de coocorrência, que é o grau de autoria do desenvolvedor sobre os documentos. Esta classe de modelos conta somente com o modelo de linguagem em dois estágios, proposto por CAO *et al (*2005). O método é centrado em documentos e se baseia no produto dos dois fatores definidos anteriormente: relevância do documento P($d|q$) e modelo de coocorrência P($e|d,q$). Por exemplo, um documento d tem uma relevância para a *query* q, e o candidato e fez algumas modificações neste documento, o modelo de coocorrência indica o quão influente foram as modificações de e sobre d, então o expertise de e sobre a *query* q é o produto da relevância do documento em relação à *query* e o grau de autoria do documento atribuído ao candidato.

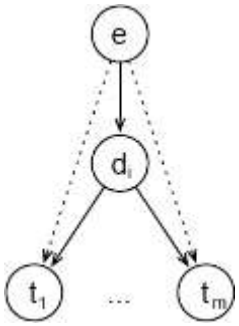
2.3.2 Modelos de geração de tópicos

Esta família de modelos se baseia em ranquear candidatos de acordo com a probabilidade do candidato e ser um *expert* dada uma *query* q, mapeando a relação entre candidatos e tópicos t da *query* q. Diferente dos modelos de geração de candidatos, que usam como base a relevância dos documentos e o grau de associação do candidato aos documentos, essa família de modelos leva em consideração também a importância do candidato e. Desta forma, é possível incorporar ao cálculo uma função que expresse que um autor tem maior conhecimento sobre os tópicos da *query* q, assim expressando conhecimento explícito do candidato e sobre os termos de q. Esta família se divide em três principais submodelos: modelos de candidatos, modelos de documentos e documentos como misturas de pessoas. Um diagrama representando os três modelos pode ser visto na Figura 1.

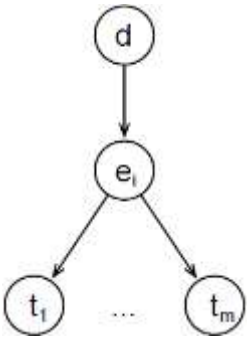
Figura 1 – Família de modelos de geração de tópicos onde as linhas tracejadas representam dependências condicionais entre os candidatos *e* e os tópicos *t*.



a: Modelo de candidatos



b: Modelo de documentos



c: Documentos como misturas de pessoas

2.3.2.1 Modelos de candidatos

Neste modelo, um candidato *e* é representado pelo conjunto de termos *t* da *query q*. Desta forma, para todo candidato analisado, observa-se todo documento *d* que está associado a este candidato, e analisa-se então o grau de *expertise* do candidato. O método utilizado neste trabalho é desta família, visto que ela inicia sua análise a partir de um conjunto de candidatos, buscando em seguida os documentos associados a estes. No contexto do trabalho desta dissertação, os candidatos são primeiramente considerados, para em seguida, analisar os projetos nos quais trabalharam.

2.3.2.2 Modelos de documentos

Este modelo se difere do modelo anterior, pois primeiramente é feito o ranqueamento dos documentos *d*, e a partir destes documentos, extraem-se os tópicos que são associados. Este modelo pode ser aplicado quando o conjunto de documentos é mais importante para a extração de candidatos do que uma lista inicial de candidatos. Um exemplo deste caso no contexto de desenvolvimento de software são os casos em que se busca desenvolvedores dentro de um projeto.

2.3.2.3 Documentos como mistura de pessoas

As famílias de modelos anteriores consideram que os tópicos *t* da *query q* são encontrados nos documentos *d*. Esta família de métodos considera que os termos são gerados pelos candidatos. Desta forma, os termos não são encontrados nos documentos, mas sim diretamente nos dados dos candidatos. Estes métodos são úteis quando é possível extrair parte da *expertise* de um candidato a partir de seus metadados e calcular a probabilidade destes tópicos estarem presentes nos documentos editados pelo candidato. Portanto, este modelo considera que coautores compartilham tópicos de conhecimento que já foram reconhecidos como tópicos de conhecimento de um dos autores. Portanto, quando se sabe que o candidato *e1* trabalhou com *e2*, já que ambos editaram um documento *d* juntos, considera-se que os tópicos de conhecimentos de *e1* e *e2* têm a probabilidade de estarem presentes no documento *d*. Então, se *e1* é um desenvolvedor que conhece o *framework react*[1] e trabalhou em um documento qualquer, há uma probabilidade deste documento se tratar de um documento contendo *react*. Portanto, se *e2* também trabalho em *d*, *e2* têm certa probabilidade de conhecer *react*.

2.4 MAPEAMENTO SISTEMÁTICO SOBRE REDES SOCIOTÉCNICAS

A partir do estudo do contexto de pesquisa relacionado a este trabalho, foi identificada a importância da investigação do uso de redes sociotécnicas. Assim, considerando DDS, a necessidade de colaboração entre os desenvolvedores e a importância que a colaboração assume no contexto do desenvolvimento de software atual, foi considerada importante uma

[1] https://reactjs.org/

investigação sobre o papel das redes sociotécnicas e seu uso para a melhoria da colaboração no desenvolvimento de software.

Para melhor compreensão sobre os conceitos relacionados às redes sociotécnicas de ECOSs foi conduzido um mapeamento sistemático sobre os relacionamentos nelas presentes, buscando principalmente descobrir as formas pelas quais as relações são representadas. Um mapeamento sistemático busca responder questões através de uma revisão da literatura, seguindo um protocolo de revisão que garante maior nível de reprodutibilidade dos resultados (KITCHENHAM, 2004). Para se realizar o mapeamento, as seguintes fases são propostas: planejamento, condução e análise dos dados.

2.4.1 Planejamento do mapeamento

Seguindo as diretrizes de KITCHENHAM (2004) para se definir um protocolo de mapeamento, primeiro o objetivo do mapeamento deve ser especificado, para que as questões de pesquisa possam ser delineadas. O objetivo deste mapeamento foi:

Analisar os tipos de relacionamentos em ecossistemas de software, **a fim de se compreender** as múltiplas visões sobre os relacionamentos **do ponto de vista** dos atores de ECOSs **em relação aos** seus papéis e interesses na rede do ECOS.

Para atingir esse objetivo é necessário definir as questões de mapeamento que direcionarão o estudo para esse objetivo. A Tabela 1 apresenta as questões de mapeamento para este trabalho.

Tabela 1 – Questões de Mapeamento

Questão de Mapeamento	Motivação
QM 1 - Quais são os tipos de relacionamentos presentes em ECOSs?	Identificar como os relacionamentos têm sido vistos pelos autores ao conduzir seus estudos de acordo com seus interesses e verificar quais são reconhecidos como forma de colaboração.
QM 2 - Quais são os tópicos estudados no aspecto sociotécnico dos ecossistemas de software?	Identificar desafios da área para se conhecer o que a literatura levantou como importante a ser tratado, observando-se sob qual ângulo os trabalhos são estudados. Como por exemplo, trabalhos que identifiquem desafios enfrentados do ponto de vista do dono/mantenedor da plataforma em relação à governança do ecossistema.
QM 3 – Como os relacionamentos foram modelados?	Identificar modelos que foram estudados e/ou criados anteriormente para se representar ECOSs.
QM 4 – Como a colaboração é estudada e apoiada?	Identificar quais fatores são estudados para melhorar a integração de colaboradores e como a colaboração tem disso apoiada no contexto de ECOSs.

A partir disso, uma tabela PICOC (WOHLIN *et al.*, 2012) foi especificada para identificar os termos que fariam parte da *string* de pesquisa, a Tabela 2 traz a definição PICOC.

Tabela 2 - Definição PICOC

Item PICOC	Valor
População (Population)	Redes sociotécnicas.
Intervenção (Intervention)	Relacionamentos nas redes sociotécnicas.
Comparação (Comparison)	Não se aplica.
Saída (Outcome)	Relacionamentos intra ECOSs
Contexto (Context)	Ecossistemas de software.

A *string* derivada da definição PICOC é dada por:

("developer" OR "development" OR "developers" OR "actor") AND ("collaboration" OR "community" OR "communities" OR "network" OR "networks" OR "social media") AND ("relationship" OR "relationships" OR "relation" OR "collaboration") AND ("software ecosystem" OR "software ecosystems")

A validação da *string* de busca foi realizada ao verificar o retorno de trabalhos escolhidos como de controle para este mapeamento, e os quais certamente deveriam fazer parte do conjunto de artigos recuperados e aceitos. São estes: (BOUCHARAS; JANSEN; BRINKKEMPER, 2009), (KUDE; DIBBERN; HEINZL, 2012) e (VAN ANGEREN; JANSEN; BRINKKEMPER, 2014). A *string* foi adaptada para cada base de acordo com a sintaxe de cada uma, a saber, **ACM Digital Library, El Compendex**, **IEEE Digital Library**, **ISI Web of Science**, **Science@Direct**, **Scopus** e **Wiley Online Library**.

Com a *string* de pesquisa definida, é necessário definir os critérios de inclusão e exclusão. Os critérios de inclusão escolhidos foram:

- A publicação não é excluída por nenhum critério de exclusão E
- A publicação estuda ou define relacionamentos entre desenvolvedores

Os critérios de exclusão são:

- A publicação não estar em inglês OU
- A publicação não possui *abstract* OU
- A publicação é um *short paper* OU
- A publicação é apenas um *abstract* OU

- A publicação não está disponível OU
- A publicação está em um serviço pago inacessível OU
- A publicação não é sobre ecossistemas de software

2.4.2 Condução do mapeamento

De acordo com as diretrizes definidas por KITCHENHAM (2004), o próximo passo é conduzir o mapeamento. Então, primeiro é definida uma estratégia de busca para estudos primários.

2.4.2.1 Estratégia de pesquisa e seleção de estudos primários

A pesquisa foi realizada a partir de pesquisa automática em bases de publicação e mecanismos de busca. As bases de publicação e os mecanismos de busca utilizados neste são apresentados na Tabela 3. Para dar suporte ao processo de mapeamento, o serviço Parsifal foi usado.

Tabela 3 - Bases de publicação e mecanismos de busca

Bases de publicação ou mecanismos de busca	Website
ACM Digital Library	http://portal.acm.org
El Compendex	http://www.engineeringvillage.com
IEEE Digital Library	http://ieeexplore.ieee.org
ISI Web of Science	http://www.isiknowledge.com
Science@Direct	http://www.sciencedirect.com
Scopus	http://www.scopus.com
Wiley Online Library	http://onlinelibrary.wiley.com

A *string* de pesquisa foi fornecida como entrada para essas bases e mecanismos. No total, foram encontradas 543 publicações, das quais 81 eram duplicatas entre bases e mecanismos de busca. Das 462 publicações restantes, 342 foram eliminadas pela leitura do título ou pela avaliação de alguns outros critérios de exclusão (linguagem, artigos curtos, não disponíveis, acesso pago). Das 120 publicações restantes, 40 foram inicialmente selecionadas por leitura do abstract. Com posterior leitura das publicações restantes, 14 foram eliminadas, restando um total de 26 publicações, as quais podem ser vistas no Anexo A.

2.4.2.2 Extração e Análise dos dados

Os dados extraídos das publicações foram utilizados para responder às questões de mapeamento e são apresentados a seguir. As publicações foram categorizadas e tabuladas de acordo com as questões de mapeamento.

QM 1 - Quais são os tipos de relacionamentos presentes em ECOSs?

Entre as publicações estudadas, muitos tipos de relacionamentos são considerados. A maioria das publicações considera as relações clássicas de complementar o software criado pelo desenvolvedor *keystone*. Outras publicações consideram novos tipos de relacionamento, como, por exemplo, o exercício do poder (por coerção, punição, entre outros.). Em seguida, detalhamos os tipos encontrados de relacionamentos.

- Parceria é o tipo de relacionamento que ocorre quando o desenvolvedor *keystone* e os complementadores da plataforma alinham seus objetivos, de modo que o *keystone* possa impulsionar a coinovação apoiando fortemente os complementadores. Os complementadores, por sua vez, podem desenvolver soluções de alto valor, uma vez que tem uma relação mais próxima com o desenvolvedor *keystone*. As parcerias são usadas para prender o parceiro ao ecossistema, tornando-o mais dependente da plataforma (SMEDLUND; FAGHANKHANI, 2015; VAN ANGEREN; JANSEN; BRINKKEMPER, 2014; WAREHAM; FOX; CANO GINER, 2013).
- Complementação é o relacionamento criado entre os desenvolvedores *keystone* e complementadores. Esse tipo de relacionamento não se preocupa em desenvolver valor direto para o ecossistema, mas em usar a plataforma como base para o desenvolvimento de software. Isso pode ser visto como um relacionamento mais genérico entre complementadores e desenvolvedores *keystone* (KUDE; DIBBERN, 2009; RICKMANN; WENZEL; FISCHBACH, 2014; VAN ANGEREN; ALVES; JANSEN, 2016; VAN ANGEREN; JANSEN; BRINKKEMPER, 2014).
- Influência por poder, estudada em VALENÇA e ALVES (2017), é composta por um conjunto de formas de poder que desenvolvedores têm e que podem usar para influenciar outros participantes do ECOS.

- Colaboração é a relação criada entre dois atores que compartilham o mesmo objetivo e fomentam a cooperação para alcançá-lo. Por meio de colaboração, os atores geram valor e conhecimento para eles e até mesmo para o ECOS, fazendo com que ele cresça (FRANCO-BEDOYA *et al.*, 2017; KILAMO *et al.*, 2012; KUDE; DIBBERN; HEINZL, 2012; SYED; JANSEN, 2013; TEIXEIRA; ROBLES; GONZÁLEZ-BARAHONA, 2015; VALENÇA; ALVES, 2017; VAN ANGEREN; BLIJLEVEN; JANSEN, 2011).
- Concorrência é a relação entre dois atores que têm um objetivo comum, mas priorizam sua participação de mercado, criando assim uma condição de competição de produtos. Promove inovação rápida e agrega valor ao ecossistema, fornecendo soluções de software mais completas (FERDMAN *et al.*, 2017; GALATEANU; AVASILCAI, 2016).
- Competição: a relação entre atores que compartilham o mesmo objetivo, priorizam sua participação de mercado, mas ao mesmo tempo fornecem soluções, inovação e/ou conhecimento para seus concorrentes, criando um ambiente que possibilita o desenvolvimento dos atores (FERDMAN *et al.*, 2017; GALATEANU; AVASILCAI, 2016).
- Fornecimento de valor, ou fornecimento de software, é o relacionamento criado entre os atores que compram/usam produtos ou serviços de outros atores. Caracterizado por transações entre atores e forma básica de receita no ECOS (BOUCHARAS; JANSEN; BRINKKEMPER, 2009; COUTINHO; SANTOS; BEZERRA, 2017; CROOYMANS; PRADHAN; JANSEN, 2015; VAN ANGEREN; BLIJLEVEN; JANSEN, 2011).
- Compartilhamento de conhecimento é um dos relacionamentos centrais nas dimensões sociais, onde os atores trocam conhecimento, agregando mais valor ao ecossistema (KUKKO; HELANDER, 2012; SEICHTER *et al.*, 2010; SYED; JANSEN, 2013; VAN DER SCHUUR; JANSEN; BRINKKEMPER, 2011).

QM 2 - Quais os tópicos estudados no aspecto sociotécnico de software e ecossistemas em relação aos desenvolvedores?

Esta questão tem como objetivo abordar os tópicos que foram estudados nas publicações anteriores, permitindo conhecer quais são as dificuldades e desafios da área. A seguir são apresentados os temas vistos como desafios:

- Barreiras e facilitadores: este tópico é abordado por trabalhos que promovem a compreensão de quais são as barreiras que impedem que os atores façam parte de um ecossistema e quais os facilitadores que ajudam na entrada de atores ao ecossistema. Ambos os tópicos são muito importantes para a saúde dos ECOSs, uma vez que um ECOS deve equilibrar entre permitir que os atores certos entrem no ecossistema e manter fora atores que possam ser prejudiciais ao ecossistema (GALATEANU; AVASILCAI, 2016; KOCH; KERSCHBAUM, 2014; KUDE; DIBBERN; HEINZL, 2012; SCACCHI *et al.*, 2006; VALENÇA; ALVES, 2017; VAN ANGEREN; ALVES; JANSEN, 2016).
- Governança de ECOSs: os estudos neste tópico estão preocupados com as implicações de desempenhar um papel central em um ecossistema de software, onde as organizações que detêm o controle das políticas do ecossistema têm de lidar com os múltiplos relacionamentos entre os atores; extensões do produto principal, as barreiras e facilitadores para entrar no ecossistema, planejamento estratégico da plataforma, fatores de saúde e outros aspectos importantes da manutenção do ECOS (QIU; HANN; GOPAL, 2013; RICKMANN; WENZEL; FISCHBACH, 2014; SMEDLUND; FAGHANKHANI, 2015; WAREHAM; FOX; CANO GINER, 2013).
- Modelagem de ECOSs: os estudos neste tópico preocupam-se em modelar o ECOS para melhor compreensão e raciocínio sobre a estrutura sociotécnica do ECOS. Este tópico é abordado em detalhe pela próxima questão de mapeamento.

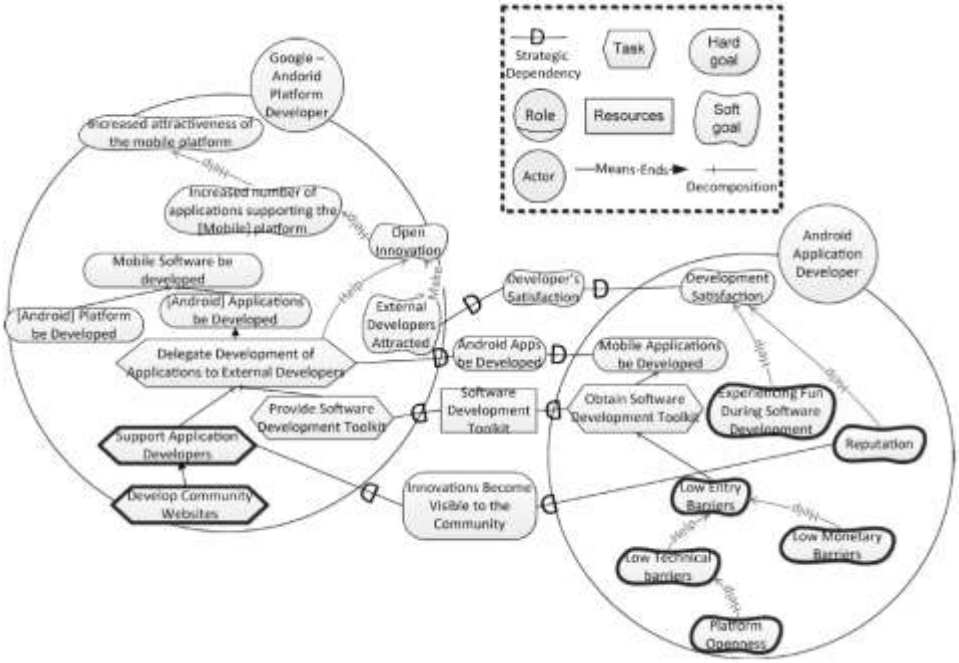
QM 3 - Como os relacionamentos encontrados foram modelados?

Quatro principais técnicas de modelagem foram encontradas na condução deste estudo: modelagem i*, modelagem por cadeias de valores, rede em grafo e *Ad-hoc*.

- Modelagem i*: SADI, DAI e YU (2015) propõem uma técnica de modelagem para redes sociotécnicas de ECOSs baseada em metas. Neste modelo, os fatores que levam os desenvolvedores a fazer parte de uma plataforma são o

foco do modelo. A técnica se baseia na representação dos objetivos que os atores têm ao entrar em um ecossistema, e as interdependências entre estes objetivos. Esta técnica busca apoiar a tomada de decisão de desenvolvedores que pretendem ou já ingressaram em um ECOS, para avaliar sua satisfação dentro do ecossistema. Verifica-se que esta técnica de modelagem permite interpretação apenas por humanos, visto que o conteúdo nela representado é livre. Um exemplo de modelo gerado pela técnica é apresentado na Figura 2, que exemplifica quais são os objetivos de um determinado desenvolvedor ao entrar no Ecossistema Android.

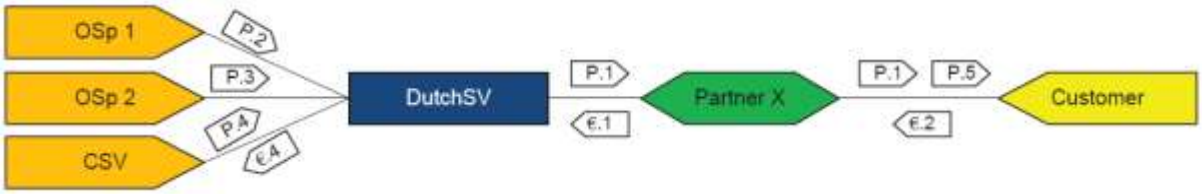
Figura 2 - Exemplo da modelagem pela técnica i* no ecossistema Android (SADI; DAI; YU, 2015)



- Modelagem de rede de fornecimento de software: Esta técnica de modelagem foi formalmente apresentada por BOUCHARAS, JANSEN e BRINKKEMPER (2009) no contexto de ECOSs. Ela formaliza um modelo para redes de fornecimento de software, o qual disponibiliza informações sobre as transações envolvendo software e os produtos destas transações entre os atores. O modelo foca em uma empresa de interesse e sua rede de fornecedores que são necessários para se criar o software de interesse, junto a intermediários que o revendem e os clientes finais do produto de interesse. Esta técnica de

modelagem busca permitir algum nível de tomada de decisão a partir do conhecimento das relações de dependência existentes entre empresas e produtos, e também permite algum nível de compreensão por máquina, visto que os relacionamentos são pré-definidos. Um modelo gerado por esta técnica pode ser visto na Figura 3, que exemplifica quais são os símbolos utilizados pela notação.

Figura 3 – Exemplo de rede de fornecimento de software (BOUCHARAS; JANSEN; BRINKKEMPER, 2009)



- Modelagem de redes em grafos: Redes em grafos são vistas em (SCACCHI *et al.*, 2006; VAN ANGEREN; ALVES; JANSEN, 2016; VAN ANGEREN; JANSEN; BRINKKEMPER, 2014), mas não são formalmente definidas por eles. Essa modelagem se baseia em teoria dos grafos e segue a definição própria de grafos, onde os vértices são as entidades representadas e arestas são as relações entre duas dessas entidades. Esta técnica de modelagem é muito versátil devido à flexibilidade de significado dos vértices e arestas. No entanto, ela permite apenas uma categoria de relacionamentos e uma categoria de entidades por grafo. Uma vantagem deste modelo é que é possível usar técnicas de análise de grafos sobre o modelo para inferir informações. Um exemplo de modelagem em grafo pode ser visto na
- Figura 4, que exemplifica uma rede de coautoria em projetos, onde os nós são desenvolvedores, e as arestas são projetos nos quais estes trabalharam em comum.
- Modelagem *ad-hoc*: A modelagem *ad-hoc* não é formalizada, permitindo assim qualquer forma de representação da rede. Depende fortemente do conhecimento do modelador sobre o domínio, ecossistema e relacionamentos. Também não há maneiras formais de representar a rede. Essa técnica de modelagem é usada quando explicações simples ou específicas da rede são de

interesse dos autores. Um exemplo pode ser visto na Figura 5, onde os autores definem as relações entre os atores em um ecossistema de software de gerenciamento de fazendas inteligentes (KRUIZE *et al.*, 2014).

Figura 4 - Exemplo de rede de relacionamentos modelada em grafo (SCACCHI *et al.*, 2006)

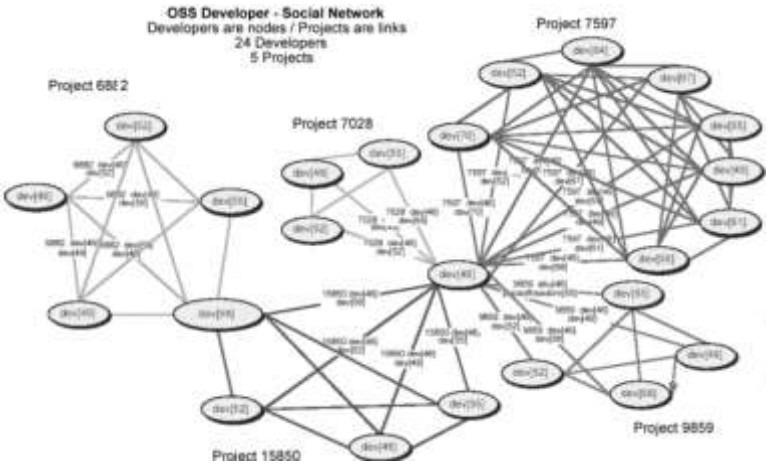
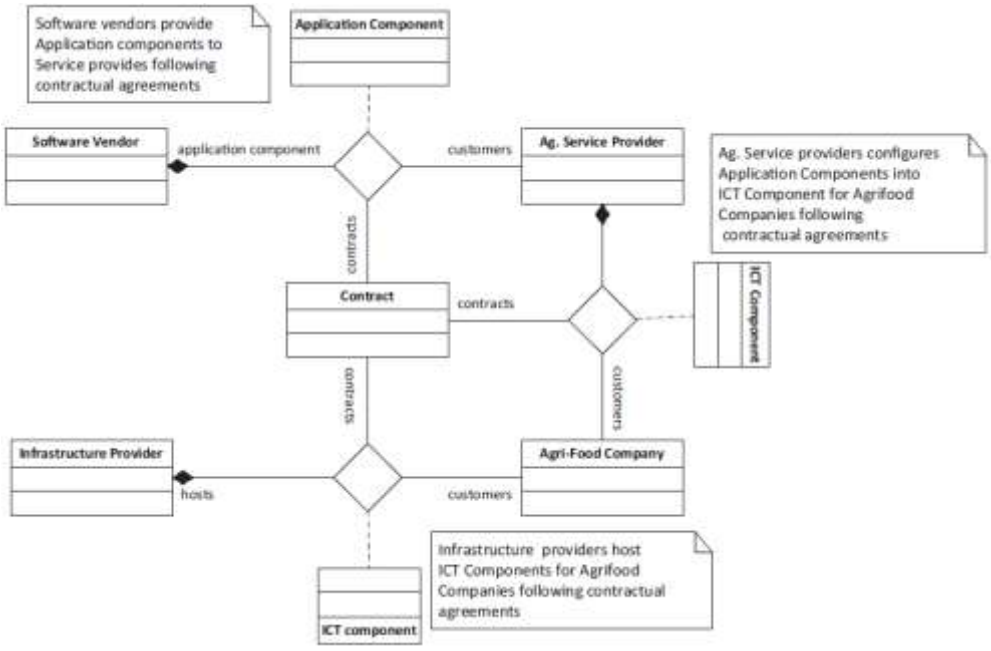


Figura 5 – Exemplo de modelo *ad-hoc* (KRUIZE *et al.*, 2014)



QM 4 – Como a colaboração é estudada e apoiada?

Apenas dois trabalhos verificaram as relações de colaboração em ECOSs a nível individual. TEIXEIRA, ROBLES e GONZÁLEZ-BARAHONA, (2015) verificaram que desenvolvedores em Ecossistemas de Software Aberto (FLOSS ECOSs) colaboram independentemente de trabalharem para empresas que competem ou não, desde que ambas

tenham interesse em um mesmo projeto. Também foi detectada uma tendência de a colaboração acontecer mais entre empregados de diferentes empresas do que da mesma em projetos FLOSS (*Free/Libre and Open Source Software*).

SYED e JANSEN, (2013), por sua vez, verificaram que desenvolvedores individuais tendem a ficar em isolamento quando desenvolvem para uma plataforma, mas ao iniciar a participação em projetos maiores tendem a criar relações de colaboração direta entre desenvolvedores, criando um ciclo virtuoso de colaboração. Também foi verificado que uma das principais razões de desenvolvedores fazerem parte de um ECOS é ajudar outras pessoas (depois de necessidades pessoais), indicando uma maior disposição a participação em projetos que beneficiam mais pessoas.

Os demais trabalhos que reconheceram colaboração como um relacionamento entre entidades de ECOSs não tratam diretamente de colaboração, mas sim apenas listam-na como uma das relações básicas de ECOSs.

2.4.3 Considerações sobre o mapeamento

Este mapeamento permitiu uma visão do estado da arte sobre redes sociotécnicas no contexto de ECOSs. Os resultados do mapeamento ajudaram a explicitar conceitos importantes sobre as entidades e relacionamentos existentes dentro de ECOSs. Estes conceitos são importantes para auxiliar na definição de um modelo semântico para redes sociotécnicas de ECOSs, pois trouxeram uma visão geral da literatura em relação aos principais conceitos e relacionamentos em um ECOS. Assim, de conhecimento dos conceitos e relações dentro de ECOSs como relações de troca de mensagens entre atores, transações de artefatos de software, trocas de mensagens, podemos através da representação das redes sociotécnicas verificar a existência destas relações dentro dos ECOSs.

Pudemos também observar no mapeamento a ausência de trabalhos que busquem reforçar a colaboração em ECOSs, enquanto pode-se verificar a existência de trabalhos que se preocupam em melhorar a qualidade da governança em ECOSs, sugerindo uma tendência da comunidade sobre a melhoria do ponto de vista dos desenvolvedores *keystone.*

Como resultado do mapeamento, consideramos que a especificação de um modelo semântico, que permita expressar conceitos e relacionamentos em um ECOS, como foco na colaboração, aliado a mecanismos computacionais de extração de conhecimento deste modelo, pode trazer algum ganho para fomentar o estudo de redes sociotécnicas em ECOSs.

Um modelo semântico neste contexto, de forma geral, é capaz de representar conceitos presentes nos modelos apresentados no mapeamento sistemático conduzido. Neste sentido,

acreditamos que o uso de ontologias (GUARINO; OBERLE; STAAB, 2009) pode ser adequado para a representação dos conceitos e relacionamentos sociotécnicos de um ECOS, sendo representada como um grafo por definição, de relações entre indivíduos ontológicos. Uma ontologia também pode ser estendida de forma a tentar compreender conceitos que seriam representados numa modelagem *ad-hoc*, representando novos conceitos de acordo com necessidades específicas do ECOS. No entanto, ainda que os mesmos conceitos sejam representados, há pouca vantagem em fazê-lo, visto que estas informações podem não ser compreensíveis por máquinas, ou possam não trazer conteúdo sobre o qual não se realize inferências. No Capítulo 3 é apresentada uma ontologia, denominada SECOn, que foi criada de forma a incluir a modelagem por redes de fornecimento de software através da relação de dependência entre softwares.

Comparando a ontologia SECOn, e os modelos gerados pela modelagem i*, esta segunda é mais genérica, sendo capaz de representar conceitos que não são passíveis de interpretação computacional. No entanto, o uso da ontologia SECOn permite a modelagem de conceitos representados computacionalmente e mais ainda, permite o processamento de conhecimento e relações implícitas no modelo, o que é uma caraterística relevante em modelos semânticos e importante para o fomento de redes sociotécnicas em ECOSs, uma vez que relações sociotécnicas não explícitas podem ser descobertas. Ademais, o uso da ontologia SECOn ainda permite a representação de conceitos que não são passíveis de interpretação computacional, como a própria representação das metas, característica à técnica i*, fazendo com que a esta seja um subconjunto da representação por ontologia, mas limitado à interpretação humana dos dados representados.

2.4.4 Ameaças à validade

Uma ameaça é a dificuldade em incluir todas as publicações relevantes na pesquisa. No entanto, essa ameaça foi parcialmente mitigada pela reformulação da *string* de busca, à medida que refinamentos nesta *string* de busca foram realizados. Outra ameaça à validade foi a não realização do processo de *snowballing*. Este processo poderia permitir a inclusão de novos trabalhos a partir das referências e citações dos trabalhos incluídos pela execução do protocolo de mapeamento sistemático. No entanto, o processo de *snowballing* não é considerado obrigatório, embora pudesse aumentar a precisão do mapeamento.

Podemos considerar também como ameaça à validade, o número de pesquisadores que participaram deste mapeamento, uma vez que o mesmo foi conduzido por somente um pesquisador.

2.5 MELHORIA DE COLABORAÇÃO EM OUTROS CONTEXTOS

Enquanto na área de ECOSs tenham sido encontrados apenas dois trabalhos diretamente relacionados ao apoio à colaboração em ECOSs, na área de Sistemas de Recomendação, por sua vez, encontramos trabalhos que buscam solucionar problemas relacionados à colaboração em outros contextos. Neste sentido, apesar do mapeamento sistemático ter tido como foco ECOSs, consideramos importante estudar e discutir alguns trabalhos que possam trazer contribuições para colaboração em ECOSs. Abaixo apresentamos alguns destes trabalhos no contexto de Desenvolvimento Distribuído de Software.

SUN *et al.* (2018) trazem uma abordagem de recomendação para resolver *issues requests*. Para cada *issue request*, verifica-se o provável conjunto de arquivos que possivelmente serão afetados pela verificação da similaridade entre o conteúdo da mensagem de *issue request* e o conteúdo das mensagens dos *commits* que modificaram cada arquivo. Se houver similaridade acima de um limite, o arquivo é considerado como um arquivo que possivelmente será modificado. Em seguida, verifica-se quais colaboradores estão familiarizados com cada arquivo, observando o grau de autoria pelos *commits*, caso o contribuinte tenha realizado *commit* do arquivo analisado, a probabilidade de o arquivo ser recomendado a ele aumenta. Por fim, os conjuntos de arquivos que podem ser modificados pela *issue request* são comparados ao conjunto de arquivos que cada autor está familiarizado. Os autores cujos conjuntos de arquivos têm a maior similaridade são recomendados. Este trabalho foca em recomendações dentro de um mesmo projeto, não promovendo colaboração inter-projetos, mas elucida-nos a importância do conteúdo de repositórios além do código-fonte como forma de verificar a aptidão dos colaboradores às tarefas a serem executadas.

AVELINO *et al.* (2018) têm uma abordagem para recomendar mantenedores de código, analisando métricas como: (i) o número de *commits* realizados por um colaborador, (ii) o número de linhas de código modificados por eles e (iii) o grau de autoria do contribuidor, o qual relaciona a quantidade de modificações que um colaborador fez em relação aos outros em um arquivo. O grau de recomendação é dado por uma função destes três valores normalizados. O autor verifica o quão recente o arquivo é, e o tamanho dele têm influência razoável no grau de recomendação. Os valores obtidos são comparados com um

oráculo, criado após uma pesquisa com desenvolvedores, para verificar a melhor abordagem para tratar a recomendação. Este trabalho nos mostra a importância do grau de autoria para se recomendar desenvolvedores, explicitando que uma relação entre o número de modificações sobre um determinado artefato pode levar a um grau maior de conhecimento sobre aquele artefato.

MONTANDON, SILVA e VALENTE 2019) apresentam formas de calcular o *expertise* de desenvolvedores em algumas dependências mais conhecidas do ecossistema Node. Para tal, coletam o valor de 13 aspectos da colaboração dos participantes (e.g. métricas como volume de *commits*, número de projetos onde o colaborador contribuiu, entre outros.) e criam dois métodos de clusterização (supervisionada e não-supervisionada) para o conjunto de dados obtidos. Ao final, cruzam os dados com *surveys* enviados aos colaboradores para verificar o grau de conhecimento deles sobre as dependências. Este trabalho nos ajuda a entender como desenvolvedores podem ser considerados candidatos a colaborarem em projetos, através do conhecimento sobre bibliotecas, mas não leva em consideração o fato se os desenvolvedores realmente têm conhecimento sobre as dependências.

2.6 CONSIDERAÇÕES

Neste capítulo, detalhamos os principais conceitos relacionados ao desenvolvimento da abordagem apresentada nesta dissertação, ressaltando a importância deste tema de pesquisa. Através de um mapeamento sistemático, verificou-se também a escassez de trabalhos que buscam apoiar a colaboração no contexto de ECOSs. Assim, pode-se verificar que há a necessidade de se apoiar a colaboração entre desenvolvedores individuais quando fazem parte de um ECOS.

Neste sentido, nos próximos capítulos, propõe-se uma abordagem que busca apoiar a colaboração através de técnicas de recomendação por *expertise*, de forma a introduzir novos colaboradores em projetos.

3 ARQUITETURA COLLAB-RS

Este capítulo apresenta a arquitetura Collab-RS e o desenvolvimento da ontologia SECOn. Ao final, é apresentado um estudo de viabilidade considerando os artefatos criados. Na seção 3.1 apresentamos a metodologia utilizada nessa pesquisa. Na seção 3.2 definimos a ontologia SECOn, fundamental para nossa abordagem. Na seção 3.3 definimos a Arquitetura Collab-RS, seus módulos, e como estes podem ser implementados. Na seção 3.4 apresentamos um estudo de viabilidade para verificar a viabilidade da proposta, junto às suas considerações.

3.1 DEFINIÇÃO METODOLÓGICA

Este trabalho considera as diretrizes de *Design Science* (HEVNER *et al.*, 2008) como metodologia de pesquisa apresentando uma arquitetura para recomendação de colaboradores externos para projetos de software (criação de artefatos), com o objetivo de automatizar a recomendação através do uso de ontologias e mineração de repositórios, auxiliando na atividade de recrutamento de colaboradores (domínio do problema). A Tabela 4 detalha a abordagem de acordo com os *guidelines* propostos por HEVNER *et al.* (2008).

Tabela 4 - Guidelines de acordo com HEVNER *et al.* (2008)

Guidelines	**Abordagem**
Desenvolvimento da abordagem como um artefato	Propõe-se uma arquitetura para apoiar a recomendação de colaboradores em projetos de software, considerando o uso de ontologias e sistemas de recomendação.
Relevância do Problema	A tarefa de recrutamento é discutida na literatura e está presente em sites relacionados a *social coding* (por exemplo, Github *Jobs* e *Jobs Stackoverflow*), mas a literatura não discute o recrutamento de colaboradores externos para projetos, enquanto estes sites demonstram uma necessidade real da indústria em relação a desenvolvedores externos.
Avaliação da Proposta	Para avaliar a viabilidade da proposta é realizado um estudo de viabilidade no contexto do ecossistema Node.JS[2] e uma Prova de Conceito relacionada ao ECOS E-SECO.
Contribuições da Pesquisa	Como contribuição, temos a especificação de uma arquitetura para recomendar colaboradores externos a projetos de desenvolvimento de software, usando ontologias e técnicas de recomendação através de informações implícitas extraídas de repositórios de software. As

[2] https://nodejs.org

	recomendações podem ser personalizadas com base na necessidade do recrutador.
Rigor da Pesquisa	Este trabalho detalha as etapas usadas para desenvolver a abordagem, como a ontologia e a arquitetura, de acordo com a metodologia *Design-Science*, bem como uma implementação da arquitetura no contexto do ecossistema Node.JS, a qual é utilizada no estudo de viabilidade e seu uso no ECOS E-SECO, a partir de uma Prova de Conceito.
Processo de Busca	Um mapeamento sistemático foi previamente realizado para conhecer o estado da arte da área, verificando como a recomendação de candidatos para projetos foi abordada pela literatura.
Comunicação da pesquisa	O modelo ontológico e o protótipo da arquitetura podem ser encontrados em https://github.com/marciotojr/Collab-RS

3.2 ONTOLOGIA SECOn

Para que fosse possível realizar a avaliação (provas de conceito sobre os ECOSs Node.js e E-SECO) da arquitetura Collab-RS (avaliação da proposta) foi proposta a ontologia SECOn. Esta ontologia tem como objetivo modelar ecossistemas de software a partir da visão tridimensional proposta por CAMPBELL e AHMED (2010), a saber:

- Dimensão arquitetural: relacionada ao desenvolvimento da plataforma. Trata o modelo arquitetural, modelos de variabilidade, interoperabilidade, processo de engenharia de domínio e outros tópicos relacionados à arquitetura da plataforma.
- Dimensão transacional: tem como foco as transações entre atores através de negócios, projetos, planejamento estratégico, aquisição de soluções e outros tópicos relacionados a transações entre atores.
- Dimensão social: centrada nas relações entre atores, promoção, compartilhamento e aquisição de conhecimento e outros temas relacionados aos relacionamentos de atores existentes no ecossistema.

O objetivo da ontologia SECOn é definir os principais conceitos de ecossistemas de software e a forma com as quais eles se relacionam e, com base em algoritmos de inferência, descobrir novas conexões entre entidades.

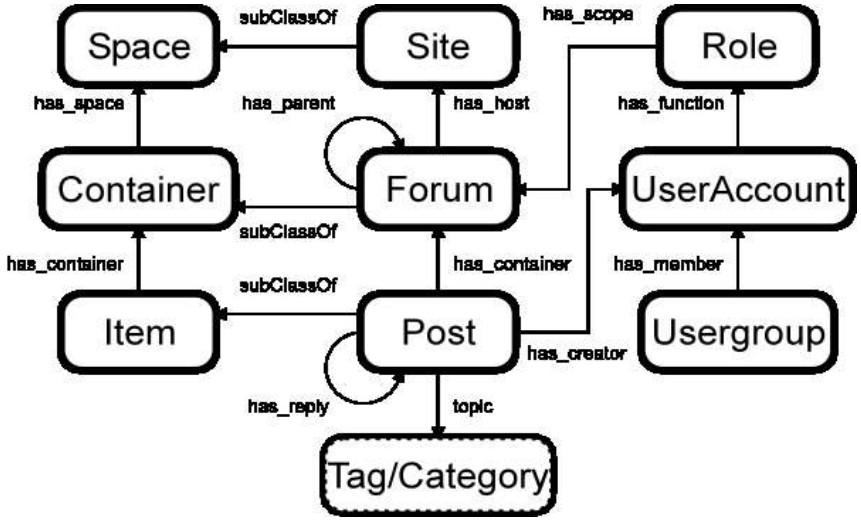
A utilização de ontologias busca promover uma representação das redes sociotécnicas, ao mesmo tempo que provê uma fonte de dados computacionalmente compreensível que é possível de ser utilizada para análise das informações da rede. Da mesma forma, a ontologia,

apoiada por um *reasoner* possibilita a inferência de informações que também podem ser processadas pelo meio computacional. Estas inferências ocorrem de forma automatizada e poupam a análise extensiva por um humano.

Como primeira etapa para o desenvolvimento da ontologia foi realizada uma busca na literatura com o objetivo de verificar a existência de ontologias relacionadas ao tema de pesquisa. A partir dessa busca, três ontologias foram reutilizadas, a saber, a ontologia SIOC (BRESLIN *et al.*, 2006), e duas ontologias da família SEON: SEON Issue Tracking Ontology e SEON History Ontology (WÜRSCH *et al.*, 2012). Também foram utilizados conceitos básicos de redes de fornecimento de software (BOUCHARAS; JANSEN; BRINKKEMPER, 2009). Apresentamos a seguir, brevemente, cada uma delas.

A ontologia SIOC é uma tentativa de descrever os principais conceitos e propriedades de comunidades *online*. A função de reutilizar conceitos desta ontologia no contexto da ontologia SECOn é descrever as interações sociais que ocorrem dentro de comunidades *online*, seja qual for sua natureza (BRESLIN *et al.*, 2006). Esta ontologia (Figura 6) organiza itens em *containers* e *containers* em espaços. Itens são a forma mais básica de comunicação, de acordo com a ontologia. Estes incluem formas atômicas de informação trocada entre usuários (*e.g. post*, mensagem de *e-mail*, resposta a outro item). *Containers* são agregações de itens ou de outros *containers* que correspondem a um mesmo contexto, compondo uma linha de mensagens correlatas (*e.g. threads* de discussão, listas de *e-mail* e fóruns). Espaços são ambientes virtuais que agregam containers, normalmente sob uma mesma plataforma (*e.g.* sites de redes sociais e serviços mensageiros instantâneos).

Figura 6 - Ontologia SIOC (BRESLIN *et al.*, 2006)



A ontologia define também que as interações de seus usuários se dão através do compartilhamento de itens e respostas a itens em um mesmo espaço criando o conceito de conta de usuário, que pode ter um papel e pertencer a grupos de usuários. A conta de usuário representa uma pessoa, física ou não, que tem a capacidade de compartilhar informação em comunidades *online*. Este usuário pode exercer diferentes papéis em diferentes fóruns (*e.g.* administrador, moderador e observador) a depender da organização de cada fórum. Também é possível que este usuário participe de diferentes grupos de usuários.

As ontologias da família SEON (WÜRSCH *et al.*, 2012) são uma tentativa de descrever formalmente os conceitos dos domínios de evolução e de repositórios de software. Os conceitos representados nestas ontologias estão relacionados à modelagem das interações de troca de conhecimento dentro do contexto de evolução de software, por exemplo, as contribuições aos projetos através de *commits*. Os principais conceitos das ontologias SEON podem ser vistos na Figura 7. A ontologia SEON *Issue Tracking* apresenta os conceitos mais comumente relacionados ao monitoramento de *issues* de software. Entre seus diversos conceitos, podemos destacar *Comment* e *Stakeholder*. *Stakeholders* representam indivíduos que têm interesse na resolução de *issues*, sendo um desenvolvedor ou não. *Comment*, por sua vez, representa comentários sobre uma *issue*. A ontologia SEON *History* apresenta conceitos relacionados às atividades de controle de versão de arquivos. Aqui destacamos os conceitos de *Commit* e *Commiter*. *Commit* representa uma mudança e está atrelado a um conjunto de arquivos, enquanto *Commiter* representa o indivíduo responsável por realizar um *commit*. Isto permite descobrir quais desenvolvedores estão relacionados a quais artefatos.

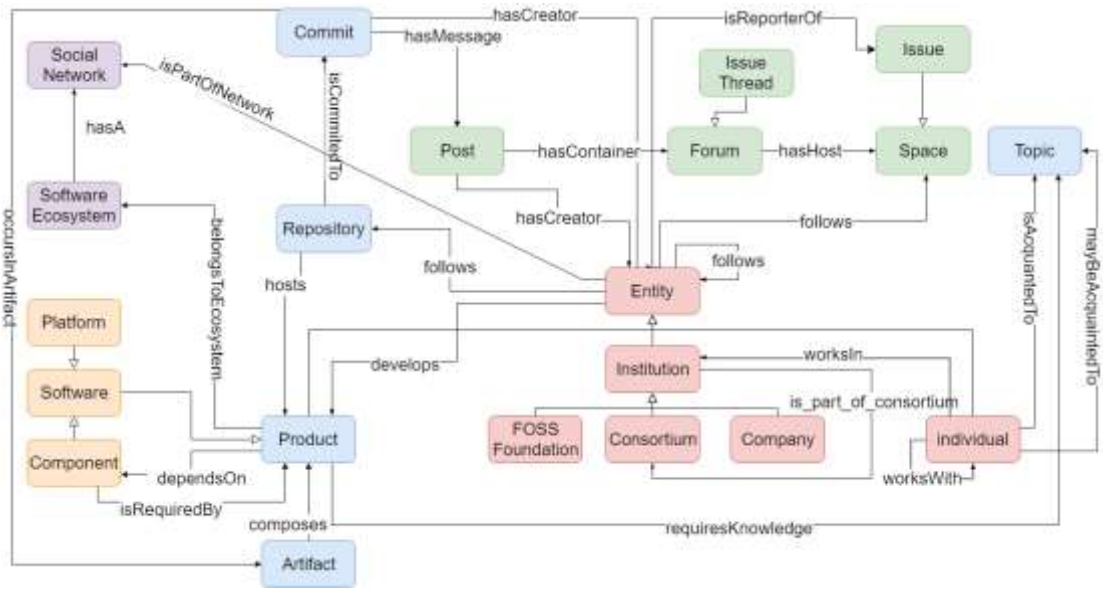
Figura 7 - Ontologias SEON Issue Tracking e SEON History[3]

[3] http://se-on.org/

A ontologia SECOn também engloba, além dos conceitos apresentados nas ontologias reutilizadas, conceitos relacionados às redes de fornecimento de software. Uma rede de fornecimento de software (RFS) é uma série de organizações de software, hardware e serviços vinculados que cooperam para satisfazer as demandas do mercado (ALVES; OLIVEIRA; JANSEN, 2017). Uma Rede de Fornecimento de Software modela as interações ocorridas entre fornecedores e clientes de soluções em software, considerando-se toda a cadeia de valores que se cria através do licenciamento das soluções. No contexto da ontologia SECOn, redes de fornecimento de software são representadas através do uso de uma solução de software por outra na forma de dependências, representando assim conceitos básicos da camada transacional de ecossistemas.

Podemos considerar, portanto, que a ontologia SECOn foi definida unindo os principais conceitos relacionados ao domínio de evolução de software em conjunto com os principais conceitos de redes de fornecimento de software, dentro de um ambiente de interações sociais online. Desta forma, a ontologia é capaz de expressar conceitos relacionados à troca de informação no âmbito de desenvolvimento distribuído de software, englobando as principais formas de troca de informação detectáveis em ambientes computacionais. Ou seja, interações sociais, colaboração no desenvolvimento de software e uso de dependências externas. A Figura 8 apresenta as principais classes da ontologia em notação gráfica.

Figura 8 - Ontologia SECOn



Revisitando a visão tridimensional proposta por CAMPBELL e AHMED (2010), a dimensão arquitetural é capturada através das interações entre as classes *Platform*, *Software*, *Component* e *Product*. Estas relações modelam as relações do mundo real entre plataforma e as soluções que a complementam. A dimensão social pode ser vista nas relações entre as entidades (*Entity*) e os espaços sociais (*Space*). Aqui ocorre a troca de conhecimento através das postagens (*Post*) em fóruns (*Forum*). A dimensão transacional por sua vez é representada também pelas relações entre *Platform*, *Software*, *Component* e *Product*. Esta dimensão não foi explorada em detalhes, já que existe um número muito grande de possibilidades de transações. Portanto, apenas a transação mais básica, quando um software utiliza outro, foi representada[4].

Para a especificação da ontologia SECOn foi utilizada a linguagem OWL[5], conjuntamente com a ferramenta Protégé[6]. Além das classes, propriedades e relacionamentos, foram especificados um conjunto de axiomas lógicos, com o objetivo de auxiliar no processamento de conhecimento novo. Estes axiomas foram especificados na linguagem SWRL (HORROCKS *et al.*, 2004). No Apêndice A desta dissertação, são apresentados o modelo de classes, propriedades, relações e regras SWRL da ontologia SECOn utilizando lógica descritiva (BAADER, 2003).

Ao se executar os mecanismos de inferência sobre a ontologia populada é possível extrair informações implícitas através das definições de *property chains*. *Property chains* são construtos que ligam indivíduos que fazem parte de uma cadeia de propriedades de objetos relacionados entre si. Na SECOn são definidas quatro *property chains* sobre as relações básicas que podem ser utilizadas pela arquitetura:

- *isRequiredBy o belongsToEcosystem SubPropertyOf: belongsToEcosystem;*
- *dependsOn o belongsToEcosystem SubPropertyOf: belongsToEcosystem;*
- *develops o requiresKnowledge SubPropertyOf: mayBeAcquaintedWith;*
- *isAcquaintedWith o isRequiredBy SubPropertyOf: mayCollaborateWith.*

As duas primeiras *property chains* garantem, ao serem inferidas, que um produto pertence ao ecossistema. Isso garante ao usuário e à arquitetura que ainda se está dentro do escopo do ecossistema, ou não. Como exemplo (o modelo completo relacionado a este exemplo será apresentado na Seção 3.4), ao saber que os softwares nos repositórios do *Github* utilizam dependências do ecossistema Node.js, sabe-se que os softwares analisados também pertencem ao Node.js.

---

[4] No entanto, em versões futuras da ontologia e dada a importância dessa dimensão no contexto organizacional, esta vertente pode ser explorada criando-se uma ontologia específica para esta dimensão.
[5] https://www.w3.org/OWL/
[6] https://protege.stanford.edu

A terceira *property chain* confere a uma entidade a possibilidade de estar relacionada a um tópico de conhecimento. Portanto, basta à implementação da arquitetura, verificar se a entidade tem conhecimento de fato sobre o tópico tratado. No exemplo do ecossistema Node.js (Seção 3.4), ao verificar que um desenvolvedor trabalha em um projeto que usa a dependência *express*, o *wrapper* do sistema de recomendação verifica se de fato o desenvolvedor tem conhecimento sobre a dependência através da análise histórica do código.

A quarta *property chain* verifica a possibilidade de um desenvolvedor ser capaz de trabalhar em um projeto que requer tópicos de conhecimentos que são de conhecimento do desenvolvedor.

A ontologia SECOn é utilizada no contexto da arquitetura Collab-RS para auxiliar na recomendação de desenvolvedores externos. Ela tem como principal função na arquitetura, representar os conceitos de Ecossistemas de Software para ser populada por indivíduos das classes nela representadas. Desta forma, a ontologia deve ser capaz de oferecer suporte à inferência de novo conhecimento que é utilizado pelo método de recomendação. Apresentamos a seguir a arquitetura Collab-RS e na Seção 3.4 um estudo de viabilidade de uso da ontologia SECOn no contexto da arquitetura Collab-RS.

3.3 ARQUITETURA COLLAB-RS

A Figura 9 apresenta o modelo conceitual da arquitetura Collab-RS, que foi especificado segundo o padrão arquitetural orientado a serviços (BUSCHMANN, 1996), permitindo a interoperabilidade entre seus módulos (serviços) e garantindo acesso remoto a suas funcionalidades. A arquitetura Collab-RS permite a recomendação de colaboradores para projetos de software através da mineração de repositórios de software e redes sociais. O uso da ontologia SECOn permite que informações implícitas sejam processadas através da execução de algoritmos de inferência (*reasoner* Pellet[7]) sobre as informações advindas da mineração. Estes dados processados são então disponibilizados para o sistema de recomendação.

A arquitetura se divide em dois módulos: Serviço de Acesso à Ontologia e o Serviço de Recomendação de Candidatos. A subseção 3.3.1 apresenta o Serviço de Acesso à Ontologia, responsável por popular, executar o *reasoner* e pesquisar na ontologia; e a subseção 3.3.2 apresenta o Serviço de Recomendação de Candidatos, responsável por realizar

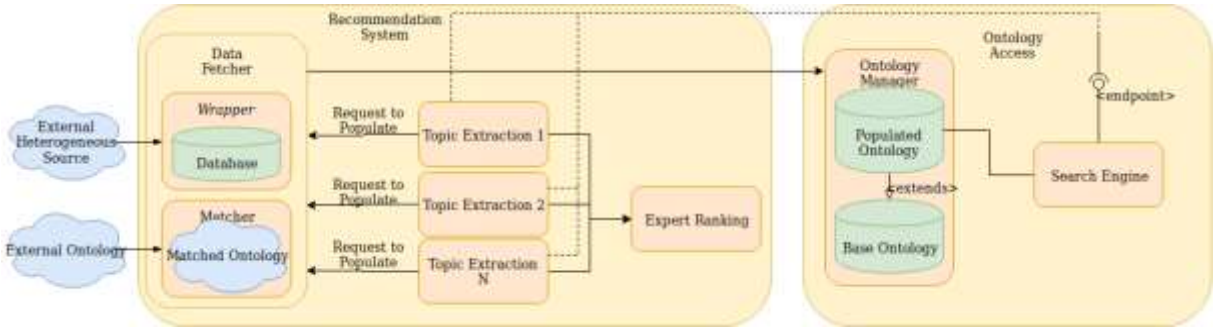
[7] https://github.com/stardog-union/pellet

a recomendação de candidatos a partir de dados processados pelo Serviço de Acesso à Ontologia.

Por ser uma arquitetura, a Collab-RS deve permitir ser estendida para diferentes Ecossistemas analisados, bem como deve ser independente do gerenciador de ontologias. A extensibilidade se verifica pela separação da arquitetura em serviços que interoperam com fontes de dados externas. Da mesma forma, o serviço de acesso à ontologia se utiliza do gerenciador de ontologias JENA[8], o qual responde a *queries* na linguagem SPARQL[9], o qual pode ser trocado, enquanto se utilizar esta linguagem para se realizar as buscas na ontologia.

A arquitetura deve ser também capaz de realizar as recomendações com um conjunto mínimo de informações, desde que tenha indivíduos da classe *Topic* que estejam relacionados aos desenvolvedores, assim, não havendo um conjunto mínimo de informações.

Figura 9 - Arquitetura Collab-RS



3.3.1 Serviço de Acesso à Ontologia

Este serviço é responsável por manipular a ontologia, processar as instâncias e inferências sobre estas instâncias para descoberta de novas conexões entre as informações e permitir consultas sobre as informações processadas.

Este serviço é baseado no *framework* Apache JENA. O *framework* é responsável pela maioria das operações processadas na ontologia, exceto pelo processamento das inferências, uma vez que o algoritmo nativo do JENA não processa *property chains*, e a arquitetura Collab-RS usa esse mecanismo. Assim, o algoritmo escolhido para processar inferências foi o Pellet (SIRIN *et al.*, 2007), que é capaz de processar *property chains*.

Para executar estas atividades, o serviço fornece um *endpoint*. Este *endpoint* permite consultas na linguagem SPARQL diretamente no modelo processado. O *endpoint* responde

[8] https://jena.apache.org/
[9] https://www.w3.org/TR/rdf-sparql-query/

através de requisições HTTP através de uma API Restful (Tabela 5). A partir desses *endpoints*, a arquitetura é capaz de popular, realizar inferência e fazer buscas sob a ontologia.

3.3.2 Serviço de Recomendação

Esse serviço é responsável por realizar as recomendações, incluindo recuperar informações dos repositórios de software, através do uso de *wrappers*, e processar as informações necessárias à atividade de recomendação, calculando o nível de expertise de cada indivíduo, além de disponibilizar os dados de contexto de recomendação baseado em consultas (*query*) relacionadas aos tópicos de interesse.

Tabela 5 - Acesso aos *Endpoints*[10]

Endpoint	Verbo HTTP	Ação
`/individual/{name}`	GET	Retorna a URI de um indivíduo cujo sufixo seja igual a `{name}`
`/objectproperty/ {obj_prop}`	GET	Retorna a URI de uma *object property* cujo sufixo seja igual a `{obj_prop}`
`/dataproperty/ {data_prop}`	GET	Retorna a URI de uma *data property* cujo sufixo seja igual a `{data_prop}`
`/setobjectproperty/ {subject}/{object}/ {prop}`	POST	Cria a relação dada pela propriedade `{prop}` entre os indivíduos `{subject}` e `{object}`
`/setstringproperty/ {subject}/{string}/ {prop}`	POST	Cria a relação dada pela propriedade `{prop}` entre a *string* `{string}` e o indivíduo `{subject}`
`/setintegerproperty/{subject}/{integer}/{prop}`	POST	Cria a relação dada pela propriedade `{prop}` entre o inteiro `{integer}` e o indivíduo `{subject}`
`/addindividual/ {name}/{class}`	POST	Cria um indivíduo da classe `{class}` com um sufixo `{name}`
`/getowl`	GET	Retorna o documento OWL que

[10] Nota-se que os as requisições GET são de recuperação de informação, enquanto as requisições POST alteram o estado do serviço, de acordo com as definições do protocolo HTTP. Todos os parâmetros podem utilizar tanto a URI de um objeto, quanto o sufixo da ontologia principal.

		representa a ontologia
`/executereasoner`	POST	Executa o *reasoner*
`/{query}`	GET	Retorna um arquivo JSON com os resultados da *query* SPARQL `{query}`

Esta consulta é composta por um conjunto de termos e dependências. A *expertise* pode ser tanto definida pelo domínio quanto pela familiaridade com as tecnologias envolvidas. A Figura 10 mostra um exemplo básico (modelo completo detalhado na Seção 3.5) de uma consulta para a recomendação de candidatos que trabalhem com desenvolvimento de *webservices backend* em Node.js e que utilize os módulos *express* e *body-parser*.

Figura 10 - *Query* em formato JSON

```
{
  "query":{
      "terms":[
        "backend",
        "webservice"
      ],
      "dependencies":[
        "express",
        "body-parser"
      ]
  }
}
```

Inicialmente, o serviço recebe a *query* de tópicos de interesse. Esta é utilizada para se realizar a extração dos tópicos de interesse advindos das fontes de dados externas (*e.g.* dependências e termos que são de interesse à recomendação). Em seguida, o serviço busca as informações necessárias à recomendação na ontologia. Caso as informações necessárias não estejam presentes na ontologia, o *wrapper* relacionado a fonte de dados é inicializado para realizar a busca por estes dados. Este mecanismo busca minimizar o impacto de se recuperar informações de fontes externas, acessando dados já instanciados na ontologia por outros *wrappers*.

Os *wrappers* e *matchers*[11] são responsáveis por obter dados de diferentes repositórios, realizando o pré-processamento, que inclui limpar e resumir os dados e traduzi-los em

[11] *Matchers* são softwares capazes de identificar a equivalência entre componentes de uma ontologia (classes, indivíduos, *properties*)

instâncias da ontologia, incluindo fazer o reconhecimento de tópicos de interesse, relacionando-os aos desenvolvedores que têm familiaridade com estes.

Wrappers e *matchers* são criados de acordo com os tipos de dados e suas origens. Por exemplo, um *wrapper* pode ser responsável por obter metadados dos repositórios de controle de versão, enquanto outro *wrapper* recuperar informações em repositórios relacionados a aplicativos de troca de mensagens. Essas informações externas também podem ser armazenadas em um banco de dados ou até mesmo em um arquivo RDF, evitando consultas constantes a fontes de dados externas, caso se opte por uma arquitetura de integração de informação do tipo *datawarehouse*. No entanto, o mecanismo nativo da arquitetura Collab-RS é a arquitetura virtual de dados (DOAN *et al.*, 2012).

No caso de outras ontologias como fontes de informação, os *matchers* devem ser capazes de realizar *matchings* entre os termos das duas ontologias, ou seja, ontologia relacionada a fonte de dados e ontologia utilizada pela arquitetura Collab-RS. Este processamento é utilizado para que os termos da ontologia externa possam ser traduzidos em termos da ontologia SECOn.

Em ambos os casos, as informações são passadas para o serviço de acesso à ontologia por meio de um *endpoint* disponibilizado pelo serviço para instanciar a ontologia.

3.3.2.1 Serviço de Recomendação de Candidatos

Com a ontologia instanciada é possível realizar a recomendação dos candidatos . A recomendação se dá através de técnicas de *expertise retrieval* que se utilizam de modelos de candidatos (BALOG *et al.*, 2012). Tais técnicas buscam gerar um conjunto de conhecimento dos candidatos a partir dos documentos que a estes estão associados. Estas técnicas contrastam com as técnicas que utilizam modelos de documentos, que verificam os documentos mais relacionados aos tópicos da *query*, e então buscam os candidatos relacionados a estes. Portanto, é necessário que se identifique os tópicos tratados diretamente pelos candidatos. Assim, os *wrappers* ao obterem os dados das fontes externas, ficam responsáveis por reconhecer e adicionar os tópicos de interesse tratados pelos candidatos à ontologia. Por exemplo, o *wrapper* responsável por analisar o código-fonte deve ser capaz de reconhecer com quais dependências o candidatos trabalha.

A seguinte fórmula define a expertise do candidato, utilizando a técnica de modelo generativo de tópico com estimativa baseada em perfil (FANG; ZHAI, 2007). Esta técnica cria um valor suavizado do grau de *expertise* para uma dada *query*, já que a técnica *a priori*

(BALOG *et al.*, 2012)considera que um candidato deve ter conhecimento sobre todos os termos membros de uma *query*.

$$P(e|\theta_q) = \prod_{t \in q} \left\{ \sum_{d} P(t|d)P(d|e) \right\}^{n(t,q)}$$

Onde, P($e|\theta q$) é a expertise do candidato *e*, *t* são os termos da *query q*, P($t|d$) =1 se o termo *t* está no documento *d* (P($t|d$)=0, caso contrário), e P($d|e$) = 1 se o candidato e faz uso do termo *t* no documento *d* (P($d|e$)=0, caso contrário) e $n(t,q)$ é a quantidade de vezes que o termo *t* aparece na *query q*.

Consideramos como documentos todo indivíduo da classe *Item*, como definido pela ontologia SIOC (e.g. post, comentário, etc), e todo arquivo encontrado nos repositórios de código-fonte. No caso de indivíduos da classe *Item* da ontologia SIOC, observa-se se ocorre a citação do termo *t* nos textos relacionados ao item. No caso de arquivos, caso estes sejam apenas texto, o mesmo tratamento se aplica. Já em caso de serem código-fonte, o termo *t* da *query q* passa a ser uma dependência. Neste contexto, o termo *t* ser citado significa tanto ocorrer a importação da dependência, quanto a invocação de métodos e valores desta ao longo do código.

Podemos, portanto, simplificar que há avaliação de uso dos termos *t* da *query q* em dois casos:

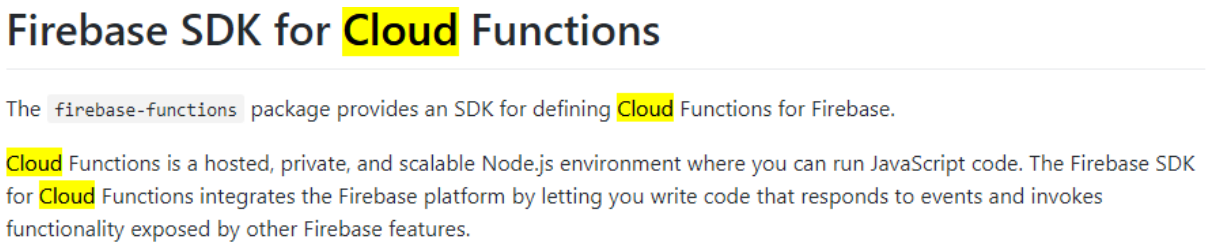
- Caso o conteúdo seja texto em linguagem natural, é avaliada a presença dos termos da *query* em citações literais ao longo do corpo do texto. A Figura 11a e a Figura 11b mostram exemplos de ocorrência do termo "*cloud*" em conteúdo de linguagem natural, sendo que o primeiro se encontra no documento "leia-me" do projeto "firebase-functions", enquanto o segundo é um comentário em uma discussão sobre uma falha no código, também citando o termo de interesse.
- Caso o conteúdo avaliado seja código, é avaliada a importação e uso das dependências. A Figura 12 apresenta a importação das dependências "firebase-admin" e "lodash", atribuídas às variáveis "firebase" e "_", que são utilizadas em outro ponto do código.

Temos então que cada citação do termo *t*, ou o uso ou importação das dependências são uma instância da presença de um termo da *query q*, no conteúdo avaliado.

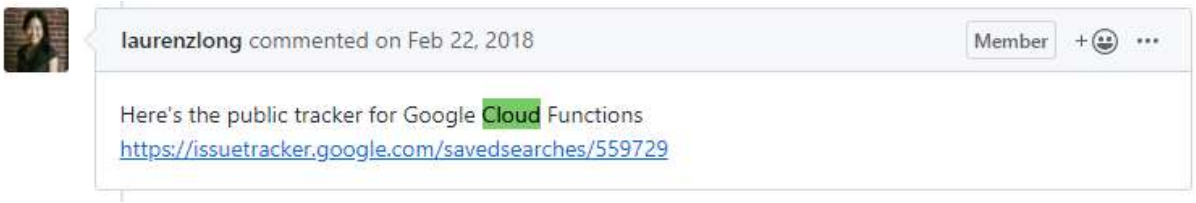
Para verificar a viabilidade de uso da arquitetura Collab-RS e a real capacidade de recomendar desenvolvedores externos para projetos, foi realizado um estudo de viabilidade

utilizando um dos maiores projetos do GitHub, o Node.js. A seção seguinte detalha este estudo de viabilidade, discutindo ao final, as lições aprendidas.

Figura 11 – Ocorrências do Termo Cloud



a: Ocorrência do termo cloud no arquivo leia-me



b: Ocorrência do termo cloud nas discussões de falha

Figura 12 – Ocorrência de uso de dependências e chamadas a métodos das dependências

```
import * as firebase from 'firebase-admin';
import * as _ from 'lodash';
import { firebaseConfig } from './config';
 ...
private get firebaseArgs() {
  return _.assign({}, firebaseConfig(), {
    credential: firebase.credential.applicationDefault(),
  });
}
```

3.4 ESTUDO DE VIABILIDADE

Como primeira etapa para a avaliação da arquitetura Collab-RS, é necessário verificar sua viabilidade e as tecnologias adotadas. Um estudo de viabilidade é uma opção apropriada, pois verifica a possibilidade de concluir um projeto com sucesso, incluindo todos os fatores relevantes. O objetivo principal deste estudo é criar conhecimento suficiente sobre a aplicação de uma tecnologia (SHULL *et al.*, 2004). Dessa forma, o pesquisador pode avaliar se essa tecnologia atende aos objetivos inicialmente definidos, justificando a manutenção ou não da pesquisa. O conhecimento obtido com a aplicação de um estudo de viabilidade fornece uma base para a proposta de melhorias.

A estrutura utilizada no estudo de viabilidade foi adaptada da formalização de estudos de caso apresentados por WOHLIN *et al.* (2012): (i) definição do estudo; (ii) formalização; (iii) planejamento; (iv) execução; e (v) apresentação das evidências observadas.

3.4.1 Definição do Estudo e Formalização

Como já apontado, a arquitetura Collab-RS visa melhorar a busca ativa de colaboradores considerando repositórios de código social de código aberto e redes sociais. Para tanto, o escopo deste estudo foi definido de acordo com a metodologia GQM (VAN SOLINGEN *et al.*, 2002): analisar a abordagem Collab-RS com o objetivo de avaliar sua viabilidade em relação à sua capacidade de extrair informações implícitas que auxiliem a recomendação de desenvolvedores externos para projetos globais de desenvolvimentos de software.

Com este escopo definido, podemos derivar a seguinte questão de pesquisa que nos guiará pela avaliação inicial:

(QP) - *Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS?*

Podemos, a partir da questão de pesquisa derivar duas questões secundárias que devem ser verificadas:

(QP1) – *Como a arquitetura facilita a identificação de conhecimento implícito dos colaboradores?*

(QP2) – *Como a arquitetura facilita o reconhecimento do contexto em que o tópico de conhecimento é abordado?*

O estudo foi então definido em duas fases, a primeira é apresentada nesta seção e apresenta uma verificação inicial da viabilidade da solução sobre o sistema Node.js, e a outra é apresentada no Capítulo 4, avaliando a extensibilidade da solução ao contexto de ecossistemas de software.

A plataforma Node.js[12] foi escolhida como fonte de dados para a realização deste estudo de viabilidade. O Node.js é um interpretador orientado a eventos de JavaScript baseado no interpretador V8 JavaScript Engine do Google Chrome[13]. Para extrair informações de dependências, comumente chamadas de pacotes, as informações contidas no Node Package

[12] https://nodejs.org/
[13] https://v8.dev

Manager[14] (NPM) foram usadas. O NPM fornece um conjunto de informações, como repositórios públicos, dependências de pacotes e outras informações que podem ser extraídas por meio de mineração a partir de sua API e arquivos de descrição de pacotes (ou módulos).

Combinando esses dados extraídos com os repositórios do Github, as interações do usuário podem ser extraídas, incluindo pacotes já utilizados pelos usuários, fóruns de discussão na forma das *issues threads* do Github, entre outros.

O processo começa recebendo a *query* de termos para a recomendação, em seguida recupera a informação baseada nessa *query* e calcula a expertise dos candidatos. Essas etapas estão detalhadas nas Seções 3.4.2 e 3.4.3.

3.4.2 Planejamento e Cenário

O projeto de software "firebase-functions" foi selecionado para ser explorado e seus dados utilizados neste estudo de viabilidade. No contexto escolhido, a recomendação dos colaboradores é dada pela familiaridade dos contribuintes com os pacotes do projeto selecionado e o uso dos termos da *query* nas discussões do Github.

A familiaridade com os pacotes usados desempenha um papel fundamental no processo de recomendação, devido ao conhecimento prévio tácito dos candidatos. Conhecendo certas dependências, os candidatos aumentam a chance de ter uma introdução mais rápida ao projeto. As dependências devem ser declaradas em um arquivo JSON que o NPM usa para gerenciá-los.

O arquivo *package.json* armazena os metadados do projeto, como o nome do projeto, o repositório no qual ele está hospedado, os autores e outras informações. Neste estudo de viabilidade, consideramos apenas duas informações sobre esses pacotes: dependências de tempo de execução (*dependencies*) e dependências de desenvolvimento (*devDependencies*). Dependências de tempo de execução são as dependências que estão sempre presentes no código-fonte, reconhecíveis pelas instruções *require*. Ao usar uma instrução *require*, a dependência é associada a uma variável ou constante que é usada em todo o código. Dependências de desenvolvimento podem ou não aparecer no código. Estas são dependências que visam tornar o trabalho do desenvolvedor mais fácil, como *scripts* para reiniciar o código toda vez que ele para de rodar por algum erro, para propósitos de teste.

O foco nas dependências é dado porque reconhecer os candidatos que têm conhecimento sobre as mesmas tecnologias do projeto é importante no contexto de

[14] https://www.npmjs.com

desenvolvimento distribuído de software (CONSTANTINOU; KAPITSAKI, 2016; MONTANDON; SILVA; VALENTE, 2019), pois isso facilita parcialmente a introdução do desenvolvedor ao projeto devido a seu conhecimento tácito prévio sobre as tecnologias. Assim, a introdução de um novo desenvolvedor ao projeto é dada por uma curva de aprendizagem mais suave, uma vez que o desenvolvedor tem algum conhecimento das tecnologias, enquanto a transmissão de conhecimento tácito é facilitada pela experiência anterior do desenvolvedor com a tecnologia.

Para a realização da *query*, definimos também um conjunto de termos que devem ser observados em textos em linguagem natural. Assim sabemos dos termos que são importantes ao recrutamento para um dado projeto. No caso de projetos Node, podemos usar as palavras-chave (*keywords*) utilizadas para definir os projetos.

Para exemplificar, podemos verificar na Figura 13 quais são os termos da *query q* como apresentados no arquivo *package.json* do projeto. Para a execução da recomendação basta adaptar essas informações para a sintaxe apresentada na Figura 13. A *query* final é apresentada na Figura 14.

3.4.3 Execução

Para realizar o estudo de viabilidade, a arquitetura Collab-RS instanciou na ontologia SECOn, considerando o projeto *firebase-functions*, os termos da *query*, a partir do serviço de acesso ontológico. Estes tópicos foram então adicionados à ontologia como indivíduos e foram então relacionados ao projeto por meio da propriedade de objeto *requiresKnowledge* da ontologia SECOn, o que significa que, para trabalhar com o projeto alvo da query, é necessário algum conhecimento sobre tais tópicos. Esse relacionamento é fundamental para especificar a necessidade de conhecimento sobre um tópico em um projeto. Neste estudo de viabilidade, é analisado o grau de *expertise* dos candidatos em relação aos termos da *query*.

Limitou-se a recomendação para candidatos que possuem algum grau de conhecimento tácito sobre o projeto, isso inclui seguidores, candidatos que realizaram *forks* do repositório, e colaboradores que postaram em threads de discussão do projeto, ou seja, colaboradores que participaram das *threads* de *issues*. Para selecionar uma amostra para o sistema de recomendação, 300 desses candidatos foram selecionados aleatoriamente. Estes candidatos foram assim selecionados por serem membros mais próximos e com maior possibilidade de interagirem com o projeto (SYED; JANSEN, 2013).

Inicialmente, para se obter as informações do Github, foi criado um *wrapper* para buscar as informações pela API Rest disponibilizada pelo próprio Github[15]. Um problema foi encontrado logo no início, devido às limitações deste recurso. Esta só permitia 5.000 requisições por hora, fazendo com que o acesso aos dados fosse muito lento devido ao número de requisições que eram necessárias para se obter informações das discussões disponíveis por repositório. Para tanto, foi necessária uma busca por ferramentas que permitissem acesso mais rápido aos dados disponíveis no Github. A solução então encontrada foi o GHTorrent[16].

Figura 13 - Termos utilizados na *query* de acordo com arquivo *package.json*


```
{
    "keywords": [
        "firebase",
        "functions",
        "google",
        "cloud"
    ],
    "dependencies": {
        "@types/express": "^4.17.0",
        "cors": "^2.8.5",
        "express": "^4.17.1",
        "jsonwebtoken": "^8.5.1",
        "lodash": "^4.17.14"
    },
    "devDependencies": {
        "@types/chai": "^4.1.7",
        "@types/chai-as-promised": "^7.1.0",
        "@types/cors": "^2.8.5",
        "@types/jsonwebtoken": "^8.3.2",
        "@types/lodash": "^4.14.135",
        "@types/mocha": "^5.2.7",
        "@types/mock-require": "^2.0.0",
        "@types/nock": "^10.0.3",
        "@types/node": "^8.10.50",
        "@types/sinon": "^7.0.13",
        "chai": "^4.2.0",
        "chai-as-promised": "^7.1.1",
        "child-process-promise": "^2.2.1",
        "firebase-admin": "^8.2.0",
        "istanbul": "^0.4.5",
        "js-yaml": "^3.13.1",
        "mocha": "^6.1.4",
        "mock-require": "^3.0.3",
        "mz": "^2.7.0",
        "nock": "^10.0.6",
        "prettier": "^1.18.2",
        "sinon": "^7.3.2",
        "ts-node": "^8.3.0",
        "tslint": "^5.18.0",
        "tslint-config-prettier": "^1.18.0",
        "tslint-no-unused-expression-chai": "^0.1.4",
        "tslint-plugin-prettier": "^2.0.1",
        "typedoc": "^0.14.2",
        "typescript": "^3.5.2",
        "yargs": "^13.2.4"
    }
}
```


[15] https://developer.github.com/v3/
[16] http://ghtorrent.org

Figura 14 - Termos utilizados na query, de acordo com sintaxe definida pela arquitetura


```
{
    "terms": [
        "firebase",
        "functions",
        "google",
        "cloud"
    ],
    "dependencies": {
        "@types/express": "^4.17.0",
        "cors": "^2.8.5",
        "express": "^4.17.1",
        "jsonwebtoken": "^8.5.1",
        "lodash": "^4.17.14",
        "@types/chai": "^4.1.7",
        "@types/chai-as-promised": "^7.1.0",
        "@types/cors": "^2.8.5",
        "@types/jsonwebtoken": "^8.3.2",
        "@types/lodash": "^4.14.135",
        "@types/mocha": "^5.2.7",
        "@types/mock-require": "^2.0.0",
        "@types/nock": "^10.0.3",
        "@types/node": "^8.10.50",
        "@types/sinon": "^7.0.13",
        "chai": "^4.2.0",
        "chai-as-promised": "^7.1.1",
        "child-process-promise": "^2.2.1",
        "firebase-admin": "^8.2.0",
        "istanbul": "^0.4.5",
        "js-yaml": "^3.13.1",
        "mocha": "^6.1.4",
        "mock-require": "^3.0.3",
        "mz": "^2.7.0",
        "nock": "^10.0.6",
        "prettier": "^1.18.2",
        "sinon": "^7.3.2",
        "ts-node": "^8.3.0",
        "tslint": "^5.18.0",
        "tslint-config-prettier": "^1.18.0",
        "tslint-no-unused-expression-chai": "^0.1.4",
        "tslint-plugin-prettier": "^2.0.1",
        "typedoc": "^0.14.2",
        "typescript": "^3.5.2",
        "yargs": "^13.2.4"
    }
}
```


Criou-se então *wrappers* para acessar as informações obtidas pela ferramenta GHTorrent. O GHTorrent é um conjunto de serviços e ferramentas disponíveis na web que monitora os eventos públicos do Github, extrai informações destes eventos e armazena em bancos de dados para fornecer uma solução mais escalável para recuperar informações públicas do Github em comparação com a própria API do Github (GOUSIOS, 2013). Ele armazena mais de 4 TB de dados e permite consultas definidas pelo usuário aos bancos de dados, em contraste com os recursos limitados que a API do Github fornece.

Outro *wrapper* que tem a função de minerar dados estáticos e implícitos do repositório usa a ferramenta ANTLR[17] (*ANorther Tool for Language Recognition*) e a ferramenta Git Blame[18]. Para informar a ontologia sobre os módulos com os quais os candidatos estão familiarizados, a ferramenta ANTLR foi usada em todos os projetos pertencentes aos candidatos. O ANTLR é um gerador de parsers que pode ser usado para ler e processar códigos estruturados, permitindo ao usuário criar uma árvore de análise que representa o código para análise em um nível sintático (PARR, 2014). Esse tipo de representação nos permite recuperar linhas nas quais as dependências foram chamadas, por exemplo.

O ANTLR analisa arquivos de código em uma árvore de elementos atômicos. Da árvore gerada pelo ANTLR, foi realizada uma busca por elementos, buscando ocorrências da declaração *require*. Na representação em árvore do ANTLR, o elemento pai do elemento que

[17] https://www.antlr.org
[18] https://git-scm.com/docs/git-blame

contém o comando *require* representa a linha de comando na qual *require* é invocado. Tendo o elemento que representa a linha, podemos verificar em qual constante ou variável o conteúdo importado pelo comando está armazenado e qual pacote foi importado. Podemos então continuar com o *parser* do arquivo, observando o uso dessa constante ou variável. Sempre que a constante ou variável for usada, registra-se em qual linha ocorreu sua invocação para verificar quem utilizou a invocação. A Figura 15 mostra uma linha contendo um comando *require*, enquanto a Figura 16 mostra a representação disso no ANTLR. A Figura 17 apresenta uma linha na qual a dependência importada é usada.

O *wrapper* então usa o comando Git Blame para recuperar qual desenvolvedor modificou as linhas de importação e uso do conteúdo de dependência pela última vez, para que possamos adicionar as dependências como um tópico conhecido para esse usuário. As dependências conhecidas de alguns desenvolvedores estão vinculadas ao indivíduo ontológico que representa o desenvolvedor por meio do relacionamento *isAcquaintedWith*.

Figura 15 - Exemplo de importação de módulo usando uma declaração *require*

```
const functions = require('firebase-functions');
```

Figura 16 - Representação em árvore dos elementos atômicos de código pelo ANTLR

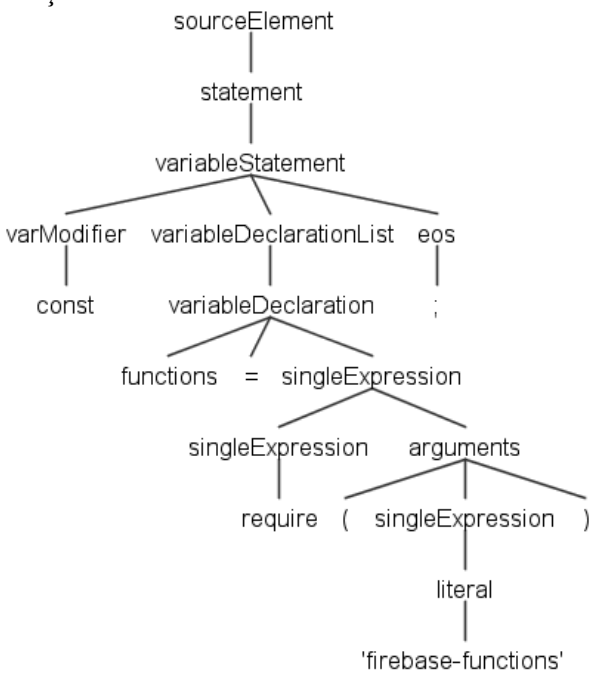


Figura 17 - Exemplo de uso de variável previamente importada pela declaração *require*

```
exports.addMessage = functions.https.onRequest((req, res) => {
```

Após as novas informações serem adicionadas, o *reasoner* Pellet foi executado, o que permitiu que novas informações fossem inferidas e a função de recomendação fosse processada. A Figura 18 apresenta a informação inferida sobre um candidato, usando a interface da ferramenta Protégé[19] para facilitar a visualização das informações. A propriedade inferida *isAcquaintedWith* informa que o candidato trabalhou em projetos usando a dependência, mas não necessariamente usou essa dependência, por exemplo, no caso de dependências de desenvolvimento, que são utilizadas durante o desenvolvimento, mas não fazem parte do código. Enquanto isso, a propriedade inferida *mayCollaborateWith* mostra projetos que o candidato pode colaborar desde que tenha usado pelo menos uma de suas dependências em outros projetos.

Neste estudo de viabilidade, o interesse estava em encontrar candidatos que já tivessem usado o conteúdo das dependências que são usadas no projeto de destino e discutiram sobre os termos da *query* que não são dependências. Portanto, a função de grau de expertise definida anteriormente é calculada para cada candidato.

Figura 18 - Exemplo de axiomas ontológico inferidos sobre um desenvolvedor na interface da ferramenta Protégé

O estudo foi realizado em um computador com as seguintes configurações: Intel® Core™ i5 6200U 2.30 GHz, 1x8GB + 1x4GB de RAM DDR3L 1600 MHz SDRAM, Windows 10 Pro. Foram selecionados 300 *stargazers* e colaboradores do próprio projeto de forma aleatória (~48%), dos quais foram observadas as participações em projetos na plataforma Node.js. Estes foram adicionados à ontologia, junto aos projetos em que

[19] https://protege.stanford.edu

colaboraram. Para cada projeto, foram adicionadas suas dependências, e sempre que a ontologia inferia que um desenvolvedor tinha a possibilidade de ter utilizado alguma dependência da *query*, os *wrappers* específicos para descobrir com quais dependências os desenvolvedores do projeto trabalharam eram executados. Dessa forma a ontologia teve informação suficiente para realizar a recomendação.

3.4.4 Evidência observada

Após executarmos a atividade de recomendação, obtivemos as informações apresentadas na Figura 19, com a pontuação de cada desenvolvedor pelo método de recomendação.

A pontuação e ranqueamento por si só não contém dados que geram subsídios sobre os quais os recrutadores possam agir. Para essas análises por parte dos recrutadores, a arquitetura Collab-RS possui uma interface (página WEB) onde apresenta detalhes das principais contribuições de cada desenvolvedor. Esta página informa com quais das dependências o desenvolvedor já trabalhou, criando um link para a página do *commit* em que ele realizou a ação, bem como cria *links* para as mensagens nos fóruns de discussão para os termos da *query*. A Figura 20 apresenta a interface da Collab-RS onde essa informação é exibida, considerando este estudo de viabilidade.

Figura 19 - Pontuação e ranqueamento dos candidatos

Developer	Score	Details link
laurenzlong	1.0000000000	
inlined	0.9794359003	
mbleigh	0.9789781968	
thechenky	0.9770043105	
merlinnot	0.9692393755	
joehan	0.9674863772	
kevinajian	0.9634830403	
esmacik	0.9632386337	

Figura 20 - Visualização da utilização das dependências da query para um candidato específico

Term	Usage Type	Project	Preview
chai	Import	laurenzlong/sample-firebase-functions	
chai	Use	laurenzlong/sample-firebase-functions	
chaiAsPromissed	Import	laurenzlong/sample-firebase-functions	
sinon	Import	laurenzlong/sample-firebase-functions	
sinon	Use	laurenzlong/sample-firebase-functions	
lodahs	Import	firebase/firebase-tools	

Desta forma, considerando as informações obtidas a partir do processamento do *dataset* completo relacionado ao projeto *firebase-functions* do ECOS Node, pudemos discutir as questões de pesquisa, a luz dos resultados obtidos por este estudo de viabilidade.

(QP1) – Como *a arquitetura facilita a identificação de conhecimento implícito dos colaboradores?*

Como pode-se perceber, o uso das dependências das *queries* só é possível de ser identificado a partir da verificação da participação de um desenvolvedor em projetos que requerem tal dependência. Isso leva à execução do *wrapper* responsável por verificar quais desenvolvedores usaram quais dependências especificamente, assim adicionando mais informação à ontologia, o que permite identificar o conhecimento implícito dos colaboradores. No entanto, é importante também destacar que os termos em linguagem natural, por sua vez, são explícitos, já que fazem parte do conteúdo textual.

(QP2) – Como *a arquitetura facilita o reconhecimento do contexto em que o tópico de conhecimento é abordado?*

Os *links* que levam às páginas dos *commits* onde o candidato utilizou as dependências, fornecem informações contextuais que facilitam a análise pelos recrutadores, o que permite a possibilidade de se verificar o conhecimento dos candidatos, e possivelmente permitir maior compreensão sobre o nível de conhecimento sobre as dependências. O mesmo vale para conteúdo textual, para o qual o recrutador pode avaliar o nível de domínio do colaborador ao acessar as mensagens diretamente.

(QP) - *Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS?*

Assim, considerando os resultados das duas questões de pesquisa secundárias, pode-se então verificar que a ferramenta auxilia os recrutadores ao indicar onde as informações que estes devem verificar estão localizadas. Isto acontece quando a ontologia é capaz de inferir a possibilidade de desenvolvedores terem trabalhado com alguma das dependências da *query*. Ao identificar essa possibilidade através do *reasoner*, o sistema de recomendação ativa novos *wrappers* que verificam o código do projeto. Feito isto, a ontologia fica munida de informação que ajuda a identificar as instâncias que indicam que os colaboradores tenham conhecimento sobre os itens da *query*.

Portanto, a disponibilização das instâncias que indicam conhecimento sobre os itens da *query* permite uma visualização direta dos artefatos em que o desenvolvedor colaborou, reduzindo-se o tempo de análise de portfólio. Ranqueando os desenvolvedores, indica-se primeiramente os desenvolvedores com maior produção relacionada aos tópicos de interesse, o que facilita ao recrutador encontrar desenvolvedores que têm maior domínio sobre o foco do recrutamento, dado que a *query* também adere ao foco do recrutamento.

Portanto, esta primeira etapa da avaliação teve como objetivo avaliar se a arquitetura Collab-RS auxilia os recrutadores na busca por contribuidores externos em projetos de desenvolvimento de software globais, a partir do estudo do projeto firebase-functions do Node. Os resultados preliminares apontam pela viabilidade da arquitetura. A partir de ajustes na arquitetura, de forma a englobar outros aspectos além das questões relacionadas a conhecimento tecnológico dos desenvolvedores, foi realizado um novo estudo, contemplando o contexto de ecossistemas de software científico e, portanto, verificar se neste contexto, a Collab-RS também é capaz de recomendar contribuidores.

3.4.5 Ameaças à validade

Como ameaças à validade, podemos mencionar a possibilidade de desenvolvedores não identificados contribuírem para os repositórios do Github. Ao colaborar em um clone local de um repositório, algumas alterações podem não corresponder ao desenvolvedor real, já que o git permite que o usuário insira seus e-mails manualmente, sem qualquer mecanismo de verificação. Portanto, é possível enviar alterações para o Github com *e-mails* do Git não registrados por um usuário do Github, impossibilitando a localização do colaborador original. Para evitar esse tipo de problema, a ontologia trata o *e-mail* como o identificador único dos

indivíduos. Assim, *e-mails* identificam colaboradores exclusivamente, permitindo recomendar até mesmo colaboradores que não fazem parte da comunidade do Github.

Deve-se observar também que o Git Blame considera que para qualquer mudança em uma linha, a ferramenta indica o último colaborador a modificar a linha como responsável. Isso pode implicar em casos em que o desenvolvedor apenas refatore uma linha, sem mesmo conhecer o significado dela, fique como responsável por modificá-la.

Deve-se verificar que o ranqueamento não utiliza uma abordagem que analisa o conteúdo de linguagem de natural de forma semântica. Portanto, pode haver correspondências de termos semanticamente diferentes, bem como considerar dúvidas e questionamentos com o mesmo peso que uma mensagem que realmente demonstre o domínio do desenvolvedor sobre o tópico.

Também é importante ressaltar que os resultados obtidos devem ser tratados como uma evidência preliminar dos benefícios do uso do Collab-RS. Assim, devemos enfatizar que a validade das conclusões deste estudo depende da replicação do estudo em outros contextos para garantir a viabilidade da abordagem e aumentar a validade do estudo. No entanto, embora os resultados não possam ser generalizados, é possível identificar situações em que resultados semelhantes podem ser alcançados.

Finalmente, como não há um conjunto mínimo de dados para que a recomendação ocorra, um conjunto de poucos dados tem maior probabilidade de ser impreciso.

4 PROVA DE CONCEITO

Considerando o contexto de ECOSs, este capítulo apresenta o uso da Collab-RS aplicada a um ECOS científico, denominado E-SECO (FREITAS *et al.*, 2015).

4.1 E-SECO

E-Science é um campo da ciência que utiliza ferramental computacional para o estudo e compreensão de sistemas científicos complexos (HEY *et al.*, 2009). Ao se trabalhar com e-Science, cientistas tentam compreender melhor a teoria através de experimentos e simulações complexos executados, em geral, através do processamento de grande volume de informações. Para que isso seja possível, é importante que seja definido de forma precisa o ciclo de vida destes experimentos científicos.

Neste sentido, o E-SECO (e-Science Ecosystem) (FREITAS *et al.*, 2015) é um ecossistema de software científico cuja funcionalidade principal é apoiar o ciclo de vida de experimentação científica em e-Science. Ou seja, experimentação apoiada por ferramental computacional. O E-SECO engloba uma plataforma criada e mantida pelo Núcleo de Engenharia do Conhecimento (NEnC) da Universidade Federal de Juiz de Fora (UFJF), conectando diversos serviços externos, cada qual focado em apoiar uma ou mais fases da condução dos experimentos. A plataforma fornece um ambiente colaborativo e distribuído através de uma rede ponto-a-ponto (P2P), onde cada nó desta rede é uma instância da plataforma localizada em alguma entidade de pesquisa (CLASSE *et al.*, 2017).

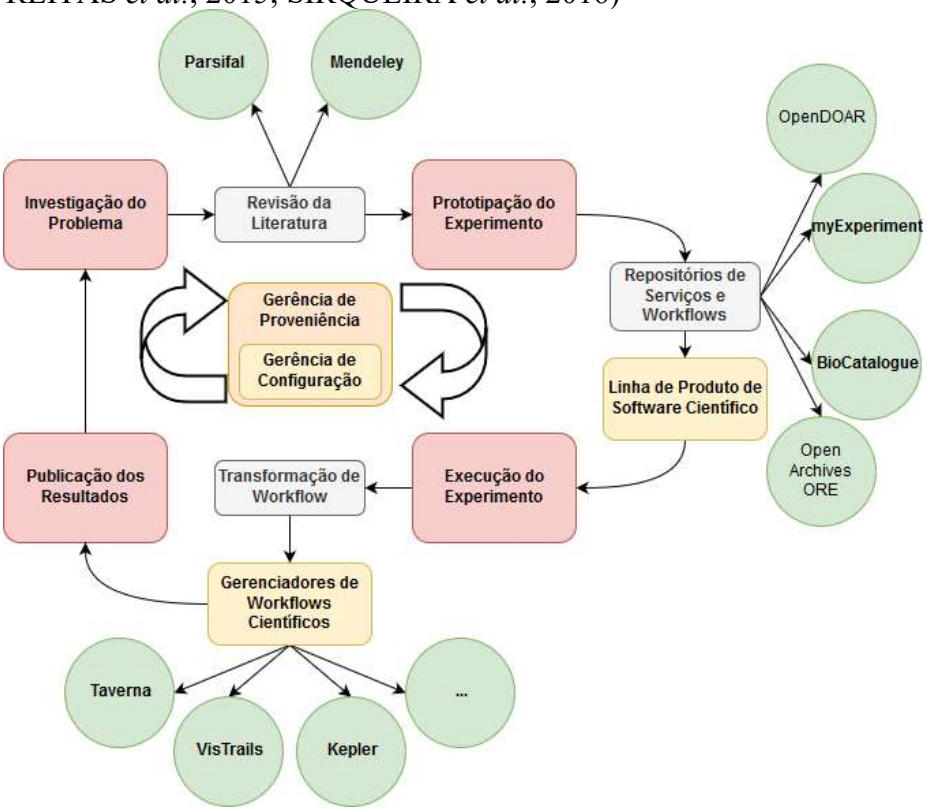
O E-SECO divide suas atividades em investigação do problema, prototipação do experimento, execução do experimento e publicação de resultados. Cada uma dessas fases é realizada por aplicações relacionadas a plataforma do E-SECO. A Figura 21 apresenta os serviços que apoiam cada fase.

A atividade de investigação do problema é apoiada por duas ferramentas: Parsifal[20] e Mendeley[21]. O Parsifal é uma ferramenta web de revisão e mapeamento sistemáticos que segue as diretrizes do protocolo de revisão definido por KITCHENHAM (2004), oferecendo um espaço compartilhado para a documentação e montagem do protocolo, execução das buscas em bases de dados, seleção de trabalhos e relatório de mapeamento. Enquanto o Mendeley oferece serviços web, desktop e mobile para gerenciamento, descoberta e

[20] https://parsif.al/
[21] https://www.mendeley.com/

compartilhamento de referências bibliográficas em conjunto a plugins que integram as referências ao Word, LibreOffice e software que utilizam o formato BibTeX.

Figura 21 - Ciclo de vida de experimentos científicos apoiado pelo E-SECO (AMBROSIO *et al.*, 2017; FREITAS *et al.*, 2015; SIRQUEIRA *et al.*, 2016)



A atividade de prototipação do experimento é apoiada pelo módulo de composição de serviços e pelos repositórios de *workflows* e serviços científicos, entre eles tem-se os repositórios myExperiment[22] e BioCatalogue[23] já integrados ao E-SECO, além de outros repositórios, como OpenDOAR[24] e Open Archives ORE[25]. Estes repositórios armazenam os metadados dos *workflows* e serviços que são utilizados pelos cientistas. A partir da escolha/desenvolvimento dos serviços a serem utilizados, passa-se para a fase de composição de serviços (MARQUES *et al.*, 2017). No processo de composição, realiza-se a busca pelos serviços que satisfazem as necessidades de uma especificação inicial de composição de serviços para o experimento.

Após a composição dos serviços, abrangendo a fase de prototipação, dá-se início à atividade de execução do experimento, onde a composição gerada é traduzida para uma

[22] https://www.myexperiment.org/home
[23] https://www.biocatalogue.org
[24] http://v2.sherpa.ac.uk/opendoar/
[25] https://www.openarchives.org/ore/

linguagem intermediária (BASTOS; BRAGA; GOMES, 2015), que permite a especificação do *workflow* em um dos Softwares Gerenciadores de *Workflows* Científicos (SGWfCs) suportados, a saber, Taverna[26], Kepler[27] ou VisTrails[28] e em seguida executados no SGWfC escolhido, gerando assim os resultados das experimentações in silico.

Estes resultados são disponibilizados na plataforma, com o suporte dos gerenciadores de configuração e de proveniência (AMBROSIO *et al.*, 2017; SIRQUEIRA *et al.*, 2016). Assim, através das informações do contexto de proveniência e do gerenciador de configuração, o E-SECO possibilita o reuso dos dados do experimento e sua reprodutibilidade. Portanto, verifica-se que o E-SECO dá suporte as quatro atividades de experimentação, cada uma apoiada por diferentes ferramentas e em diferentes níveis.

O E-SECO tem como foco a dimensão arquitetural de um ECOS, que é de fato importante para o desenvolvimento da plataforma. No entanto, as outras dimensões, transacional e social são também importantes no contexto de um ECOS, principalmente em um ECOS científico, como é o caso do E-SECO, considerando a escassez de desenvolvedores especializados e a necessidade primordial de colaboração entre estes e os cientistas relacionados ao domínio científico específico do experimento a ser conduzido. Neste sentido, a arquitetura Collab-RS traz um suporte importante, a partir da recomendação de desenvolvedores para aplicações científicas relacionadas a um dado experimento. Esse suporte é essencial para a saúde do E-SECO, auxiliando na sua expansão e aderência de desenvolvedores e cientistas à plataforma.

Neste sentido, a verificação da aplicabilidade da Collab-RS no contexto do E-SECO é importante. Assim, foi conduzida uma Prova de Conceito (*Proof of Concept - PoC*). Uma prova de conceito é um método utilizado para se verificar a aplicabilidade de um conceito teórico em um cenário prático (BELL, 1993). No contexto de uma arquitetura, a prova de conceito engloba a implementação desta arquitetura e a verificação dos seus efeitos na prática. As provas de conceito normalmente podem ser divididas sob a perspectiva do que exploram: estrutura e comportamento. As que exploram estrutura podem ser usadas para comparar tecnologias e/ou ferramentas, enquanto as provas de conceito que exploram comportamento investigam um cenário de utilização. Nesta dissertação, foi adotada a perspectiva que explora o comportamento em um cenário de utilização.

[26] https://taverna.incubator.apache.org/
[27] https://kepler-project.org/
[28] https://www.vistrails.org/

O objetivo desta prova de conceito foi definido de acordo com o "Goal" da abordagem Goal/Question/Metric (GQM) (VAN SOLINGEN *et al.*, 2002). Os objetivos, segundo a abordagem GQM, devem ser formulados conforme o template a seguir:

"Analisar o <objeto de estudo> com a finalidade de <objetivo> com respeito à <foco da qualidade> do ponto de vista de <perspectiva> no contexto de <contexto>".

Com base no template para objetivos do GQM, o objetivo desta prova de conceito é: "Analisar a arquitetura Collab-RS com a finalidade de verificar a recomendação de desenvolvedores externos para experimentos do ponto de vista de desenvolvimento de aplicações científicas no contexto do ECOS E-SECO".

Seguindo a metodologia de Design Science e a questão de pesquisa principal formulada, devemos verificar: "*Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS*"

Para isso, a Seção 4.2 apresenta o planejamento da PoC, a Seção 4.3 detalha o cenário de condução da PoC e a Seção 4.4 apresenta os resultados.

4.2 PLANEJAMENTO

Esta PoC foi conduzida ao longo dos meses de junho de 2019 a outubro de 2019. A arquitetura Collab-RS foi adaptada para que fosse capaz de realizar a busca de informações nas bases dos serviços de repositório de *workflows* já integrados ao E-SECO (*i.e.,* myExperiment e BioCatalogue). Estes repositórios oferecem metadados sobre os serviços e provêm uma forma de desenvolvedores colaborarem uns com os outros através de anotações semânticas aos serviços oferecidos. A Figura 22 apresenta a implementação da arquitetura no contexto do E-SECO.

Figura 22 - Arquitetura Collab-RS no contexto do E-SECO

Um ponto importante a ser destacado é a flexibilidade da Collab-RS, que pode ser configurada para ser utilizada em diferentes contextos. Diferente do uso no contexto do projeto Node.js apresentado no Capítulo 3, os repositórios relacionados a experimentos científicos, como é o caso do myExperiment e Biocatalogue, não oferecem informações da implementação dos serviços. Ou seja, não estão disponíveis informações mais específicas sobre os tópicos de conhecimento dos desenvolvedores, como é o caso do Node.js. No entanto, possuem anotações semânticas relacionados aos serviços, o que pode trazer um diferencial em relação ao domínio dos serviços desenvolvidos, como tópicos, informação de domínio, entre outros.

4.3 CENÁRIO

Para que as informações semânticas pudessem ser utilizadas, foi necessário, assim como no caso do Node.js, um pré-processamento dos dados. Foi utilizada a seguinte configuração computacional para o pré-processamento dos dados: Processador Intel® Core™ i5 6200U 2.30 GHz, 1x8GB + 1x4GB de RAM DDR3L 1600 MHz SDRAM, Windows 10 Pro.

Primeiramente, as anotações foram extraídas dos repositórios de experimentos científicos e foram persistidas em um banco de dados, juntamente com os dados dos desenvolvedores. Em seguida, foram extraídos os tópicos de conhecimento dos desenvolvedores. Esta última etapa foi realizada em dois passos: extração dos tópicos explícitos, e extração dos tópicos implícitos baseados na query de consulta[29]. Os tópicos explícitos e implícitos são aqueles que fazem parte das anotações.

Ambos os serviços oferecem as seguintes anotações para seus *workflows*: título, descrição, *tags*, *workflows* dos quais dependem, e *workflows* que deles dependem, e o BioCatalogue, exclusivamente, oferece anotações de categorias. O repositório myExperiment trata categorias e *tags* como um mesmo campo. Os tópicos explícitos são então retirados das *tags*, categorias e dependências. Estes tópicos são relacionados a cada *workflow* na ontologia SECOn. Foram extraídos então os tópicos que estavam implícitos. Estes são observados no título e descrição, já que estes conteúdos são puramente textuais. Os termos da *query* foram comparados com o conteúdo textual, e em caso de correspondência dos termos, o tópico do termo foi relacionado ao *workflow* correspondente na ontologia SECOn.

[29] A *query* é definida como um conjunto de termos, que deve ser comparada ao conteúdo textual relacionado direta ou indiretamente aos colaboradores. No cenário do E-SECO, esta deve conter os termos buscados nas *tags*, títulos, descrições e outros comentários sobre os *workflows*.

A partir da instanciação da ontologia SECOn com os dados extraídos dos repositórios, foi processado o algoritmo de inferência (Pellet) sobre estes dados, considerando os tópicos de conhecimento de cada desenvolvedor através das relações de autoria dos *workflows* e serviços, e os tópicos de cada serviço. Para ilustrar, a Figura 23 apresenta como o BioCatalogue apresenta as informações anteriores: (A) título, (B) categorias, (C) *tags* e (D) descrição. Enquanto isso, a Figura 24 apresenta as anotações: (A) título, (B) descrição e (C) *tags* do myExperiment.

A partir do processamento da inferência, a recomendação foi então realizada, baseada na técnica de modelo generativo de tópico com estimativa baseada em perfil (FANG; ZHAI, 2007), definida no Capítulo 3. O valor do ranking foi então normalizado, considerando o desenvolvedor mais recomendado com o valor máximo de 1, e os demais com o valor proporcional a ele.

Para se realizar a PoC do uso da arquitetura Collab-RS no contexto do ECOS E-SECO foi necessário analisar as formas com a qual se encontram os desenvolvedores atualmente e em seguida avaliar se houve melhorias no processo de busca por desenvolvedores, respondendo à questão de pesquisa proposta.

Assim, foi planejada uma consulta (*query*) simples e uma consulta (*query*) mais complexa, envolvendo múltiplos termos, para verificar a capacidade da arquitetura em analisar ambas as bases e recomendar os desenvolvedores baseado em suas colaborações em cada uma destas, verificando assim a questão de pesquisa proposta: “Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS”.

4.4 EXECUÇÃO

Para que pudéssemos verificar se a arquitetura Collab-RS traria ganhos na recomendação de desenvolvedores no contexto do E-SECO, primeiro foi necessário verificar a forma com a qual desenvolvedores/especialistas podiam ser encontrados sem o uso da arquitetura. Ou seja, somente usando a plataforma E-SECO, sem o suporte da Collab-RS. Assim, a partir destes resultados, conduzir consultas a partir do uso da arquitetura Collab-RS. Assim, a Seção 4.4.1 apresenta a execução de consultas sem o uso da Collab-RS e a Seção 4.4.2 apresenta a execução das consultas com o uso da Collab-RS.

Figura 23 - Anotações do BioCatalogue

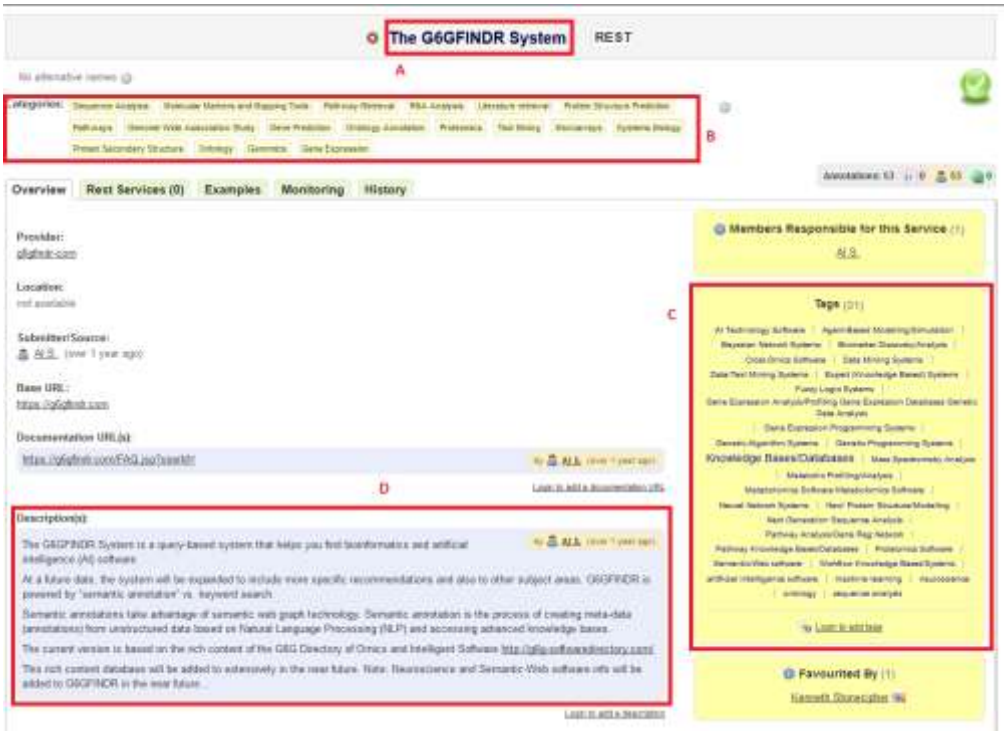
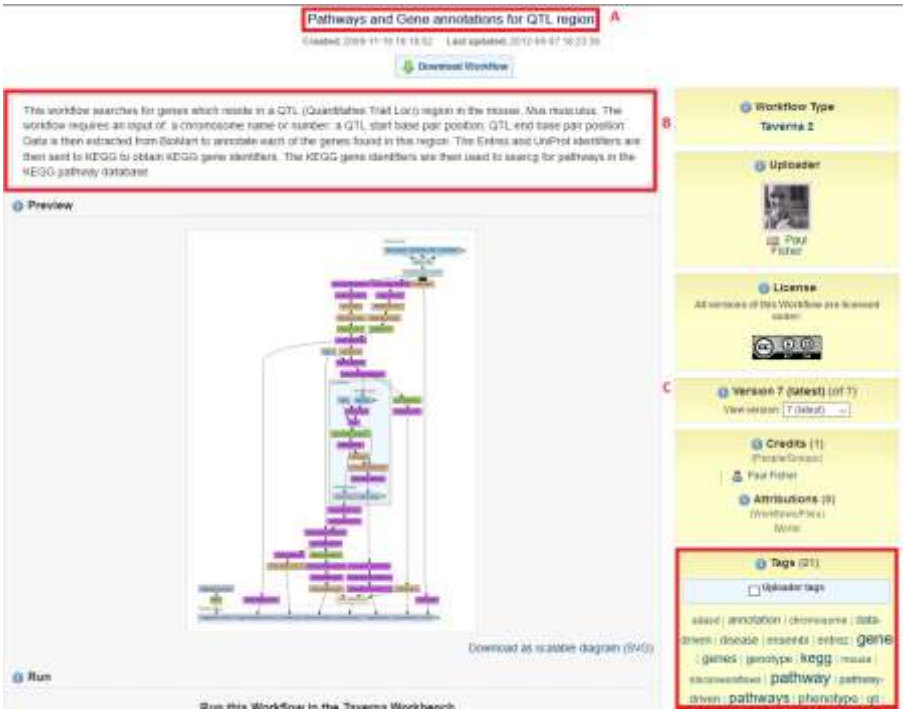


Figura 24 - Anotações do myExperiment



4.4.1 Busca por desenvolvedores especialistas sem o uso da arquitetura CollabRS

Para isso, foi realizada, em agosto de 2019 uma pesquisa pelo termo “KEGG” (Kyoto Encyclopedia of Genes and Genomes) que é um dos termos mais comumente utilizados tanto no myExperiment quanto no BioCatalogue. KEGG é uma coleção de bancos de dados que lidam com genomas, vias biológicas, doenças, medicações e substâncias químicas[30].

O E-SECO não oferecia uma forma direcionada de buscar pelos desenvolvedores relacionados a experimentos. Assim, foi necessário localizar diretamente pelas ferramentas dos repositórios. No caso do myExperiment, podemos buscar tanto pela *tag*, quanto podemos criar uma consulta. Ao utilizar a busca por *tag* do myExperiment, a consulta retornou apenas serviços/*workflows* com a *tag* KEGG anotada, no entanto muitos serviços/*workflows* tratam de informações relacionadas a KEGG e que não têm a *tag* anotada, sendo retornados apenas na busca com *query*. Verificou-se uma diferença entre os 101 serviços/*workflows* retornados pela query, contra os 58 retornados pela *tag*. No BioCatalogue, verificou-se uma situação mais complexa, visto que durante todo o processo de análise, a busca por *tag* não funcionou e a busca por *query* retornou apenas 14 serviços.

Em uma segunda tentativa, também em agosto de 2019, buscou-se pela combinação “KEGG” ou “gene”. Novamente o serviço myExperiment apresentou uma divergência com a busca com *query* retornando 176 resultados e a busca com *tags* retornando 97. Enquanto isso o BioCatalogue retornou nenhum serviço.

Pôde-se verificar nessas consultas, que o reconhecimento da expertise dos desenvolvedores fica por conta do usuário, já que nenhuma das duas ferramentas apresenta alguma forma de pesquisar sua relevância. O mais próximo que se pôde verificar é a quantidade de serviços/*workflows* que um desenvolvedor colaborou no myExperiment, como pode ser visto na Figura 25, mas ainda assim sem explicitar algum grau de relevância de cada desenvolvedor.

O BioCatalogue é ainda mais limitado, visto que ele sequer lista os desenvolvedores na página de busca, forçando quem estiver buscando por especialistas a abrir as páginas de cada serviço e verificar manualmente.

[30] https://www.genome.jp/kegg/

Figura 25 - Desenvolvedores dos serviços/*workflows* retornados pela query “KEGG” no myExperiment

4.4.2 Busca por desenvolvedores especialistas com o uso da arquitetura Collab-RS

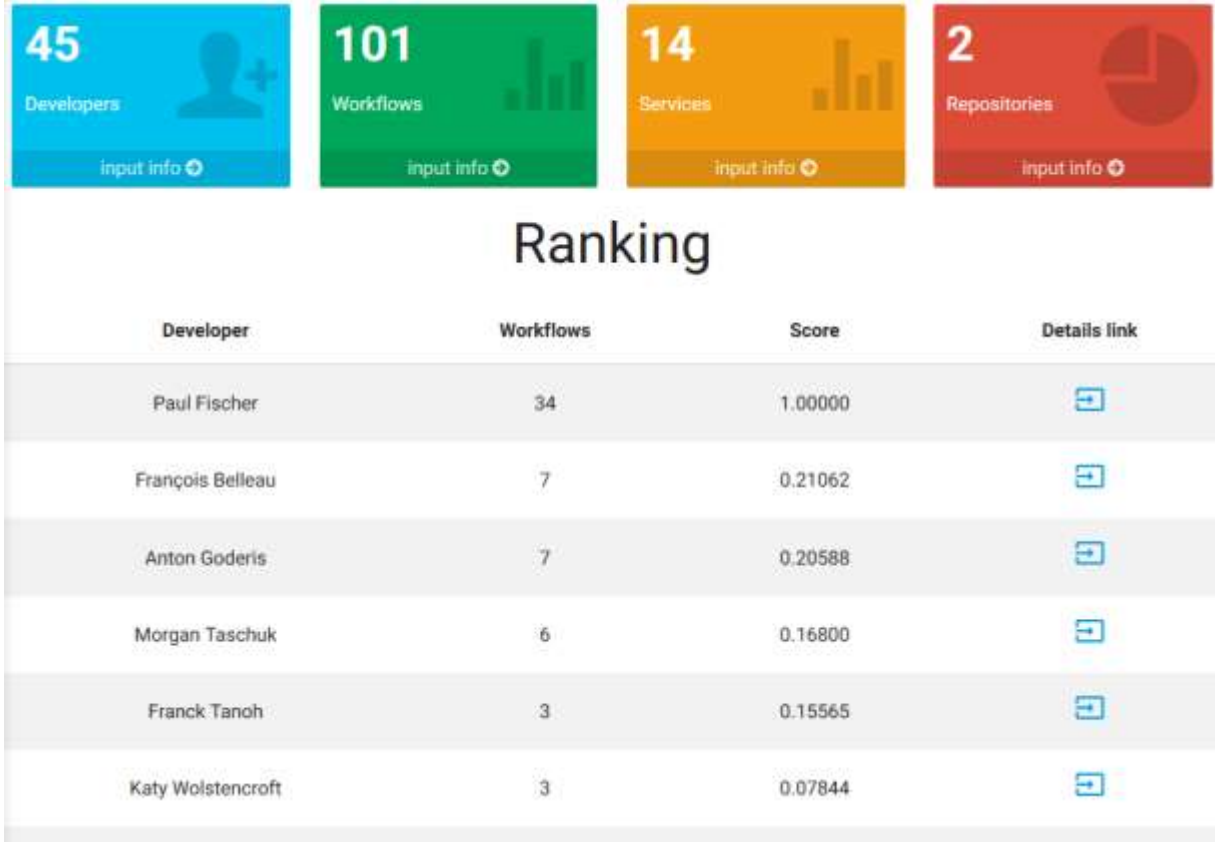
A partir da identificação das dificuldades em se encontrar especialistas para o desenvolvimento de serviços/*workflows* científicos, foi então planejada a realização, com o suporte da arquitetura Collab-RS, de duas buscas: uma com a *query* “KEGG” e outra com a query “KEGG” ou “protein” ou “gene”, para verificar o comportamento da solução de acordo com o aumento do número de termos.

A Seção 4.4.2.1 detalha a execução da PoC, considerando a consulta “KEGG”, que será aqui denominada Cenário 1, e a Seção 4.4.2.2 verifica o uso da *query* ““KEGG” ou “protein” ou “gene””, aqui denominada Cenário 2.

4.4.2.1 Cenário 1

Dada a query “KEGG”, era sabido, considerando o exposto na Seção 4.4.1, que havia um total de 101 serviços/*workflows* no repositório myExperiment e 14 serviços/*workflows* no repositório BioCatalogue. O objetivo era verificar se com o uso da arquitetura Collab-RS podíamos melhorar os resultados da busca considerando a *query* “KEGG”. Para isso foi realizada uma busca com a mesma *query* “KEGG” em agosto de 2019, mas utilizando a arquitetura Collab-RS e obteve-se os resultados apresentados na Figura 26.

Figura 26 - Desenvolvedores dos serviços/*workflows* retornados pela query “KEGG” utilizando a arquitetura Collab-RS



Como pode ser observado na Figura 26, informações básicas ao resultado da recomendação aparecem, como os nomes dos desenvolvedores, e o número de serviços/*workflows* que estes desenvolveram e/ou realizaram *upload*, dando alguma informação para ao recrutador. A tabela ficou bem próxima à lista de filtros da Figura 25, mudando apenas a ordem, o que é justificado pelo valor da coluna “Score”, que é a coluna com o valor normalizado dos *scores* de cada desenvolvedor calculados pelo algoritmo de modelo generativo de tópico com estimativa baseada em perfil pertencente à abordagem que apresenta-se neste trabalho (FANG; ZHAI, 2007). Portanto, verificamos alguma proximidade entre o resultado da busca e a abordagem desenvolvida neste trabalho, isto acontece pelo fato da *query* ter apenas um termo, portanto todos os *workflows* devem aparecer na busca nos repositórios, e como a ordenação dos autores nos repositórios depende do número de *workflows* por estes desenvolvidos, observa-se uma relação parecida com o resultado da Collab-RS, já que esta se baseia no número de aparições dos termos da *query*. Logo, autores com maior número de *workflows* tendem a ter maior número de menções aos termos, assim, também aparecendo na mesma ordem.

Ao abrir o link para os detalhes dos desenvolvedores, têm-se acesso à tela de contexto de recomendação do desenvolvedor. Nela é apresentada uma tabela com as informações que foram utilizadas na recomendação. Esta tabela é apresentada para detalhar a proveniência da recomendação. Ou seja, para que se possa verificar a confiabilidade da recomendação e os parâmetros utilizados. Isto indica a extensibilidade da arquitetura para o contexto de ECOSs, visto que se utiliza de conceitos próprios de ecossistemas, mas não necessariamente de código, como visto no caso do capítulo 3. A Figura 27 apresenta as informações do contexto de recomendação para um desenvolvedor.

Figura 27 - Informações do contexto de recomendação

Developer Profile

Ranking:	#1
Score:	1.0000
Name:	Paul Fisher
Username:	None
Email:	None
Date Joined:	Thursday 26 July 2007 16:46:52 (UTC)
Location:	Manchester, United Kingdom
Website:	http://twitter.com/#!/paul_r_fisher
URI:	Open external
URL:	Open external

Developer Interactions

Workflow	Location	Topic	Link
Pathways and Gene annotations for QTL region	Tag	KEGG	
Pathways and Gene annotations for QTL region	Description	KEGG	
HUMAN Microarray CEL file to candidate pathways	TAG	KEGG	
Entrez Gene to KEGG Pathway	Title	KEGG	
Entrez Gene to KEGG Pathway	Tag	KEGG	
Entrez Gene to KEGG Pathway	Description	KEGG	
Add Mesh String to Biological Process	Description	KEGG	
KEGG pathways common to both QTL and microarray based investigations	Title	KEGG	
KEGG pathways common to both QTL and microarray based investigations	Tag	KEGG	
KEGG pathways common to both QTL and microarray based investigations	Description	KEGG	

Ao clicar no ícone de *link* externo, a ferramenta redireciona para as páginas nas quais estas informações foram extraídas, permitindo que se obtenha um maior detalhamento sobre o tipo de informação que levou à recomendação. A origem dos tópicos está separada por tipo,

para que se possa verificar também as múltiplas fontes das informações obtidas através da query.

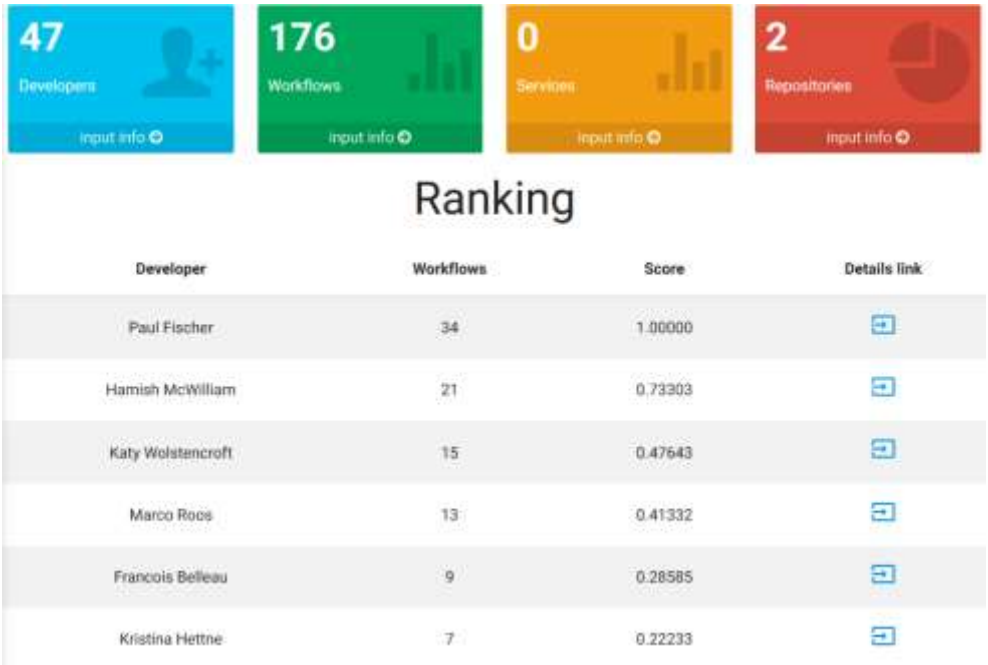
Com estas informações, verificamos que a arquitetura se aplica no contexto do E-SECO realizando as recomendações, considerando uma *query* simples. Para verificar seu uso em consultas mais complexas, foi conduzida nova consulta para verificar outros contextos onde a Collab-RS podia ser utilizada.

4.4.2.2. Cenário 2

Considerando os resultados apresentados na Seção 4.3.1, e considerando a *query* "KEGG" ou "gene" ou "protein", obteve-se um total de 176 serviços/*workflows* em ambos os repositórios, e um total de 87 serviços/*workflows* ao se realizar uma busca avançada pela mesma *query* com a busca do myExperiment, enquanto o repositório BioCatalogue não retornou nenhum resultado para a query.

Desta forma, com o uso da Collab-RS, o objetivo é verificar se é possível observar um maior detalhamento nos resultados da busca considerando a *query* "KEGG" ou "*gene*" ou "*protein*", incluindo referências à localização dos tópicos pesquisados. Para isso foi realizada uma busca com a *query* "KEGG" ou "*gene*" ou "*protein*" *em agosto de 2019*, utilizando a arquitetura Collab-RS. Os resultados são apresentados na Figura 28.

Figura 28 - Desenvolvedores dos *workflows* retornados pela *query* "KEGG" ou "gene" ou "protein" pela arquitetura Collab-RS



Novamente, verifica-se que as informações básicas da recomendação aparecem, acrescidas neste caso (uso da Collab-RS), das informações de contexto da recomendação. A Figura 29 apresenta parte da tabela com as informações de contexto da recomendação para o desenvolvedor com maior grau de recomendação para a *query* analisada. Verifica-se que agora os outros dois tópicos da *query* estão presentes na tabela, o que indica que a Collab-RS utilizou de todos os tópicos da *query* para uma busca, não se limitando a um único termo. Com isso é possível verificar a informação de contexto para cada termo da *query* de forma independente. Novamente, facilitando a análise das informações que o recrutador deve observar para se procurar um novo colaborador.

Figura 29 - Informações do contexto de recomendação para query composta

Developer Interactions

Workflow	Location	Topic	Link
Pathways and Gene annotations for QTL region	Title	gene	
Pathways and Gene annotations for QTL region	Tag	gene	
Pathways and Gene annotations for QTL region	Tag	KEGG	
Pathways and Gene annotations for QTL region	Description	gene	
Pathways and Gene annotations for QTL region	Description	KEGG	
HUMAN Microarray CEL file to candidate pathways	Description	gene	
HUMAN Microarray CEL file to candidate pathways	Tag	KEGG	
Retrieve Protein Sequence	Title	protein	
Retrieve Protein Sequence	Title	Tag	
Entrez Gene to KEGG Pathway	Tag	KEGG	
Entrez Gene to KEGG Pathway	Description	KEGG	
Add Mesh String to Biological Process	Description	KEGG	
KEGG pathways common to both QTL and microarray based investigations	Title	KEGG	
KEGG pathways common to both QTL and microarray based investigations	Tag	KEGG	
KEGG pathways common to both QTL and microarray based investigations	Description	KEGG	

Isto demonstra que a arquitetura Collab-RS é capaz de verificar a combinação de todos os termos nas diversas configurações possíveis dos serviços armazenados nos repositórios, e

como pode ser notado, retornou mais itens que em ambos os repositórios somados, demonstrando uma maior compreensão de *workflows* presentes nos repositórios, e provendo mais informação do cenário, sendo mais abrangente que os mecanismos de busca dos repositórios.

4.5 RESULTADOS E DISCUSSÕES

Durante a execução dos dois cenários anteriores, foi possível verificar que a arquitetura Collab-RS recomenda desenvolvedores externos para experimentos científicos relacionados ao ECOS E-SECO, já que cria um ranqueamento a partir de consulta aos dados disponíveis nos repositórios.

Além do ranqueamento, verificou-se a disponibilização das informações relevantes ao contexto da recomendação, permitindo que recrutadores verifiquem a origem das informações que foram descobertas durante o processamento dos dados que foram utilizados na atividade de recomendação. Estas informações foram descobertas através do processamento de inferências sobre as instâncias da ontologia SECOn, gerando conhecimento implícito a partir de novos relacionamentos sobre as informações.

Ao disponibilizar as informações do contexto de recomendação, a arquitetura dá maior poder de escolha aos recrutadores, enquanto justifica o ranqueamento apresentado inicialmente. Conferindo ao recrutador um novo olhar sobre informações que não poderiam ser obtidas por meios previamente disponíveis.

Para responder à questão de pesquisa "*Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS*", foi realizada uma avaliação com um especialista sênior, com grande experiência no uso do ECOS E-SECO para a condução de experimentos científicos.

Este especialista tem mais de 20 anos de experiência em desenvolvimento de software e 10 anos de experiência na condução de experimentos científicos utilizando plataformas computacionais. Possui doutorado em Engenharia de Sistemas, com ênfase em reuso de software.

A avaliação foi realizada a partir da disponibilização dos resultados das consultas sem a Collab-RS em uma primeira etapa e da execução do Cenário 1 e do Cenário 2 em uma segunda etapa.

Na primeira etapa foram apresentados ao especialista apenas os resultados considerando o E-SECO sem o uso da Collab-RS. Após sua avaliação e questionamentos

sobre a execução das consultas, foi pedido ao mesmo que resumisse sua impressão dos resultados retornados pela consulta. A seguinte resposta, reproduzida aqui usando as mesmas palavras do especialista, foi obtida: *"A plataforma E-SECO tem uma grande deficiência atualmente que é a falta de suporte ao desenvolvimento distribuído de aplicações científicas. É fácil especificar um experimento distribuído, onde cientistas do mundo todo podem colaborar. A partir de aplicações científicas já conhecidas da comunidade, a plataforma E-SECO é capaz de compor os experimentos científicos. No entanto, quando se quer desenvolver novas aplicações para serem utilizadas nos experimentos, o suporte é mínimo e mais ainda considerando a busca por desenvolvedores adequados. Os repositórios associados a E-SECO, como pode ser visto no retorno das consultas dessa Prova de Conceito, não auxiliam em nada nessa busca por desenvolvedores externos. Essas respostas reforçam a necessidade de evoluir a plataforma no suporte organizacional. Espero que a Collab-RS possa auxiliar em um primeiro passo nessa direção. Espero ansiosa pelos resultados alcançados pelo seu uso"*.

Na segunda etapa, após explicações iniciais de como foi aplicada a arquitetura Collab-RS à plataforma E-SECO, os resultados das consultas, utilizando a Collab-RS foram apresentados ao especialista. Este, após uma análise detalhada dos resultados, relatou: *"Sem dúvidas o uso da Collab-RS melhoram os resultados, considerando a busca por desenvolvedores especialistas em aplicações científicas relacionadas a tópicos específicos mas na verdade eu esperava resultados mais contundentes, no sentido de recomendar desenvolvedores relacionados a técnicas científicas específicas mas também que pudéssemos identificar desenvolvedores relacionados a tecnologias de desenvolvimento específicas, como java, python, etc, e necessidades específicas de retorno das aplicações científicas, que pudessem ser compostas de forma facilitada com outras aplicações em um workflow científico. No entanto, parabenizo o trabalho e os resultados retornados, pois sem dúvida é um passo importante para a plataforma E-SECO. Ressalto ainda que sei que os resultados podem ser sensivelmente melhorados a partir da melhoria dos metadados disponibilizados pelos repositórios associados. Nesse sentido, devemos fazer um esforço para criarmos um meta-repositório na plataforma E-SECO, que armazene metadados ligados às aplicações armazenadas nos repositórios associados, de forma a alimentar a busca da Collab-RS com mais informações, aprimorando seus resultados. De qualquer forma, tivemos avanços no suporte a busca por desenvolvedores externos na E-SECO, mas devemos continuar aprimorando. Considerando a pergunta que vc me fez: Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS,*

acredito que a mesma pôde ser respondida, uma vez que agora, com o uso da Collab-RS temos um mecanismo que apresenta, de forma detalhada, como se dá o processo de recomendação de desenvolvedores no E-SECO”.

Considerando as avaliações feitas pelo especialista pudemos ter indícios que a questão de pesquisa, *“Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS”* uma vez que este especialista sinalizou positivamente em relação a recomendação de desenvolvedores externos para experimentos científicos relacionados a um ecossistema de software realizada com o apoio da arquitetura Collab-RS. Esta recomendação é possível a partir do uso da ontologia SECOn, fornecendo ao recrutador informações capazes de ajudar na tomada de decisão, seja por via do ranqueamento, seja por via das informações de contexto.

Como resultado preliminar da PoC, consideramos os resultados da avaliação satisfatórios. No entanto, estes resultados não podem ser generalizados, e novos experimentos devem ser conduzidos, uma vez que a avaliação utilizada só é válida para o cenário e dados utilizados.

4.6. LIMITAÇÕES E AMEAÇAS À VALIDADE

Foram detectadas algumas ameaças à validade do estudo. Uma delas é o fato de apenas um especialista ter participado da avaliação. Apesar de ser um especialista sénior, somente uma avaliação pode enviesar os resultados. Desta forma, novos estudos devem ser conduzidos com um número maior de especialistas.

Nessa avaliação foi apresentada ao usuário uma seleção dos dados originais, referentes a apenas um projeto. Avaliações com outros dados podem levar a resultados diferentes dos obtidos nesta PoC.

4.7 CONSIDERAÇÕES FINAIS

Este capítulo detalhou uma PoC com o intuito de se realizar uma avaliação preliminar sobre o uso da arquitetura Collab-RS no contexto de um ECOS científico. Com a ajuda de um especialista do domínio, resultados preliminares foram discutidos, validando a questão de pesquisa proposta.

Diferente do contexto apresentado no Capítulo 3, com um estudo utilizando dados do Node.js, esta avaliação não utilizou a análises relacionadas às implementações dos usuários,

visto que estes detalhes não são disponibilizados pelos repositórios científicos. Neste sentido, pudemos verificar uma nova abordagem de uso da arquitetura Collab-RS, ao extrair tópicos de conhecimento de outras fontes. Estes mesmos tópicos ao serem extraídos, foram adicionados à ontologia, e a partir de então, tanto a inferência de conhecimento implícito, quanto a recomendação ocorreram da mesma forma que no contexto do Node.js.

Assim, considerando os resultados da recomendação de colaboradores no Node.js e de desenvolvedores de software científico no ECOS E-SECO, pudemos verificar a viabilidade da solução de considerando diferentes contextos. No próximo capítulo, são apresentadas as considerações finais e trabalhos futuros.

5 CONCLUSÃO

Este capítulo apresenta as considerações finais deste trabalho, suas contribuições, limitações e possíveis trabalhos futuros.

5.1 SUMÁRIO

Este trabalho detalhou a arquitetura Collab-RS, relacionada à recomendação de desenvolvedores externos para projetos de desenvolvimento de software. Para embasar o trabalho desenvolvido, esta dissertação apresentou os principais conceitos relacionados a ecossistemas de software e suas redes sociotécnicas.

Após um mapeamento sistemático da literatura relacionado ao tema, identificou-se a necessidade de apoiar a colaboração de indivíduos em ECOSs para o desenvolvimento de novas aplicações, ao mesmo tempo em que se pôde observar a ausência de trabalhos que tratem explicitamente deste tema. Considerando esta lacuna na literatura, a condução da pesquisa seguiu os *guidelines* de *Design Science Research*, para propor uma solução relacionada à identificação de desenvolvedores externos a projetos de software.

A abordagem se baseia em um sistema de recomendação capaz de utilizar dados advindos de redes sociotécnicas e dos artefatos envolvidos nela. Estas informações são representadas utilizando uma ontologia, que permite a representação das principais informações relacionadas, bem como, através de regras ontológicas, descobrir novos relacionamentos entre os conceitos.

Foram realizados dois estudos para verificar a viabilidade da solução. No primeiro, a solução foi aplicada ao projeto Node.js, mais especificamente sobre os repositórios do Node Package Manager (NPM) e Github. Neste primeiro estudo, verificou-se a viabilidade da solução sobre o Node ao ver que a Collab-RS foi capaz de encontrar tópicos de interesse considerando os artefatos dos projetos (como código e mensagens trocadas). No entanto, a avaliação deste estudo teve como objetivo verificar a viabilidade da solução técnica, o uso da ontologia, mecanismos de inferência e a troca de informação entre os módulos da Collab-RS para a realização de recomendações adequadas, respondendo assim a questão de pesquisa proposta "*Como a arquitetura Collab-RS pode recomendar desenvolvedores externos para projetos de software relacionados a um ECOS*", considerando a solução técnica proposta. No segundo estudo, verificamos a utilização da Collab-RS no contexto do ECOS E-SECO, considerando especificamente os artefatos dos repositórios myExperiment e BioCatalogue.

Neste estudo, a partir de análises de buscas por colaboradores externos com e sem o uso da Collab-RS e após uma avaliação de um especialista em um ECOS científico, pode-se obter indícios da viabilidade da proposta, considerando seu uso em um ECOS, a partir de uma avaliação inicial de um oráculo (profissional com mais de 20 anos de experiência, especialista no desenvolvimento de aplicações em um ECOS Científico).

Como os estudos conduzidos foram iniciais, um primeiro para verificar a viabilidade técnica da arquitetura e um segundo para verificar a viabilidade da solução para a recomendação de desenvolvedores em um ECOS científico, novos estudos experimentais devem ser conduzidos, de forma a comprovar os resultados destes estudos iniciais.

5.2 CONTRIBUIÇÕES

Podemos citar como contribuições deste trabalho:

- A condução de um mapeamento sistemático da literatura, que verificou o atual cenário da literatura sobre ecossistemas de software e suas redes socio-técnicas;
- O desenvolvimento de uma ontologia, denominada SECOn, com o objetivo de representar redes socio-técnicas de desenvolvimento de software relacionadas a ECOSs;
- Desenvolvimento de uma arquitetura, denominada Collab-RS, capaz de recomendar desenvolvedores para projetos de desenvolvimento distribuído de software;
- Estudos de viabilidade da solução proposta:
    - Um primeiro estudo, relacionado a capacidade de recomendar desenvolvedores para projetos do ecossistema Node.js;
    - Um segundo estudo, relacionado ao ECOS E-SECO, para verificar a recomendação de desenvolvedores para aplicações científicas relacionadas a experimentos científicos;

5.3 PESQUISA PUBLICADA

Foram publicados os seguintes trabalhos oriundos de resultados dessa dissertação:

- De Oliveira, M.T., Jr, Braga Villela, R.M.M, Nazar David, J.M, Pereira Araújo, M.A., De Andrade Menezes, V.S., Campos, F.C.A., 2018, April. **An**

ontology-based approach to identify developers in a software ecosystem based on their skillset. In 21st Ibero-American Conference on Software Engineering (CIbSE) (pp. 185-198).

- Este trabalho apresentou um estudo inicial da ontologia SECOn, para representar redes socio-técnicas de ECOSs utilizando ontologia;

- De Oliveira, M.T., Jr, Braga Villela, R.M.M, Ghiotto, G.M., Nazar David, J.M, 2019, September. **Recommending External Developers to Software Projects based on Historical Analysis of Previous Contributions**. In Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES) (pp. 417-426).
  - Este trabalho apresentou uma versão inicial da arquitetura Collab-RS, aplicada ao contexto do projeto Node.js.

5.4 LIMITAÇÕES E AMEAÇAS

Durante o desenvolvimento deste trabalho, verificou-se uma dificuldade em se encontrar especialistas capazes de realizar a atividade de recrutamento para os projetos, o que gerou dificuldades em alinhar as provas de conceito com necessidades reais de projetos em termos de recrutamento de desenvolvedores. Para lidar com este problema, ao aplicar a solução para o ECOS Node.js, utilizou-se como projeto-alvo da recomendação, um projeto com destaque no ecossistema, e que tivesse um conjunto de dependências interessantes ao recrutamento. Ainda assim, nota-se que a abordagem foi verificada apenas na linguagem JavaScript, portanto não se pode generalizá-la para outras linguagens.

Outra ameaça à metodologia do trabalho é que a avaliação da solução dentro do ECOS E-SECO foi realizada somente por um especialista. Isto torna o resultado suscetível a vieses deste especialista, já que o resultado dependeu de um único indivíduo. No entanto, não existe uma dificuldade inerente a este domínio pelo fato da comunidade que integra o ECOS ser pequena, e que a maior parte dos atores do E-SECO terem participado em fases distintas, durante um período limitado de tempo, tornando-o assim um ecossistema escasso em especialistas, apesar de apresentar muitos atores.

Quanto ao mapeamento sistemático, como exposto no Capítulo 2, a ausência da prática de *snowballing* pode ter deixado alguns trabalhos relevantes de fora do mapeamento. No entanto, a prática não é obrigatória pela metodologia de mapeamento sistemático, visando complementar o *corpus* levantado pelo mapeamento.

5.5 TRABALHOS FUTUROS

Como trabalhos futuros podemos citar:

- Complementar a implementação para o Ecossistema E-SECO, adicionando *wrappers* para os repositórios ainda ausentes no Ecossistema;
- Aperfeiçoar a técnica de recomendação, utilizando algoritmos que processem métricas relacionadas ao envolvimento de desenvolvedores em seus respectivos projetos, e no Ecossistema;
- Aperfeiçoar a arquitetura para que se utilize do Virtuoso[31] como gerenciador de ontologia, substituindo o JENA, assim evitando os problemas de carga que podem ocorrer com um maior volume de dados;
- Desenvolver uma abordagem capaz de utilizar de técnicas similaridade semântica entre a *query* e conteúdo textual para realizar a recomendação de desenvolvedores que tratam/trataram de tópicos relacionados à *query*;
- Criar uma implementação base da arquitetura que possa se utilizar de *wrappers* implementados a partir do ANTLR, para verificar o uso de dependências em outras linguagens (*e.g*. Python);
- Adicionar a revisão histórica de repositórios Git para melhorar a qualidade das recomendações sobre o uso de dependências;
- Verificar como as relações de poder e governança em ECOSs podem influenciar na recomendação, e adaptar a abordagem para se utilizar destas relações para a recomendação de candidatos.

[31] https://virtuoso.openlinksw.com/

REFERÊNCIAS

ADOMAVICIUS, G; TUZHILIN, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. **In *IEEE Transactions on Knowledge and Data Engineering.*** v. 17, n. 6, p. 734–749, 2005.

ALVES, Carina; OLIVEIRA, Joyce; JANSEN, Slinger. **Software ecosystems governance a systematic literature review and research agenda**. 2017. SciTePress, 2017. p. 215–226.

AMBROSIO, Lenita M. *et al.* WIP: **Prov-SE-O: A Provenance Ontology to Support Scientists in Scientific Experimentation Process**. 29 jun. 2017. Institute of Electrical and Electronics Engineers Inc., 29 jun. 2017. p. 15–21.

AUDY, Jorge Luis Nicolas.; PRIKLADNICKI, Rafael. **Desenvolvimento Distribuído de Software.** Elsevier Editora, 2007.

AVELINO, Guilherme *et al.* **Who Can Maintain this Code? Assessing the Effectiveness of Repository-Mining Techniques for Identifying Software Maintainers**. *IEEE Software*, p. 1–1, 2018.

BAADER, Franz. Terminological cycles in a description logic with existential restrictions. **In IJCAI'03: Proceedings of the 18th international joint conference on Artificial intelligence.** 2003. 2003. p. 325–330.

BALOG, Krisztian *et al.* **Expertise Retrieval.** Foundations and Trends® in Information Retrieval, v. 6, n. 2–3, p. 127–256, 2012.

BASTOS, Bruno Fernandes; BRAGA, Regina Maria Maciel; GOMES, Antônio Tadeu Azevedo. Scientific workflow interchanging through patterns: Reversals and lessons learned. **In 2015 IEEE 11th International Conference on e-Science**. 2015. 2015. p. 557–564.

BELL, David A. **From data properties to evidence**. *IEEE Transactions on Knowledge and Data Engineering*, v. 5, n. 6, p. 965–969, 1993.

BOBADILLA, J. *et al.* **Recommender systems survey**. *Knowledge-Based Systems*, v. 46, p. 109–132, 1 jul. 2013.

BOSCH, Jan. From software product lines to software ecosystems. **In SPLC '09 Proceedings of the 13th International Software Product Line Conference, 2009***, págs. 111-119*, p. 111–119, 2009.

BOUCHARAS, Vasilis; JANSEN, Slinger; BRINKKEMPER, Sjaak. Formalizing software ecosystem modeling. **In IWOCE '09: Proceedings of the 1st international workshop on Open component ecosystems**. 2009. 2009. p. 41–50.

BRESLIN, John G. *et al.* **SIOC: an approach to connect web-based communities**. *International Journal of Web Based Communities*, v. 2, n. 2, p. 133, 2006.

BUSCHMANN, Frank. **Pattern-Oriented Software Architecture: A System of Patterns**. Volume 1. Ashish Raut, 1996. v. 1.

CAMPBELL, P. R.J.; AHMED, Faheem. A three-dimensional view of software ecosystems. **In ECSA '10: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume.** 2010. 2010. p. 81–84.

CAO, Yunbo *et al.* **Research on Expert Search at Enterprise Track of TREC 2005**. TREC 2005. 2005.

CLASSE, Tadeu *et al.* **A Distributed Infrastructure to Support Scientific Experiments.** *Journal of Grid Computing*, v. 15, n. 4, p. 475–500, 1 dez. 2017.

CLEMENTS, Paul; NORTHROP, Linda. ***Software product lines : practices and patterns***. Addison-Wesley, 2002.

CONSTANTINOU, Eleni; KAPITSAKI, Georgia M. Identifying Developers' Expertise in Social Coding Platforms. **In 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).** 2016. IEEE, 2016. p. 63–67.

COUTINHO, Emanuel Ferreira; SANTOS, Italo; BEZERRA, Carla Ilane Moreira. **A Software Ecosystem for a Virtual Learning Environment: SOLAR SECO**. 28 jun. 2017. Institute of Electrical and Electronics Engineers Inc., 28 jun. 2017. p. 41–47.

CROOYMANS, Wesley; PRADHAN, Priyanka; JANSEN, Slinger. **Exploring network modelling and strategy in the Dutch software business ecosystem**. 2015. Springer Verlag, 2015. p. 45–59.

DOAN, AnHai *et al.* Introduction. **Principles of Data Integration**. Morgan Kaufmann. p. 1–18, 1 jan. 2012.

DOS SANTOS, Rodrigo Pereira *et al.* **Using Social Networks to Support Software Ecosystems Comprehension and Evolution**. *Social Networking*, v. 03, n. 02, p. 108–118, 27 fev. 2014.

FANG, Hui; ZHAI, ChengXiang. **Probabilistic Models for Expert Finding**. *Advances in Information Retrieval*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 418–430.

FERDMAN, Sela *et al.* **Quantifying the web browser ecosystem**. *PLOS ONE*, v. 12, n. 6, p. e0179281, 23 jun. 2017.

FRANCO-BEDOYA, Oscar *et al.* **Open source software ecosystems: A Systematic mapping**. *Information and Software Technology*, v. 91, p. 160–185, 1 nov. 2017.

FREITAS, V *et al.* An architecture for scientific software ecosystem. **In 11th Working IEEE/IFIP Conference on Software Architecture (WICSA)**. 2015. p. 41–48.

GALATEANU, Elena; AVASILCAI, Silvia. **Framing the Competitive Behaviors of Niche Players: The Electric Vehicle Business Ecosystem Perspective**. *Procedia - Social and Behavioral Sciences*, v. 221, p. 342–351, jun. 2016.

GOUSIOS, Georgios. **The GHTorrent dataset and tool suite**. MSR '13, 2013, Piscataway,

NJ, USA: IEEE Press, 2013. p. 233–236.

GUARINO, Nicola; OBERLE, Daniel; STAAB, Steffen. **What Is an Ontology?** *Handbook on Ontologies*. [S.l: s.n.], 2009. p. 1–17.

HEVNER, Alan R *et al.* **Design science in information systems research**. *Management Information Systems Quarterly*, v. 28, n. 1, p. 6, 2008.

HEY, Anthony J G *et al.* **The fourth paradigm: data-intensive scientific discovery.** Microsoft research Redmond, WA, 2009. v. 1.

HORROCKS, Ian *et al.* **SWRL: A semantic web rule language combining OWL and RuleML**. *W3C Member submission*, v. 21, n. 79, p. 1–31, 2004.

JANSEN, Slinger; FINKELSTEIN, Anthony; BRINKKEMPER, Sjaak. A sense of community: A research agenda for software ecosystems. **In 31st International Conference on Software Engineering - Companion Volume.** IEEE, 2009. p. 187–190.

KILAMO, Terhi *et al.* **From proprietary to open source - Growing an open source ecosystem**. *Journal of Systems and Software*, v. 85, n. 7, p. 1467–1478, jul. 2012.

KITCHENHAM, Barbara. **Procedures for performing systematic reviews**. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

KOCH, Stefan; KERSCHBAUM, Markus. **Joining a smartphone ecosystem: Application developers' motivations and decision criteria**. *Information and Software Technology*, v. 56, n. 11, p. 1423–1435, 2014.

KRUIZE, Jan Willem *et al.* Integrating ICT applications for farm business collaboration processes using FI space. **In 2014 Annual SRII Global Conference.** IEEE, 2014. p. 232–240.

KUDE, Thomas; DIBBERN, Jens. Tight versus Loose Organizational Coupling within Inter-Firm Networks in the Enterprise Software Industry – The Perspective of Complementors. **In *AMCIS 2009 Proceedings***, 1 jan. 2009.

KUDE, Thomas; DIBBERN, Jens; HEINZL, Armin. **Why Do Complementors Participate? An Analysis of Partnership Networks in the Enterprise Software Industry**. *IEEE Transactions on Engineering Management*, v. 59, n. 2, p. 250–265, maio 2012.

KUKKO, Marianne; HELANDER, Nina. **Knowledge sharing barriers in growing software companies**. 2012. IEEE Computer Society, 2012. p. 3756–3765.

LUNGU, Mircea; ROBBES, Romain; LANZA, Michele. Recovering inter-project dependencies in software ecosystems. **In ASE '10: Proceedings of the IEEE/ACM international conference on Automated software engineering.** 2010. p. 309–312.

MANIKAS, Konstantinos; HANSEN, Klaus Marius. **Software ecosystems – A systematic literature review.** *Journal of Systems and Software*, v. 86, n. 5, p. 1294–1306, 1 maio 2013.

MARQUES, Phillipe *et al.* **Apoiando a Composição de Serviços em um Ecossistema de Software Científico**. *Juiz de Fora, MG, Brasil: Universidade Federal de Juiz de Fora*, 2017.

MESSERSCHMITT, David G.; SZYPERSKI, Clemens. ***Software ecosystem : understanding an indispensable technology and industry***. MIT Press, 2003.

MESSERSCHMITT, David G.; SZYPERSKI, Clemens. ***Software ecosystem : understanding an indispensable technology and industry***. 2nd Editio ed. MIT Press, 2005.

MONTANDON, Joao Eduardo; SILVA, Luciana Lourdes; VALENTE, Marco Tulio. Identifying Experts in Software Libraries and Frameworks among GitHub Users. **In MSR 2019: 16th International Conference on Mining Software Repositories**. 2019.

PARR, Terence. ***The definitive ANTLR 4 reference***. The Pragmatic Programmers, 2014.

QIU, Yixin; HANN, Il-Horn; GOPAL, Anand. From invisible hand to visible hand: platform governance and institutional logic of independent Mac developers, 2001-2012. **In *ICIS 2013 Proceedings***, 16 dez. 2013.

RICKMANN, Theresa; WENZEL, Stefan; FISCHBACH, Kai. Software Ecosystem Orchestration: The Perspective of Complementors. **In *Twentieth Americas Conference on Information Systems****, Savannah*, 2014.

SADI, Mahsa H.; DAI, Jiaying; YU, Eric. **Designing software ecosystems: How to develop sustainable collaborations? scenarios from apple ios and google android.** Advanced Information Systems Engineering Workshops.. 2015. p. 161–173.

SANTOS, Rodrigo Pereira Dos; WERNER, Cláudia Maria Lima. ReuseECOS: An Approach to Support Global Software Development through Software Ecosystems.**In 2012 IEEE Seventh International Conference on Global Software Engineering Workshops.** IEEE, 2012. p. 60–65.

SCACCHI, Walt *et al.* **Understanding Free/Open Source Software Development Processes.** *Software Process: Improvement and Practice***,** v. 11, n. 2, p. 95–105, mar. 2006.

SEICHTER, Dominik *et al.* **Knowledge management in software ecosystems**. 2010. Association for Computing Machinery (ACM), 2010. p. 119.

SHULL, Forrest *et al.* **Knowledge-Sharing Issues in Experimental Software Engineering.** *Empirical Software Engineering*, v. 9, n. 1/2, p. 111–137, 2004.

SIRIN, Evren *et al.* **Pellet: A practical OWL-DL reasoner**. *Journal of Web Semantics*, v. 5, n. 2, p. 51–53, 1 jun. 2007.

SIRQUEIRA, Tássio F.M. *et al.* **E-SECO ProVersion: An Approach for Scientific Workflows Maintenance and Evolution**. 2016. Elsevier B.V., 2016. p. 547–556.

SMEDLUND, Anssi; FAGHANKHANI, Hoda. **Platform orchestration for efficiency, development, and innovation**. 26 mar. 2015. IEEE Computer Society, 26 mar. 2015. p. 1380–1388.

SUN, Xiaobing *et al.* **Personalized project recommendation on GitHub.** ***Science China Information Sciences***, Continuação do artigo Scalable Relevant Project Recommendation on GitHub, v. 61, n. 5, 20 maio 2018.

SYED, S; JANSEN, S. **On clusters in open source ecosystems**. 2013. CEUR-WS, 2013. p. 13–25.

TEIXEIRA, Jose; ROBLES, Gregorio; GONZÁLEZ-BARAHONA, Jesús M. **Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software ecosystem**. *Journal of Internet Services and Applications*, v. 6, n. 1, p. 14, 24 ago. 2015.

VALENÇA, George; ALVES, Carina. We need to discuss the relationship: An analysis of facilitators & barriers of software ecosystem partnerships. **In *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems***, v. 2, n. Iceis, p. 978–989, 2017.

VAN ANGEREN, Joey; ALVES, Carina; JANSEN, Slinger. **Can we ask you to collaborate? Analyzing app developer relationships in commercial platform ecosystems**. *Journal of Systems and Software*, v. 113, p. 430–445, mar. 2016.

VAN ANGEREN, Joey; BLIJLEVEN, Vincent; JANSEN, Slinger. Relationship intimacy in software ecosystems: A survey of the dutch software industry. **In MEDES '11: Proceedings of the International Conference on Management of Emergent Digital EcoSystems**. 2011. p. 68–75.

VAN ANGEREN, Joey; JANSEN, Slinger; BRINKKEMPER, Sjaak. Exploring the Relationship between Partnership Model Participation and Interfirm Network Structure: An Analysis of the Office365 Ecosystem. **In ICSOB 2014: Software Business. Towards Continuous Value Delivery.** 2014. p. 1–15.

VAN DER SCHUUR, Henk; JANSEN, Slinger; BRINKKEMPER, Sjaak. The power of propagation: On the role of software operation knowledge within software ecosystems. **In MEDES '11: Proceedings of the International Conference on Management of Emergent Digital EcoSystems.** 2011. p. 76–84.

VAN SOLINGEN, Rini *et al.* **Goal Question Metric (GQM) Approach.** ***Encyclopedia of Software Engineering***. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2002.

WAREHAM, Jonathan Douglas; FOX, Paul B.; CANO GINER, Josep Lluís. **Technology Ecosystem Governance**. *SSRN Electronic Journal*, 2013.

WOHLIN, C. *et al.* **Experimentation in Software Engineering**. Springer, 2012.

WÜRSCH, **Michael *et al.* SEON: a pyramid of ontologies for software evolution and its applications**. *Computing*, v. 94, n. 11, p. 857–885, nov. 2012.

APÊNDICE A - Ontologia SECOn em lógica descritiva

Classes		
Activity Activity ⊆ SeonThing Agent Artifact Artifact ⊆ SeonThing Assignee Assignee ⊆ Developer Attachment Attachment ⊆ File Branch Branch ⊆ SeonThing Bug Bug ⊆ Issue BugFix BugFix ⊆ Activity ChangeSet ChangeSet ⊆ SeonThing Comment Comment ⊆ Artifact Commit Commit ⊆ Activity Commit Committer Committer ⊆ Developer Community Community ⊆ ¬ UserAccount Community ⊆ ¬ Item Community ⊆ ¬ Role Company Company ⊆ Institution Company ⊆ ¬ FOSSFoundation Component Component ⊆ Software Consortium Consortium ⊆ Institution Container Container ⊆ ¬ Item Container ⊆ ¬ User Container ⊆ ¬ Usergroup Container ⊆ ¬ UserAccount Container ⊆ ¬ Role Developer Developer ⊆ User Developer ⊆ Individual Developer ⊆ UserAccount Developer ⊆ Individual ∩ ∃ develops Software¬ Developer ⊆ Stakeholder Directory Directory ⊆ File Document Enhancement Enhancement ⊆ Activity	FOSSFoundation ⊆ ¬ Company FeatureAddition FeatureAddition ⊆ Activity FeatureRequest FeatureRequest ⊆ Issue File File ⊆ Artifact FileUnderVersionControl FileUnderVersionControl ⊆ File Forum Forum ⊆ Container Graph Improvement Improvement ⊆ Issue Individual Individual UserAccount Individual ⊆ OnlineAccount Individual ⊆ Entity Institution Institution ⊆ OnlineAccount Institution ⊆ Entity Issue Issue ⊆ Artifact Item Item ⊆ ¬ Usergroup Item ⊆ ¬ Role Item ⊆ ¬ User Item ⊆ ¬ Community Item ⊆ ¬ Container Item ⊆ ¬ UserAccount Item ⊆ ¬ Space Milestone Milestone ⊆ SeonThing OnlineAccount OnlineAccount Entity Platform Platform ⊆ Software Post Post ⊆ Document Post ⊆ Item Priority Priority ⊆ SeonThing Product Product ⊆ OnlineAccount Product ⊆ Entity Product ⊆ SeonThing Release Release ⊆ SeonThing Reporter Reporter ⊆ Stakeholder Repository Resolution	Role ⊆ ¬ Space Role ⊆ ¬ Usergroup Role ⊆ ¬ Container Role ⊆ ¬ User SeonThing Severity Severity ⊆ SeonThing Site Site ⊆ Space SocialNetwork Software Software ⊆ Product SoftwareEcosystem Space Space ⊆ ¬ User Space ⊆ ¬ Usergroup Space ⊆ ¬ UserAccount Space ⊆ ¬ Role Space ⊆ ¬ Item Stakeholder Stakeholder ⊆ SeonThing Status Status ⊆ SeonThing Thing Thread Thread ⊆ Container Topic User User UserAccount User ⊆ OnlineAccount User ⊆ Entity User ⊆ ¬ Space User ⊆ ¬ Item User ⊆ ¬ Usergroup User ⊆ ¬ Container User ⊆ ¬ Role UserAccount UserAccount User UserAccount Individual UserAccount ⊆ OnlineAccount UserAccount ⊆ Entity UserAccount ⊆ ¬ Community UserAccount ⊆ ¬ Usergroup UserAccount ⊆ ¬ Role UserAccount ⊆ ¬ Space UserAccount ⊆ ¬ Item UserAccount ⊆ ¬ Container Usergroup Usergroup ⊆ ¬ Item Usergroup ⊆ ¬ Space Usergroup ⊆ ¬ User Usergroup ⊆ ¬ UserAccount Usergroup ⊆ ¬ Container Usergroup ⊆ ¬ Role

Entity Entity OnlineAccount FOSSFoundation FOSSFoundation ⊆ Institution	Resolution ⊆ SeonThing Role Role ⊆ ¬ Item Role ⊆ ¬ UserAccount Role ⊆ ¬ Community	Version Version ⊆ SeonThing
Object properties		
about ∃ about Thing ⊆ Item account account of account⁻ account of account of account⁻ ∃ account of Thing ⊆ UserAccount T ⊆ ∀ account of Agent addressed to ∃ addressed to Thing ⊆ Item administrator of administrator of has administrator⁻ ∃ administrator of Thing ⊆ UserAccount T ⊆ ∀ administrator of Site appearsInRelease ∃ appearsInRelease Thing ⊆ Version T ⊆ ∀ appearsInRelease Release attachment ∃ attachment Thing ⊆ Item avatar ∩depiction ∃ avatar Thing ⊆ UserAccount belongsToRelease belongsToRelease hasMilestone⁻ ∃ belongsToRelease Thing ⊆ Milestone T ⊆ ∀ belongsToRelease Release blocksIssue ∩dependsOnIssue⁻ blocksIssue isBlockedBy⁻ ∃ blocksIssue Thing ⊆ Issue T ⊆ ∀ blocksIssue Issue carriesOutActivity carriesOutActivity isCarriedOutBy⁻ ∃ carriesOutActivity Thing ⊆ Stakeholder T ⊆ ∀ carriesOutActivity Activity collaboresWith ∃ collaboresWith Thing ⊆ Entity T ⊆ ∀ collaboresWith Entity commentsIssue	hasDuplicate hasDuplicate⁻ ∃ hasDuplicate Thing ⊆ Issue T ⊆ ∀ hasDuplicate Issue hasMessage ∃ hasMessage Thing ⊆ Commit T ⊆ ∀ hasMessage Post hasMilestone belongsToRelease hasMilestone⁻ ∃ hasMilestone Thing ⊆ Release T ⊆ ∀ hasMilestone Milestone hasParent hasChild hasParent⁻ ∃ hasParent Thing ⊆ SeonThing T ⊆ ∀ hasParent SeonThing hasPart hasPriority ∃ hasPriority Thing ⊆ Issue T ⊆ ∀ hasPriority Priority hasRelease hasRelease isReleaseOf⁻ ∃ hasRelease Thing ⊆ Product T ⊆ ∀ hasRelease Release hasReporter hasReporter isReporterOf⁻ ∃ hasReporter Thing ⊆ Issue T ⊆ ∀ hasReporter Reporter hasResolution ∃ hasResolution Thing ⊆ Issue T ⊆ ∀ hasResolution Resolution hasSeverity ∃ hasSeverity Thing ⊆ Issue T ⊆ ∀ hasSeverity Severity hasSibling hasSibling hasSibling⁻ ∃ hasSibling Thing ⊆ SeonThing T ⊆ ∀ hasSibling SeonThing hasSocialNetwork ∃ hasSocialNetwork Thing ⊆ SoftwareEcosystem T ⊆ ∀ hasSocialNetwork SocialNetwork hasStatus ∃ hasStatus Thing ⊆ Issue	isCommentedBy ∩hasAuthor commentsIssue isCommentedBy⁻ ∃ isCommentedBy Thing ⊆ Issue T ⊆ ∀ isCommentedBy Stakeholder isCommitedTo ∃ isCommitedTo Thing ⊆ Commit T ⊆ ∀ isCommitedTo Repository isCommittedBy commitsVersion isCommittedBy⁻ ∃ isCommittedBy Thing ⊆ Version T ⊆ ∀ isCommittedBy Committer isCommittedIn containsVersion isCommittedIn⁻ ∃ isCommittedIn Thing ⊆ Version T ⊆ ∀ isCommittedIn ChangeSet isDevelopedBy develops isDevelopedBy⁻ isOnBranch ∃ isOnBranch Thing ⊆ Version T ⊆ ∀ isOnBranch Branch isPartOfConsortium ∃ isPartOfConsortium Thing ⊆ Company T ⊆ ∀ isPartOfConsortium Consortium isPartOfEcosystem T ⊆ ∀ isPartOfEcosystem SoftwareEcosystem isPartOfSocialNetwork ∃ isPartOfSocialNetwork Thing ⊆ Entity t Software T ⊆ ∀ isPartOfSocialNetwork SocialNetwork isReleaseOf hasRelease isReleaseOf⁻ ∃ isReleaseOf Thing ⊆ Release T ⊆ ∀ isReleaseOf Product isReporterOf hasReporter isReporterOf⁻ ∃ isReporterOf Thing ⊆ Reporter T ⊆ ∀ isReporterOf Issue isRequiredBy isRequiredBy requiresKnowledge⁻ isSimilar isSimilar isSimilar⁻ ∃ isSimilar Thing ⊆ SeonThing T ⊆ ∀ isSimilar SeonThing isVersionOf hasVersion isVersionOf⁻ ∃ isVersionOf Thing ⊆ Version

∩hasAuthor⁻
commentsIssue
isCommentedBy⁻
∃ commentsIssue Thing ⊆ Stakeholder
T ⊆ ∀ commentsIssue Issue
commitsChangeSet
∃ commitsChangeSet Thing ⊆ Committer
T ⊆ ∀ commitsChangeSet ChangeSet
commitsVersion
commitsVersion isCommittedBy⁻
∃ commitsVersion Thing ⊆ Committer
T ⊆ ∀ commitsVersion Version
complements
∩dependsOn⁻
complements dependsOn⁻
∃ complements Thing ⊆ Component
T ⊆ ∀ complements Product
composes
∃ composes Thing ⊆ Artifact
T ⊆ ∀ composes Product
constituesVersion
∃ constituesVersion Thing ⊆ Commit
T ⊆ ∀ constituesVersion Version
container of
∩hasPart
∩partOf⁻
container of has container⁻
∃ container of Thing ⊆ Container
T ⊆ ∀ container of Item
containsFile
∃ containsFile Thing ⊆ Directory
T ⊆ ∀ containsFile File
containsVersion
containsVersion isCommittedIn⁻
∃ containsVersion Thing ⊆ ChangeSet
T ⊆ ∀ containsVersion Version
creator of
creator of has creator⁻
∃ creator of Thing ⊆ UserAccount
delivered at
∃ delivered at Thing ⊆ Item
dependsOn
∃ dependsOn Thing ⊆ SeonThing
T ⊆ ∀ hasStatus Status
hasUsed
∃ hasUsed Thing ⊆ Entity
T ⊆ ∀ hasUsed Product
hasVersion
hasVersion isVersionOf⁻
∃ hasVersion Thing ⊆ FileUnderVersionControl
T ⊆ ∀ hasVersion Version
has administrator
administrator of has administrator⁻
∃ has administrator Thing ⊆ Site
T ⊆ ∀ has administrator UserAccount
has container
∩hasPart⁻
∩partOf
container of has container⁻
∃ has container Thing ⊆ Item
T ⊆ ∀ has container Container
has creator
creator of has creator⁻
T ⊆ ∀ has creator UserAccount
has discussion
discussion of has discussion⁻
∃ has discussion Thing ⊆ Item
has function
function of has function⁻
T ⊆ ∀ has function Role
has group
group of has group⁻
has host
∩has space
∩space of⁻
has host host of⁻
∃ has host Thing ⊆ Container
T ⊆ ∀ has host Site
has member
has member member of⁻
∃ has member Thing ⊆ Usergroup
T ⊆ ∀ has member UserAccount
has moderator
has moderator moderator of⁻
∃ has moderator Thing ⊆ Forum
T ⊆ ∀ has moderator UserAccount
has modier
has modier modier of⁻
T ⊆ ∀ has modier UserAccount
has owner
has owner owner of⁻
T ⊆ ∀ has owner UserAccount
T ⊆ ∀ isVersionOf FileUnderVersionControl
later version
earlier version later version⁻
TransitivePropertylater version
∃ later version Thing ⊆ Item
T ⊆ ∀ later version Item
latest version
∃ latest version Thing ⊆ Item
T ⊆ ∀ latest version Item
links to
∩references
mayCollaborateWith
member of
has member member of⁻
∃ member of Thing ⊆ UserAccount
T ⊆ ∀ member of Usergroup
mentions
∃ mentions Thing ⊆ Item
T ⊆ ∀ mentions UserAccount
moderator of
has moderator moderator of⁻
∃ moderator of Thing ⊆ UserAccount
T ⊆ ∀ moderator of Forum
modier of
has modier modier of⁻
∃ modier of Thing ⊆ UserAccount
next by date
next by date previous by date⁻
∃ next by date Thing ⊆ Item
T ⊆ ∀ next by date Item
next version
∩later version
∩earlier version⁻
next version previous version⁻
∃ next version Thing ⊆ Item
T ⊆ ∀ next version Item
owner of
has owner owner of⁻
∃ owner of Thing ⊆ UserAccount
owns
∃ owns Thing ⊆ Entity
T ⊆ ∀ owns (Product t Repository)
parent of
∩partOf⁻
∩hasPart
has parent parent of⁻
∃ parent of Thing ⊆ Container
T ⊆ ∀ parent of Container
partOf
part of
has part part of⁻
performsCommit
∃ performsCommit Thing ⊆ Committer
T ⊆ ∀ performsCommit Commit
precedesVersion
followsVersion precedesVersion⁻
∃ precedesVersion Thing ⊆ Version

T ⊆ ∀ dependsOn SeonThing
dependsOn
∩dependsOn
complements dependsOn⁻
dependsOnIssue
∩dependsOn
∃ dependsOnIssue Thing ⊆ Issue
T ⊆ ∀ dependsOnIssue Issue
depiction
develops
develops isDevelopedBy⁻
∃ develops Thing ⊆ Entity
T ⊆ ∀ develops Product
discussion of
discussion of has discussion⁻
T ⊆ ∀ discussion of Item
earlier version
earlier version later version⁻
TransitivePropertyearlier version
∃ earlier version Thing ⊆ Item
T ⊆ ∀ earlier version Item
email
∃ email Thing ⊆ UserAccount
embeds knowledge
∃ embeds knowledge Thing ⊆ Item
T ⊆ ∀ embeds knowledge Graph
follows
follows follows
∃ follows Thing ⊆ UserAccount
T ⊆ ∀ follows UserAccount
follows
follows follows
∩follows
followsVersion
followsVersion precedesVersion⁻
∃ followsVersion Thing ⊆ Version
T ⊆ ∀ followsVersion Version
function of
function of has function⁻
∃ function of Thing ⊆ Role
generator
∃ generator Thing ⊆ Item
group of
group of has group⁻
hasAssignee
hasAssignee isAssigneeOf⁻
∃ hasAssignee Thing ⊆ Issue
T ⊆ ∀ hasAssignee Assignee
hasAttachment
hasAttachment isAttachementOf⁻
∃ hasAttachment Thing ⊆ has parent
∩hasPart⁻
∩partOf
has parent parent of⁻
∃ has parent Thing ⊆ Container
T ⊆ ∀ has parent Container
has part
has part part of⁻
has reply
∩related to⁻
∩related to
has reply reply of⁻
∃ has reply Thing ⊆ Item
T ⊆ ∀ has reply Item
has scope
has scope scope of⁻
∃ has scope Thing ⊆ Role
has space
∩partOf
∩hasPart⁻
has space space of⁻
T ⊆ ∀ has space Space
has subscriber
has subscriber subscriber of⁻
∃ has subscriber Thing ⊆ Container
T ⊆ ∀ has subscriber UserAccount
has usergroup
has usergroup usergroup of⁻
∃ has usergroup Thing ⊆ Space
T ⊆ ∀ has usergroup Usergroup
host of
∩has space⁻
∩space of
has host host of⁻
∃ host of Thing ⊆ Site
T ⊆ ∀ host of Container
hosts
∃ hosts Thing ⊆ Repository
T ⊆ ∀ hosts Product
isAcquaintedWith
isAssigneeOf
hasAssignee isAssigneeOf⁻
∃ isAssigneeOf Thing ⊆ Assignee
T ⊆ ∀ isAssigneeOf Issue
isAttachementOf
hasAttachment isAttachementOf⁻
∃ isAttachementOf Thing ⊆ Attachment
T ⊆ ∀ isAttachementOf Issue
isBasedOn
∃ isBasedOn Thing ⊆ Artifact
T ⊆ ∀ isBasedOn Artifact
T ⊆ ∀ precedesVersion Version
previous by date
next by date previous by date⁻
∃ previous by date Thing ⊆ Item
T ⊆ ∀ previous by date Item
previous version
∩later version⁻
∩earlier version
next version previous version⁻
∃ previous version Thing ⊆ Item
T ⊆ ∀ previous version Item
read at
∃ read at Thing ⊆ Item
reference
∃ reference Thing ⊆ Post
references
related to
reply of
∩related to⁻
∩related to
has reply reply of⁻
∃ reply of Thing ⊆ Item
T ⊆ ∀ reply of Item
requiresKnowledge
isRequiredBy requiresKnowledge⁻
∃ requiresKnowledge Thing ⊆ Product
T ⊆ ∀ requiresKnowledge Topic
respond to
∃ respond to Thing ⊆ Item
scope of
has scope scope of⁻
T ⊆ ∀ scope of Role
shared by
∃ shared by Thing ⊆ Item
T ⊆ ∀ shared by UserAccount
sibling
sibling sibling⁻
∃ sibling Thing ⊆ Item
T ⊆ ∀ sibling Item
space of
∩partOf⁻
∩hasPart
has space space of⁻
∃ space of Thing ⊆ Space
subject
subscriber of
has subscriber subscriber of⁻
∃ subscriber of Thing ⊆ UserAccount
T ⊆ ∀ subscriber of Container
topic
∩subject
usergroup of
has usergroup usergroup of⁻
∃ usergroup of Thing ⊆ Usergroup
T ⊆ ∀ usergroup of Space
worksIn
∃ worksIn Thing ⊆ Individual

Issue T ⊆ ∀ hasAttachment Attachment hasAuthor ∃ hasAuthor Thing ⊆ Artifact T ⊆ ∀ hasAuthor Stakeholder hasChild hasChild hasParent⁻ ∃ hasChild Thing ⊆ SeonThing T ⊆ ∀ hasChild SeonThing hasClone hasComment hasComment isCommentOf⁻ ∃ hasComment Thing ⊆ Issue T ⊆ ∀ hasComment Comment hasDuplicate ∩hasClone⁻ ∩hasClone	isBlockedBy ∩dependsOnIssue blocksIssue isBlockedBy⁻ ∃ isBlockedBy Thing ⊆ Issue T ⊆ ∀ isBlockedBy Issue isCarriedOutBy carriesOutActivity isCarriedOutBy⁻ ∃ isCarriedOutBy Thing ⊆ Activity T ⊆ ∀ isCarriedOutBy Stakeholder isCommentOf hasComment isCommentOf⁻ ∃ isCommentOf Thing ⊆ Comment T ⊆ ∀ isCommentOf Issue	T ⊆ ∀ worksIn Institution belongsToEcosystem ∃ isRequiredBy Software ⊆ belongsToEcosystem belongsToEcosystem ∃ dependsOn Software ⊆ belongsToEcosystem mayBeAcquaintedWith ∃ develops Software ⊆ requiresKnowledge mayCollaborateWith ∃ isAcquaintedWith Topic ⊆ isRequiredBy
Data properties		
hasCity ⊆ hasLocation T ⊆ ∀ hasCity Datatype http://www.w3.org/2001/XMLSchemastring **hasCountry** ⊆ hasLocation T ⊆ ∀ hasCountry Datatype http://www.w3.org/2001/XMLSchemastring **hasDescription** T ⊆ ∀ hasDescription Datatype http://www.w3.org/2001/XMLSchemastring	**hasEmail** ∃ hasEmail Datatype http://www.w3.org/2000/01/rdf-schemaLiteral ⊆ Stakeholder **hasId** T ⊆ ∀ hasId Datatype http://www.w3.org/2001/XMLSchemastring **hasLocation** T ⊆ ∀ hasLocation Datatype http://www.w3.org/2001/XMLSchemastring	**hasLogin** T ⊆ ∀ hasLocation Datatype http://www.w3.org/2001/XMLSchemastring **hasName** T ⊆ ∀ hasName Datatype http://www.w3.org/2001/XMLSchemastring **hasURL** T ⊆ ∀ hasURL Datatypehttp://www.w3.org/2001/XMLSchemastring

ANEXO A – Referências do Mapeamento Sistemático

BOUCHARAS, Vasilis; JANSEN, Slinger; BRINKKEMPER, Sjaak. Formalizing software ecosystem modeling. **In IWOCE '09: Proceedings of the 1st international workshop on Open component ecosystems**. 2009. 2009. p. 41–50.

COUTINHO, Emanuel Ferreira; SANTOS, Italo; BEZERRA, Carla Ilane Moreira. **A Software Ecosystem for a Virtual Learning Environment: SOLAR SECO**. 28 jun. 2017. Institute of Electrical and Electronics Engineers Inc., 28 jun. 2017. p. 41–47.

CROOYMANS, Wesley; PRADHAN, Priyanka; JANSEN, Slinger. **Exploring network modelling and strategy in the Dutch software business ecosystem**. 2015. Springer Verlag, 2015. p. 45–59.

FERDMAN, Sela *et al.* **Quantifying the web browser ecosystem**. *PLOS ONE*, v. 12, n. 6, p. e0179281, 23 jun. 2017.

FRANCO-BEDOYA, Oscar *et al.* **Open source software ecosystems: A Systematic mapping**. *Information and Software Technology*, v. 91, p. 160–185, 1 nov. 2017.

GALATEANU, Elena; AVASILCAI, Silvia. **Framing the Competitive Behaviors of Niche Players: The Electric Vehicle Business Ecosystem Perspective**. *Procedia - Social and Behavioral Sciences*, v. 221, p. 342–351, jun. 2016.

KILAMO, Terhi *et al.* **From proprietary to open source - Growing an open source ecosystem.** *Journal of Systems and Software*, v. 85, n. 7, p. 1467–1478, jul. 2012.

KOCH, Stefan; KERSCHBAUM, Markus. **Joining a smartphone ecosystem: Application developers' motivations and decision criteria.** *Information and Software Technology*, v. 56, n. 11, p. 1423–1435, 2014.

KRUIZE, Jan Willem *et al.* Integrating ICT applications for farm business collaboration processes using FI space. **In 2014 Annual SRII Global Conference.** IEEE, 2014. p. 232–240.

KUDE, Thomas; DIBBERN, Jens. Tight versus Loose Organizational Coupling within Inter-Firm Networks in the Enterprise Software Industry – The Perspective of Complementors. **In *AMCIS 2009 Proceedings***, 1 jan. 2009.

KUDE, Thomas; DIBBERN, Jens; HEINZL, Armin. **Why Do Complementors Participate? An Analysis of Partnership Networks in the Enterprise Software Industry**. *IEEE Transactions on Engineering Management*, v. 59, n. 2, p. 250–265, maio 2012.

KUKKO, Marianne; HELANDER, Nina. **Knowledge sharing barriers in growing software companies**. 2012. IEEE Computer Society, 2012. p. 3756–3765.

QIU, Yixin; HANN, Il-Horn; GOPAL, Anand. From invisible hand to visible hand: platform governance and institutional logic of independent Mac developers, 2001-2012. **In *ICIS 2013 Proceedings***, 16 dez. 2013.

RICKMANN, Theresa; WENZEL, Stefan; FISCHBACH, Kai. Software Ecosystem Orchestration: The Perspective of Complementors. **In *Twentieth Americas Conference on Information Systems**, Savannah*, 2014.

SADI, Mahsa H.; DAI, Jiaying; YU, Eric. **Designing software ecosystems: How to develop sustainable collaborations? scenarios from apple ios and google android.** Advanced Information Systems Engineering Workshops.. 2015. p. 161–173.

SCACCHI, Walt *et al.* **Understanding Free/Open Source Software Development Processes.** *Software Process: Improvement and Practice***,** v. 11, n. 2, p. 95–105, mar. 2006.

SEICHTER, Dominik *et al.* **Knowledge management in software ecosystems**. 2010. Association for Computing Machinery (ACM), 2010. p. 119.

SMEDLUND, Anssi; FAGHANKHANI, Hoda. **Platform orchestration for efficiency, development, and innovation**. 26 mar. 2015. IEEE Computer Society, 26 mar. 2015. p. 1380–1388.

SYED, S; JANSEN, S. **On clusters in open source ecosystems**. 2013. CEUR-WS, 2013. p. 13–25.

TEIXEIRA, Jose; ROBLES, Gregorio; GONZÁLEZ-BARAHONA, Jesús M. **Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software ecosystem**. *Journal of Internet Services and Applications*, v. 6, n. 1, p. 14, 24 ago. 2015.

VALENÇA, George; ALVES, Carina. We need to discuss the relationship: An analysis of facilitators & barriers of software ecosystem partnerships. **In *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems***, v. 2, n. Iceis, p. 978–989, 2017.

VAN ANGEREN, Joey; ALVES, Carina; JANSEN, Slinger. **Can we ask you to collaborate? Analyzing app developer relationships in commercial platform ecosystems**. *Journal of Systems and Software*, v. 113, p. 430–445, mar. 2016.

VAN ANGEREN, Joey; BLIJLEVEN, Vincent; JANSEN, Slinger. Relationship intimacy in software ecosystems: A survey of the dutch software industry. **In MEDES '11: Proceedings of the International Conference on Management of Emergent Digital EcoSystems**. 2011. p. 68–75.

VAN ANGEREN, Joey; JANSEN, Slinger; BRINKKEMPER, Sjaak. Exploring the Relationship between Partnership Model Participation and Interfirm Network Structure: An Analysis of the Office365 Ecosystem. **In ICSOB 2014: Software Business. Towards Continuous Value Delivery.** 2014. p. 1–15.

VAN DER SCHUUR, Henk; JANSEN, Slinger; BRINKKEMPER, Sjaak. The power of propagation: On the role of software operation knowledge within software ecosystems. **In MEDES '11: Proceedings of the International Conference on Management of Emergent Digital EcoSystems.** 2011. p. 76–84.

WAREHAM, Jonathan Douglas; FOX, Paul B.; CANO GINER, Josep Lluís. **Technology Ecosystem Governance**. *SSRN Electronic Journal*, 2013.