UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Gabriel Gonçalves Nogueira

Um Repositório para Reúso de Ontologias Orientado a Objetivos

Projeto de Graduação apresentado ao Departamento de Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Monalessa Perini Barcellos
Coorientador: Cássio Chaves Reginato

VITÓRIA
2019

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Gabriel Gonçalves Nogueira

Um Repositório para Reúso de Ontologias Orientado a Objetivos

COMISSÃO EXAMINADORA

_____
Profa. Monalessa Perini Barcellos, D. Sc.

_____
Prof. Vítor Estêvão Silva Souza, Ph.D.…….

_____
Profa. Jordana Sarmenghi Salamon, M. Sc.…….

Vitória, 19 de Julho de 2019

"O hoje é o ontem de amanhã,
mas amanhã o que será?
digo isso a cada manhã
e o amanhã responderá"

(Nelson Sylvan)

AGRADECIMENTOS

Agradeço a Deus por ter me guiado por todo o caminho percorrido e me auxiliado nos momentos difíceis.

Também agradeço a minha família, por ter me criado e me ajudado a superar todas as adversidades e barreiras encontradas durante a vida.

Aos meus amigos de curso, que conviveram comigo durante todos esses anos. Agradeço pelas horas na cantina do CT, conversando sobre os assuntos mais aleatórios possíveis.

Agradeço aos professores do Curso de Ciência da Computação da UFES, pelo conhecimento compartilhado, em especial minha orientadora Monalessa.

Agradeço também ao Cássio pelas orientações e pela oportunidade de trabalhar no projeto deu origem ao GoopHub.

E, especialmente, a Renata, por ter me acompanhando durante grande parte da minha vida acadêmica e por sempre me motivar a crescer e evoluir.

RESUMO

Ontologias têm sido usadas para atribuir semântica aos dados na área de Web Semântica e para apoiar a integração de dados de diferentes sistemas e fontes. Entretanto, desenvolver uma ontologia é uma tarefa difícil. Dessa forma, o reúso de ontologias pode contribuir positivamente nesse contexto.

A reutilização de ontologias proporciona maior dinamismo ao processo de desenvolvimento da ontologia, diminuindo o custo, utilizando menos tempo e recursos, além de estimular o uso de boas práticas. Para favorecer o reúso, o uso de padrões (*patterns*) tem se mostrado uma abordagem promissora na Engenharia de Ontologias. Um padrão pode ser definido como uma solução bem-sucedida para um problema recorrente.

Um dos desafios do reúso de ontologias é a obscuridade por trás do *design rationale* das ontologias disponíveis. Engenharia de Requisitos Orientada a Objetivos (GORE – *Goal Oriented Requirements Engineering*) tem sido utilizada em algumas iniciativas no âmbito da Engenharia de Ontologias para enriquecer o levantamento de requisitos, auxiliando na definição do escopo e representação do *design rationale* da ontologia.

Neste trabalho explora-se o uso combinado de padrões ontológicos e GORE para apoiar reúso de ontologias. Para isso, foi desenvolvido *GoopHub*, um repositório que permite o armazenamento de fragmentos de ontologias na forma de padrões ontológicos associados a objetivos e a utilização desses padrões para o desenvolvimento de novas ontologias considerando-se os objetivos que a ontologia deve atender.

Palavras-chave: Ontologia, Reúso, Modelagem de Objetivos, GORE, *Pattern*

SUMÁRIO

Capítulo 1
Introdução

Este capítulo apresenta uma breve introdução ao tema do trabalho, seus objetivos, histórico do desenvolvimento e a organização deste documento.

1.1 Introdução

Ontologia pode ser definida como uma descrição formal de um domínio. Tal descrição pode ser compartilhada entre diferentes aplicações (Noy, 2004). Para Guarino (1997), uma ontologia é um artefato, constituído por um vocabulário próprio usado para descrever uma determinada realidade, contendo, ainda, uma série de inferências nítidas acerca do significado das palavras deste vocabulário. Nos dias de hoje, engenheiros de ontologias são auxiliados por diversos métodos e ferramentas na área de Engenharia de Ontologias. Entretanto, desenvolver novas ontologias ainda tem se mostrado uma tarefa difícil e trabalhosa (Falbo *et al.*, 2013).

Reúso de ontologias é uma questão de pesquisa complexa e uma das áreas mais desafiadoras e negligenciadas da Engenharia de Ontologias (Gangemi e Presutti, 2009). Embora existam diversas ferramentas para auxiliar nas atividades de Engenharia de Ontologias, muitas dificuldades ainda persistem. Uma prática que pode contribuir no desenvolvimento de ontologias é o reúso, tendo em vista que fragmentos já desenvolvidos previamente podem ser reutilizados na construção de novas ontologias (Falbo *et al.*, 2013). A reutilização de ontologias proporciona maior dinamismo ao processo de desenvolvimento da ontologia, diminuindo o custo, utilizando menos tempo e recursos, além de estimular o uso de boas práticas (Poveda-Villalón, Carmen Suárez-Figueroa e Gómez-Pérez, 2010). Ainda assim, engenheiros de ontologias se deparam com problemas para selecionar as ontologias mais adequadas para reúso e para integrar esses fragmentos em uma nova ontologia (Park, Oh e Ahn, 2011).

O uso de padrões (*patterns*) tem se mostrado uma abordagem promissora no contexto de reúso de ontologias. Padrões são veículos para encapsular conhecimento. Um padrão ontológico (*ontology pattern* - OP) descreve um problema de modelagem recorrente que emerge a partir do desenvolvimento de ontologias em contextos específicos e apresenta uma solução para ele. Assim, em Engenharia de Ontologias, eles se referem a fragmentos que funcionam como blocos de construção que podem ser combinados para tratar os problemas relevantes para a ontologia a ser desenvolvida (Falbo *et al.*, 2013).

Reúso de ontologias envolve selecionar ontologias (ontologias fonte) a serem reutilizadas e integrá-las para que se obtenha uma nova ontologia (ontologia resultante) (López de Vergara *et al.*, 2003). Um dos desafios no que tange reúso de ontologias é a obscuridade por trás do *design rationale* das ontologias disponíveis (Gangemi e Presutti, 2009). Com isso, torna-se difícil a seleção das ontologias a serem integradas, bem como entendê-las, o que é um passo crucial para que a integração ocorra de forma correta.

Segundo van Lamsweerde (2002), a utilização da Engenharia de Requisitos Orientada a Objetivos (*Goal-Oriented Requirements Engineering* - GORE) preocupa-se com o uso de objetivos para elicitação, elaboração, estruturação, especificação, análise, negociação, documentação e modificação de requisitos. O autor defende que os objetivos fornecem um *design rationale* para os requisitos e o refinamento destes proporciona uma estrutura simples para documentação de requisitos.

GORE tem sido utilizada em algumas iniciativas em Engenharia de Ontologias. Fernandes, Guizzardi e Guizzardi (2011) aplicaram as técnicas de GORE com sucesso para enriquecer o levantamento de requisitos no processo de engenharia de ontologias, utilizando modelagem de objetivos como recurso para a definição do escopo da ontologia. Salamon *et al.* (2017) aplicaram modelagem de objetivos na reengenharia de uma ontologia para o domínio de e-commerce. Por fim, em Salamon (2018) foi proposta uma abordagem de desenvolvimento de ontologias baseado em integração e orientado a objetivos.

Este projeto de graduação foi desenvolvido no contexto de um projeto de pesquisa mais amplo, no qual vem sendo explorado o uso combinado de padrões ontológicos e GORE para apoiar reúso de ontologias (Salamon, 2018) e (Reginato *et al.*, 2019). No escopo deste projeto de graduação, foi desenvolvido *GoopHub*, um repositório que provê apoio computacional para GO-FOR (*Goal-Oriented Framework for Ontology Reuse*), proposto no contexto do referido projeto de pesquisa (Reginato *et al.*, 2019). *GoopHub* permite o armazenamento de fragmentos de ontologias na forma de padrões ontológicos associados a objetivos (chamados *Goal-Oriented Ontology Patterns* – GOOPs) e a utilização desses padrões para o desenvolvimento de novas ontologias considerando-se os objetivos que a ontologia deve atender.

1.2 Objetivos

O objetivo deste projeto de graduação é *desenvolver uma solução computacional que permita o armazenamento de padrões ontológicos associados a objetivos e a seleção desses padrões para o desenvolvimento de novas ontologias considerando-se os objetivos que a ontologia deve atender.* São objetivos específicos deste trabalho:

i. Analisar o problema e projetar uma solução para a criação, armazenamento e recuperação de GOOPs;

ii. Implementar a solução projetada;

iii. Definir um conjunto de GOOPs e disponibilizá-los para reúso;

iv. Aplicar a solução implementada para desenvolvimento de ontologias com reúso.

1.3 Histórico de Desenvolvimento do Trabalho

Para atingir o objetivo apresentado na seção anterior, as seguintes atividades foram realizadas:

(i) *Revisão Bibliográfica:* O trabalho teve início com uma revisão bibliográfica sobre Ontologias, Reúso de Ontologias, Padrões Ontológicos e GORE.

(ii) *Estudo de Tecnologias:* Nesta atividade foi realizado o estudo de tecnologias relevantes para projeto, destacando-se: linguagem de programação Java; linguagem de programação Web Javascript; ambiente de desenvolvimento *IntelliJ Community*; *Spring Boot* (*framework* para desenvolvimento Web utilizando Java)*;* banco de dados *Stardog* (*Knowledge Graph*), linguagem de modelagem de objetivos *i** (GORE *framework*) e editor de ontologias *Protégé*.

(iii) *Especificação e Análise de Requisitos e Projeto da Solução:* Nesta atividade foram especificadas as funcionalidades que a solução computacional deveria prover, foi elaborado seu modelo estrutural e a arquitetura da solução foi definida.

(iv) *Implementação e Testes do Framework:* Consistiu na implementação da solução computacional e realização de testes (tanto ao longo do desenvolvimento quanto após a conclusão da implementação).

(v) *Elaboração de Artigos e da Monografia:* Consistiu na elaboração de um artigo científico em conjunto com os demais membros do projeto de pesquisa no qual este trabalho está inserido (Reginato *et al.*, 2019) e na escrita desta monografia.

1.4 Organização do Texto:

Esta monografia é organizada em quatro capítulos e contém, além deste capítulo de introdução, os seguintes capítulos.

- **Capítulo 2** – Fundamentação Teórica: Apresenta uma revisão da literatura acerca dos temas relevantes ao contexto deste trabalho, a saber: Ontologias, Reúso de Ontologias, Padrões Ontológicos, GORE, GO-FOR e OWL;
- **Capítulo 3** – A Ferramenta *GoopHub:* Trata de aspectos relacionados ao desenvolvimento e uso do repositório, apresentando artefatos produzidos ao longo do desenvolvimento de *GoopHub*, tais como diagrama de casos de uso, modelo estrutural e questões relacionadas ao projeto e implementação de *GoopHub*. Apresenta, também, o desenvolvimento de uma ontologia utilizando GO-FOR **(***Goal-Oriented Framework for Ontology Reuse),* que é o *framework* apoiado por *GoopHub*.
- **Capítulo 4** – Considerações Finais: Apresenta a conclusão do trabalho, as dificuldades encontradas, limitações e propostas para trabalhos futuros.

Capítulo 2
Fundamentação Teórica

Este capítulo apresenta os principais aspectos teóricos que fundamentam este trabalho. Ele está organizado em 5 seções, a saber: a Seção 2.1 aborda ontologias; a Seção 2.2 fala sobre reúso de ontologias; a Seção 2.3 introduz o conceito de GORE; a Seção 2.4 apresenta GO-FOR; e, por fim, na Seção 2.5 é apresentado OWL.

2.1 Ontologias

Uma ontologia é uma especificação explícita de uma conceituação compartilhada. O termo é emprestado da Filosofia, onde uma ontologia pode ser vista como o estudo da organização e da natureza do mundo, independentemente da forma de nosso conhecimento sobre ele. Conceituação se refere a um modelo abstrato e simplificado de uma realidade que identifica seus conceitos mais relevantes. Toda base de conhecimento, sistema baseado em conhecimento ou agente de nível de conhecimento é vinculado com alguma conceituação, explícita ou implícita (Gruber, 1993).

Nas sub-áreas da Ciência da Computação, como Engenharia de Software e Web Semântica, ontologias são um tipo de artefato computacional utilizado para modelar formalmente a estrutura de um sistema de informação. Conceitos, relações e axiomas previamente definidos e fundamentados são usados para descrever um modelo de domínio uniforme e não ambíguo de entidades e suas relações, fornecendo assim uma conceituação sobre o domínio modelado (Falbo, Guizzardi e Duarte, 2004).

Guarino (1998) sugere uma classificação de ontologias definindo quatro classes de ontologias de acordo com seu grau de generalidade, como mostra a Figura 2.1:

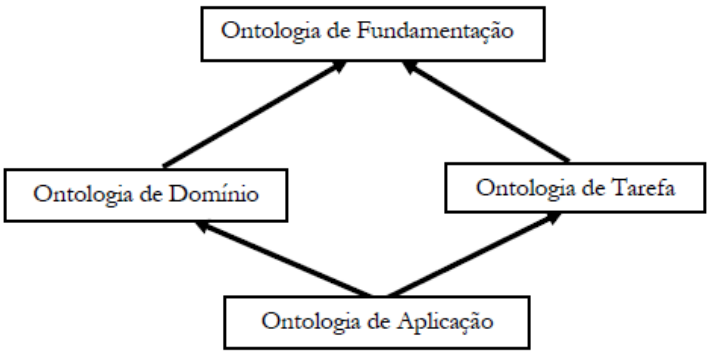


Figura 2.1 -Tipos de ontologias de acordo com o grau de generalidade (Guarino, 1998).

i. Ontologias de Fundamentação: Conhecidas também como ontologias de alto nível. Descrevem conceitos genéricos como espaço, tempo, objetos, eventos, etc. Servem como base para as ontologias de domínio e tarefas.

ii. Ontologias de Domínio: Descrevem o vocabulário referente a um domínio específico, como engenharia de software, medicina ou automóveis. Utilizam especializações de termos introduzidos na ontologia de nível superior.

iii. Ontologia de Tarefas: Descrevem tarefas ou atividades genéricas, como compra e venda. Assim como ontologias de domínio, seus conceitos são especializados de entidades existentes em ontologias de fundamentação.

iv. Ontologias de Aplicação: Descrevem conceitos ao mesmo tempo dependentes de um domínio particular e uma tarefa, as quais são frequentemente especializados de ambas ontologias relacionadas.

Scherp *et al.* (2011) também classificam ontologias de acordo com sua generalidade, porém acrescentando um nível entre ontologias de fundamentação e ontologias de domínio, denominado ontologia de núcleo (*core ontology*). Uma ontologia de núcleo fornece uma definição precisa do conhecimento estrutural em uma área específica que cobre diferentes domínios de aplicação. São construídas baseadas em ontologias de fundamentação e representam um refinamento dessas, adicionando conceitos e relações específicos da área considerada (Scherp *et al.*, 2011).

Para Falbo *et al.* (2013), a variação de generalidade entre as ontologias pode ser vista como uma linha contínua, variando de ontologias de fundamentação para ontologias de domínio. Sendo assim, pode haver diferentes níveis de generalidade nas ontologias dentro do modelo que estão classificadas. A Figura 2.2 ilustra essa visão contínua.

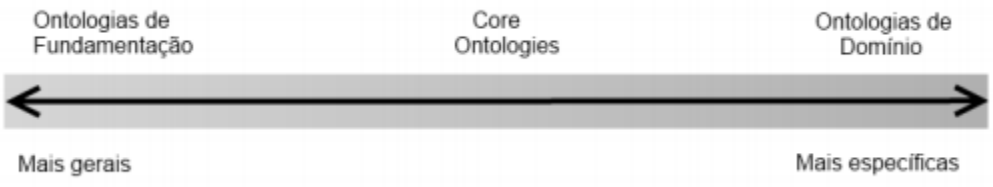


Figura 2.2 –Níveis de generalidade de ontologias (adaptado de (Falbo *et al.*, 2013)).

2.2 Reúso de Ontologias

Ontologias têm sido empregadas com sucesso na resolução de problemas de gerenciamento de conhecimento compartilhado e distribuído, e também da integração de dados entre aplicações (Park, Oh e Ahn, 2011). Muito desse sucesso depende da habilidade de compartilhar e reusar ontologias existentes (van Lamsweerde, 2002). Com a grande adoção de ontologias para resolução de problemas computacionais, métodos e ferramentas que facilitam o reuso se tornaram importantes (Ding e Fensel, 2001).

Desenvolver uma nova ontologia é um processo custoso que requer grande esforço dos engenheiros de ontologias, até mesmo para uma ontologia de médio porte (Corcho, Fernández-López e Gómez-Pérez, 2006). Fernandez *et al.* defendem que para enfrentar este problema, técnicas e métodos eficientes de reúso e avaliação de ontologias são necessários. No entanto, tais ferramentas não estão disponíveis, tornando o reúso de ontologies uma tarefa difícil (Suárez-Figueroa *et al.*, 2012).

Reúso pode ser definido como um processo no qual ontologias disponíveis são utilizadas como material para gerar uma nova ontologia (Bontas, Mochol e Tolksdorf, 2005). Intuitivamente, reúso diz respeito a encontrar uma ontologia existente e manipulá-la de forma que os requisitos do domínio (da nova ontologia) sejam satisfeitos (Katsumi e Grüninger, 2016).

Há diversos métodos de reúso, tais como integração (criação de uma ontologia a partir de outras, sendo permitido fazer alterações nas ontologias reutilizadas e acrescentar novos conceitos ou relações), mapeamento de ontologias (identificação de conceitos ou relações equivalentes entre diferentes ontologias*),* alinhamento de ontologias (acordo mútuo que torna as ontologias consistentes e coerentes*)* e *merging* de ontologias (criação de uma nova ontologia a partir da combinação de outras existentes, sem alterar as ontologias reutilizadas). Há, ainda, o reúso de padrões ontológicos (Katsumi e Grüninger, 2016).

Padrões ontológicos são uma abordagem promissora que favorece o reúso de experiências e boas práticas (Falbo *et al.,* 2013). Padrões são veículos para encapsular o conhecimento (Greenfield, 2010). São considerados um dos meios mais eficazes para nomear, organizar e raciocinar sobre o conhecimento. Segundo Buschmann, Henney e Schmidt (2007), um padrão descreve um problema recorrente de modelagem, que surge em contextos específicos do projeto e apresenta uma solução bem fundamentada para o problema. Dessa forma, padrões

ontológicos são modelos elaborados e que representam uma perspectiva de consenso sobre como resolver um problema específico em um domínio específico (Gangemi e Presutti, 2009).

Pesquisas, como as conduzidas por Presutti *et al.* (2009), relatam que engenheiros de ontologias consideram padrões ontológicos úteis, e que os utilizando, a qualidade e a usabilidade das ontologias resultantes são melhoradas.

Ruy *et al.* (2015) apresentam dois tipos de padrões, de acordo com seu nível de generalidade: FOPs (*Foundational Ontology Patterns*), que são fragmentos de ontologias de fundamentação, e DROPs (*Domain-Related Ontology Patterns*), que são fragmentos de ontologias de núcleo ou de ontologias de domínio. FOPs e DROPs são padrões ontológicos conceituais, ou seja, são utilizados durante a fase de modelagem conceitual da ontologia, sem preocupação com aspectos tecnológicos (Falbo *et al.*, 2016). Este trabalho está interessado nesse tipo de padrão.

Padrões são comumente considerados e aplicados de forma separada. No entanto, nenhum padrão é uma ilha. Pelo contrário, padrões são encontrados agrupados: às vezes com um padrão como uma alternativa a outro, às vezes com um padrão como complemento de outro, ou um número de padrões associados com um grupo bem formado (Buschmann, Henney e Schmidt, 2007). Assim, ao aplicar um padrão, é importante entender e considerar suas relações (Falbo *et al.*, 2013). Neste trabalho, propõe-se uma ferramenta para apoiar o uso de modelagem de objetivos para definir padrões e estabelecer as relações entre eles.

2.3 GORE - *Goal Oriented Requirements Engineering*

A principal forma de medir o sucesso de um software é analisando o grau com que ele atinge seu objetivo. Portanto, identificar esse objetivo deve ser prioridade no desenvolvimento de sistemas de software. Requisitos inadequados, incompletos, ambíguos e inconsistentes têm um impacto significativo na qualidade do software (Lapouchnian, 2005). Pesquisas em Engenharia de Requisitos têm se concentrado em objetivos como forma de prover o *rationale* dos sistemas a serem desenvolvidos (van Lamsweerde, 2002), ajudando na identificação, organização e gerenciamento de requisitos, assim como na forma com que o processo de elaboração de requisitos ocorre (Anton, 2002).

Objetivos são afirmações declarativas de intenções a serem alcançadas. Podem expressar propriedades funcionais (aptidão, capacidade) ou não funcionais (qualidade) com níveis

distintos de abstração: de objetivos estratégicos de alto nível (*e.g.*, otimizar o uso de energia a objetivos táticos de baixo nível (*e.g.*, desligar as luzes no fim do dia (Negri *et al.*, 2017).

A Engenharia de Requisitos Orientada a Objetivos (*Goal Oriented Requirements Engineering* - GORE) surgiu para criar e estudar métodos que abordam a engenharia de requisitos a partir de uma perspectiva orientada para objetivos (Horkoff *et al.*, 2017). Objetivo é considerado o conceito chave em Engenharia de Requisitos (Kavakli, 2002).

Em GORE, objetivos são representados por meio de modelos, utilizando notações definidas por abordagens como iStar (Dalpiaz, Franch e Horkoff, 2016), Tropos (Bresciani *et al.*, 2004), KAOS (Van Lamsweerde, 2009) e Techne (Jureta *et al.*, 2010).

Na área de Engenharia de Ontologias, GORE pode ajudar a compreender o domínio de interesse e os atores envolvidos. Para desenvolver o modelo de objetivos, é preciso que o engenheiro de ontologias identifique os atores do domínio e construa um modelo de objetivos para cada ator. Cada modelo representa os objetivos e tarefas do ator a serem abordados pela ontologia, fornecendo uma visão abrangente da ontologia. Além disso, é possível derivar as questões de competência (i.e., questões que a ontologia deve ser capaz de responder com seus conceitos, relações e axiomas) a partir do modelo de objetivos bem como utilizá-lo como base para o desenvolvimento do modelo conceitual da ontologia, para detalhar o seu escopo (Fernandes, Guizzardi e Guizzardi, 2011).

A Figura 2.3 ilustra, como exemplo, um modelo de objetivos (parcial) do ator Administração de Hospital do domínio hospitalar. O modelo foi construído usando a linguagem iStar (Dalpiaz, Franch e Horkoff, 2016). Na figura, o ator é representado por um círculo no canto superior esquerdo, objetivos são representados por retângulos com bordas arredondadas, tarefas são representadas por hexágonos e recursos são representados por retângulos. Objetivos são decompostos em subobjetivos. Tarefas são realizadas para alcançar objetivos. Recursos são necessários para a realização das tarefas.

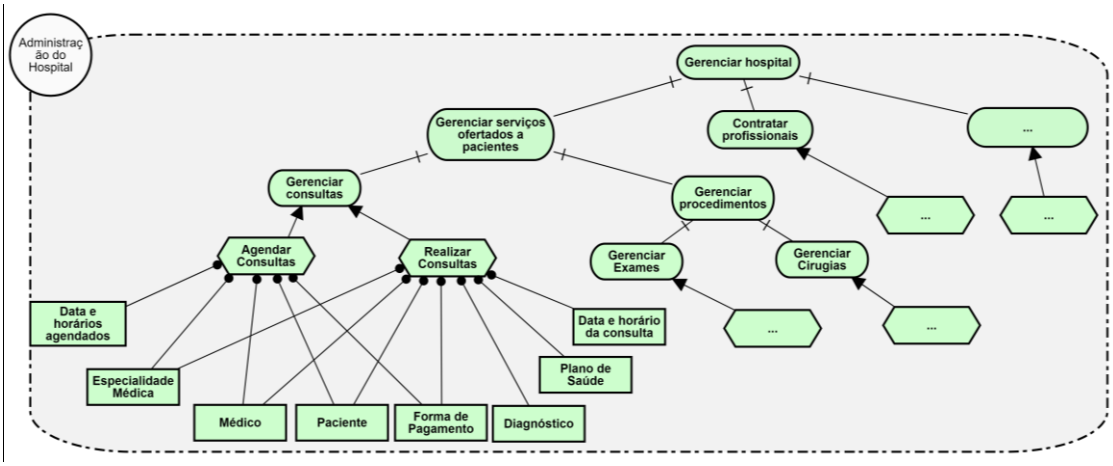


Figura 2.3 – Exemplo de modelo de objetivos para o domínio hospitalar (Salamon, 2018).

A partir do modelo de objetivos é possível derivar questões de competência para definir os requisitos da ontologia. Por exemplo, o modelo mostra que para alcançar o objetivo "Gerenciar Consultas" é necessário realizar a tarefa "Agendar Consultas" e, para isso, é necessário informar a data e horário agendados. Assim, uma questão de competência que a ontologia deve ser capaz de responder é "Qual a data e o horário de uma consulta agendada?".

2.4 GO-FOR - *Goal-Oriented Framework for Ontology Reuse*

Construir ontologias com reúso depende de encontrar ontologias candidatas para serem reusadas (Park, Oh e Ahn, 2011). A busca e seleção de ontologias a serem reusadas deve considerar o alinhamento entre seus escopos e o escopo da ontologia a ser desenvolvida. Isto é, para cada modelo ontológico candidato, deve ser verificado qual fragmento satisfaz os requisitos da nova ontologia. Sendo assim, o reúso de ontologias deve ser baseado nos requisitos da ontologia alvo (Reginato *et al.*, 2019).

GO-FOR (Reginato *et al.*, 2019) é um *framework* orientado a padrões e objetivos para auxiliar o reúso de ontologias. Sendo assim, GO-FOR se beneficia do uso de GORE na definição dos requisitos da ontologia e do uso de padrões ontológicos para favorecer reúso. GO-FOR é baseado em quatro princípios relacionados a questões recorrentes no design de ontologias abordados por Gruber (1991). As questões são:

- *Q1:* Quais fragmentos de informação sobre conceitos são críticos no suporte a compartilhamento (*e.g.,* nome, definição textual, tipo, etc.)?
- *Q2:* Como descrever o objetivo de determinada ontologia?

- *Q3:* Como obter e utilizar *design rationale*?
- *Q4:* Como identificar similaridades entre ontologias?

Para tratar essas questões, GO-FOR segue quatro princípios (Reginato *et al.* , 2019):

(*P1) Padronizar Terminologias e Busca Semântica:* Este princípio está relacionado à questão Q1. É difícil determinar que informações sobre um conceito usadas para nomear estruturas ontológicas (*e.g.,* modelos, padrões) são mais críticas para apoiar o reúso, entretanto é fácil de identificar que eles podem influenciar o reúso de forma negativa (*e.g.*, a utilização de nomes arbitrários e sem significado para estruturas ontológicas dificulta o reúso). Por exemplo, se o engenheiro de ontologias buscar por ontologias sobre comércio eletrônico utilizando o termo "*e-commerce",* ele espera encontrar modelos de ontologias que tratem sobre transações, autenticação, compras, etc. No entanto, sua busca poderia retornar, por exemplo, a ontologia *Good Relations* (Hepp, 2008), cuja descrição afirma que fornece um vocabulário para comércio eletrônico, sendo que, na verdade, fornece vocabulário para especificar ofertas na web, não abordando diversos aspectos sobre comércio eletrônico (Salamon *et al.*, 2017). A deficiência de padrões para nomear estruturas de ontologias prejudica a busca de ontologias adequadas e, consequentemente, compromete a reutilização de ontologias.

(P2) Aplicação do Design Rationale: É importante explicitar os motivos para a construção da ontologia (*e.g.,* o que levou o engenheiro de ontologias a incluir determinados conceitos). Tornar explícito o *design rationale* contribui para tratar as questões Q2 e Q3*,* uma vez que o *design rationale* apoia a definição do objetivo da ontologia.

(P3) Resolver Overlaps: No processo de desenvolvimento de ontologias, deve-se buscar utilizar ontologias já existentes. Evitando o risco de *overlaps* entre ontologias. Sendo assim, é importante estabelecer correspondências entre ontologias para aumentar o reúso. Isso está relacionado à questão Q4, uma vez que resolver *overlaps* envolve o estabelecimento de correspondências entre as ontologias.

(P4) Foco em Padrões: Modelos de ontologias costumam abranger um grande escopo e cobrir vários requisitos, podendo ser difícil identificar um modelo que satisfaça um requisito específico desejado para reutilização. Além disso, ao encontrar um modelo é preciso identificar qual fragmento atende o requisito desejado. Utilizar padrões funciona melhor neste caso, promovendo o reúso, pois modulariza os modelos de ontologias de uma maneira mais fácil de realizar a busca e reutilização.

A Figura 2.4 mostra uma visão geral do funcionamento de GO-FOR. Seguindo o princípio P4, os principais elementos de GO-FOR são padrões ontológicos orientados a objetivos, denominados GOOPs (*Goal-Oriented Ontology Patterns*). Os GOOPs são definidos pela associação entre um fragmento de ontologia e o objetivo que ele satisfaz. Dessa forma, GOOPs podem ser reutilizados de acordo com os objetivos a que estão relacionados. Esses padrões são armazenados em um repositório de padrões ontológicos orientado a objetivos, chamado GOOPR (*Goal-Oriented Ontology Patterns Repository*). Dentro desse repositório os GOOPs podem se relacionar entre si de acordo com seus objetivos.

Para que um engenheiro de ontologias reutilize os GOOPs no desenvolvimento de uma ontologia, ele deve iniciar identificando os atores do domínio e elaborando o modelo de objetivos da ontologia, como sugerido em (Fernandes, Guizzardi e Guizzardi, 2011) e em (Salamon, 2018). O uso de modelos de objetivos está alinhado ao princípio P2, uma vez que contribui para a definição do escopo da ontologia e representação de seu *design rationale*.

Tomando o modelo de objetivos elaborado para a ontologia como base, o engenheiro de ontologias deve verificar se existe um GOOP no GOOPR associado a cada objetivo do modelo. Se existir, o engenheiro de ontologias pode reusar o GOOP realizando a integração deste com o modelo conceitual da ontologia (*i.e.,* desenvolvimento *com* reúso). Caso não exista, o engenheiro pode criar um fragmento de ontologia que satisfaça aquele objetivo. O engenheiro de ontologias pode, então, relacionar o fragmento com o objetivo, dando origem a um novo GOOP e inseri-lo no GOOPR para uso futuro (*i.e.,* desenvolvimento *para* reúso).

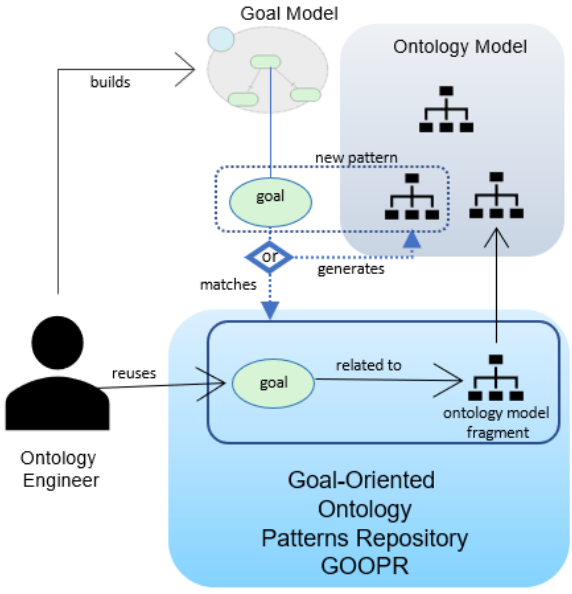


Figura 2.4 – Visão geral da abordagem GO-FOR (Reginato *et al.*, 2019).

A seguir são discutidos alguns aspectos relevantes sobre os principais elementos de GO-FOR: GOOPs e GOOPR.

2.4.1 GOOP – *Goal-Oriented Ontology Pattern*

Um GOOP é formado por um fragmento de ontologia associado a um objetivo. Esse vínculo indica que o fragmento pode ser usado para satisfazer o objetivo. A criação de um GOOP pode ocorrer utilizando-se um fragmento de ontologia já existente ou criando-se um novo modelo visando satisfazer o objetivo.

Um GOOP também é relacionado ao ator que deseja realizar o objetivo contido no GOOP. Atores podem ter o mesmo objetivo, porém os fragmentos necessários para satisfazê-los podem ser distintos. Por exemplo: um médico e um biólogo possuem o objetivo de "Descrever doença" e necessitam de diferentes conceitos para isso. Desta forma, o engenheiro de ontologias deve considerar os atores relacionados aos GOOPs para que o fragmento escolhido possa se encaixar corretamente na ontologia desenvolvida.

Para satisfazer o princípio *P1*, os nomes dados aos objetivos contidos nos GOOPs devem seguir uma terminologia padronizada. Reginato *et al.* (2019) propuseram a utilização de verbos no infinitivo e um substantivo para definição dos nomes (*e.g.,* "Definir oferta de produto", "Descrever artefato"). Analisando diversos modelos de ontologias e seus objetivos, percebeu-se que em sua maioria os objetivos eram associados a verbos similares (*e.g.,* classificar, especificar, definir, descrever, etc). A sugestão tem como objetivo simplificar a busca por GOOPs.

Um GOOP é composto por outro quando o objetivo relacionado ao GOOP é uma decomposição de outro objetivo, também relacionado a outro GOOP. Por exemplo, se o objetivo "Descrever Documento" é subobjetivo de "Descrever Artefato" e o GOOP-A se relaciona com "Descrever Documento", e o GOOP-B se relaciona com "Descrever Artefato", então o GOOP-A é parte do GOOP-B. Essa relação é transitiva e não exclusiva, ou seja, um GOOP pode fazer partes de diversos GOOPs. Uma vez que essa relação é estabelecida, é possível identificar a sobreposição de fragmentos e evitar a duplicação, favorecendo o princípio *P3*.

2.4.2 GOOPR – *Goal-Oriented Ontology Pattern Repository*

GOOPs são armazenados no GOOPR, representando uma camada abstrata para o desenvolvimento de ontologias. Esta camada tem o objetivo de auxiliar o reúso de fragmentos de ontologias já existentes, que quando associados aos objetivos são chamados de GOOPs.

Ao desenvolver uma nova ontologia, o engenheiro de ontologias pode buscar por GOOPs para serem reutilizados para atender o escopo da nova ontologia. A definição dos objetivos que a ontologia em desenvolvimento deve atender é feita a partir do modelo de objetivos. Com o modelo, o engenheiro de ontologias pode buscar GOOPs no GOOPR realizando a comparação dos objetivos que deseja alcançar com os objetivos relacionados a cada GOOP. Por exemplo, suponha que um engenheiro de ontologias tenha desenvolvido um modelo de objetivos para uma ontologia que aborda oferta de produtos na web. No modelo de objetivos o ator Fornecedor deseja "Descrever oferta de produto". Esse objetivo é decomposto em "Descrever produto" e "Descrever condições de oferta" que, por sua vez, é decomposto em "Especificar métodos de pagamento", "Especificar métodos de envio" e "Especificar seguro". Em uma abordagem *bottom-up*, o engenheiro de ontologias busca por GOOPs que satisfaçam o objetivo mais específico, por exemplo "Descrever métodos de pagamento". Desta forma, para alcançar o objetivo mais geral o engenheiro integra os GOOPs que se relacionam com os objetivos mais específicos. Utilizando uma abordagem *top-down,* o engenheiro de ontologias realiza a busca por GOOPs compostos por outros, explorando a decomposição do GOOP selecionado para apoiar o reúso de ontologias.

Explorando a estrutura de composição dos objetivos, o engenheiro de ontologias é capaz de identificar sobreposições e descartar esses fragmentos. Por exemplo, se o GOOP que satisfaz "Descrever condições de oferta" contiver o subobjetivo "Descrever validade da oferta" e o engenheiro de ontologias não estiver interessado nesse subobjetivo, não seria necessário adicionar o fragmento relacionado a este objetivo.

Por fim, ao realizar uma busca e selecionar um GOOP, o engenheiro de ontologias deve considerar o ator relacionado ao objetivo deste GOOP, uma vez que diferentes atores podem possuir o mesmo objetivo, porém necessitarem de fragmentos de ontologias distintos para alcançá-los.

2.5 OWL – *Ontology Web Language*

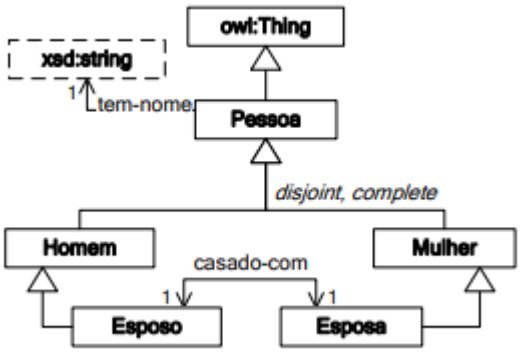
A linguagem OWL (*Ontology Web Language*) é um padrão recomendado pela W3C (*World Wide Web Consortium*) para representação de ontologias na Web Semântica. Ela foi desenvolvida para que aplicações possam processar conteúdo da informação, ao invés de apenas apresentar essa informação para as pessoas (McGuinness e Harmelen, 2004). OWL pode ser usada para representar categorias de objetos e as relações entre eles, além de informações acerca do próprio objeto (Horrocks, Patel-Schneider e Van Harmelen, 2003).

A OWL foi proposta com o objetivo de atender às seguintes restrições: (i) manter a compatibilidade com o padrão RDF (*Resource Description Framework*) para representação de informações na *Web*, expandindo a capacidade de representar o conhecimento denominado "ontológico"; (ii) ter uma sintaxe e semântica bem definidas, assim como um poder de expressividade que atenda a demanda de processamento computacional das informações (Antoniou e Van Harmelen, 2003).

Essa linguagem tem sido utilizada para representar informações em diversos domínios como medicina, biologia, geografia, astronomia e por indústrias aeroespaciais (Grau *et al.*, 2008). Entretanto, com a ampla utilização da linguagem, suas limitações como uma linguagem para lógica descritiva foram expostas (Welty, Fikes e Makarios, 2006). Para suprir essa lacuna, foi proposta uma extensão da linguagem original, chamada de OWL 2 (Grau *et al.*, 2008).

2.5.1 Sintaxe e Semântica

Esta seção detalha as principais funcionalidade e propriedades que a linguagem OWL provê. Visando facilitar o entendimento da linguagem, a Figura 2.6 apresenta um modelo UML mapeado para OWL utilizado em (Zamborlini; Guizzardi, 2010).



Class: Pessoa
EquivalentTo: Homem **ou** Mulher
SubClassOf: owl:Thing
Class: Esposa
EquivalentTo: casada-com **algum** Homem
SubClassOf: Mulher
Class: Esposo
EquivalentTo: casada-com **algum** Mulher
SubClassOf: Homem
DataProperty: tem-nome
Domain: Pessoa
Range: String

Class: Mulher
SubClassOf: Pessoa, casada-com **apenas** Homem
DisjointWith: Homem
Class: Homem
SubClassOf: Pessoa, casada-com **apenas** Mulher
DisjointWith: Mulher
ObjectProperty: casada-com
Domain: Pessoa
Range: Pessoa

Figura 2.6 – Exemplo de modelo UML para representação de OWL

A principal distinção neste domínio de exemplo é que os indivíduos classificados como **pessoa** são homens ou mulheres. Esta distinção é representada no diagrama pelo conjunto de generalização (*generalization set*) que contém as classes **Homem** e **Mulher**, que especializam a classe **Pessoa**, rotulado com os termos *disjoint* e *complete*. As especializações, representadas nas linhas 1 e 2 da primeira coluna do modelo OWL (vide conteúdo textual da Figura 2.6), significam que a interpretação da classe Homem, bem como de Mulher, está contida na interpretação da classe Pessoa, ou seja, todo indivíduo que seja homem ou mulher também é uma instância de pessoa.

O termo *disjoint*, representado na linha 3 da segunda coluna, declara que as classes Homem e Mulher são disjuntas, e significa que a interseção da interpretação destes é vazia, ou seja, um homem não pode ser mulher e vice-versa. O termo *complete*, representado no modelo, declara que a classe Pessoa é particionada de forma completa pelas classes Homem e Mulher, e significa que a interpretação da classe Pessoa é equivalente à interpretação da união das classes Homem e Mulher, ou seja, toda pessoa é um homem ou uma mulher. Além disso, um Homem pode ser Esposo, e uma Mulher pode ser Esposa. A explicação dessas especializações é semelhante à outra já realizada neste parágrafo.

Também é verdade que toda pessoa tem exatamente um nome, que é do tipo *string*. Isto é representado no diagrama pela relação nomeada **tem-nome** entre a classe Pessoa e o tipo de dado *xsd:String* com uma restrição de cardinalidade igual a 1. No modelo OWL, isto se refere à linha 9 da primeira coluna, e significa que a interpretação da classe Pessoa é equivalente à interpretação da expressão de classe que corresponde a todos os indivíduos que instanciam a propriedade de dados **tem-nome** com exatamente um elemento do domínio de dados do tipo *string*.

Ainda as linhas 10 e 11 da primeira coluna restringem respectivamente o domínio e a imagem desta propriedade. A primeira significa que, para cada instância da propriedade tem-nome, o elemento do domínio da propriedade pertence à interpretação da classe Pessoa, por exemplo, se **tem-nome**(João, "João da Silva") então a interpretação de João pertence à união da interpretação da classe Pessoa. Analogamente, a segunda significa que, para cada instância da propriedade tem-nome, o elemento da imagem pertence à interpretação do tipo de dados *String*. Por exemplo, se **tem-nome**(João, "João da Silva") então "João da Silva" tem que ser uma *string*.

Um indivíduo do tipo Esposo é necessariamente casado com exatamente um outro indivíduo do tipo Esposa, e viceversa. Essa situação é representada no diagrama pela relação bidirecional casado-com entre as classes Esposo e Esposa (vide linhas 4 e 7 da primeira coluna e 7 a 9 da segunda coluna). As duas primeiras significam que a interpretação da classe Esposo (Esposa) é equivalente à interpretação da expressão de classe que corresponde a todos os indivíduos que instanciam a propriedade de objetos casado-com com algum elemento do domínio que pertença à interpretação da classe Esposa (Esposo).

A linha 7 a 9 restringem respectivamente domínio e imagem da propriedade casado-com, e, sendo semelhantes, significam que, para cada instância da propriedade casado-com, o elemento do domínio (ou imagem) da propriedade pertence à interpretação da expressão de classe que a todos os indivíduos que pertencem à união da interpretação das classes Esposo e Esposa, por exemplo, se **casado-com**(João, Maria) então a interpretação de João pertence à união da interpretação das classes Esposo e Esposa, bem como a de Maria.

A Tabela 2.1 apresenta os conceitos principais utilizados na linguagem OWL.

Tabela 2.1 – Principais coneitos de OWL.

Sintaxe	Descrição
Características de RDF	
Class	Uma classe define um grupo de indivíduos quem compartilham as mesmas propriedades. Por exemplo, **Debora** e **Frank** pertencem a classe **Pessoa**. Classes podem ser organizadas em uma hierarquia de especialização utilizando *subClassOf*. Em OWL por padrão todas as classes são subclasse de uma classe geral denominada ***Thing***.

Tabela 2.1 – Principais coneitos de OWL (cont.).

Sintaxe	Descrição
Características de RDF	
rdf:Property	Property pode ser usada para definir relações entre individuos ou entre individiduos e valores. Por exemplo, *possui-filhos, casado-com, tem-nome, possui-idade*. Os dois primeiros são propriedades que relacionam duas intancias da classe Pessoa, já os dois ultimos podem ser usados para relacionar uma intancia de Pessoa com uma propriedade do tipo *String* ou *Integer*.
rdfs:subPropertyOf	Hierarquia de propriedades são criadas definindo que uma ou mais propriedade é subpropriedade de outra. As propriedades *possui-irmão* e *possui-parente* são exemplos, onde *possui-irmão* é subproprieade de *possui-parente*. Desta forma, um computador pode processar essa informação e deduzir que se um individuo relacionado a outro por meio da propriedade *possui-irmão*, então ele também está relacionado ao outro individuo através da propriedade *possui-parente*.
rdfs:domain	O domínio de uma propriedade limita os individuos no qual a proprieadade pode ser aplicada. Se uma propriedade relaciona dois individuos, e a propriedade possui uma classe como domínio, então o individuo deve pertencer a aquela classe. Por exemplo, a propriedade *possui-filho* teve seu domínio definido sendo a classe *Mamifero*. Sendo assim, um computador ao processar essa informação pode deduzir que se Frank *possui-filho* Anna, então Frank deve ser um *Mamifero*.
rdfs:range	O alcance de uma propriedade limita os indivíduos que elas podem se associar de acordo com sua classe. Se uma propriedade relaciona dois indivíduos, e a propriedade possui como alcance uma determinada classe, então o indivíduo deve pertencer a classe definida. A propriedade *possui-filho* teve seu alcance definido como a classe *Mamifero*. Sendo assim, é possível deduzir que se Louise se relaciona com Debora por meio da propriedade *possui-filho*, então Debora é instância de *Mamifero*.
Individual	Indivíduos são instâncias de classes, onde propriedades podem ser usadas para relacionar indivíduos entre si. Por exemplo, um indivíduo chamado Debora pode ser descrito como uma instância da classe *Pessoa*, e a propriedade *trabalha-em* pode ser usada para relacionar o indivíduo Debora com o indivíduo UFES.

Tabela 2.1 – Principais coneitos de OWL (cont.).

Sintaxe	Descrição
Características de RDF	
Igualdades e Desigualdade	
equivalentClass	Duas classes podem ser declaradas como equivalente. Classes equivalentes possuem as mesmas intâncias. Equivalência pode ser usada para criar classes sinônimas. Por exemplo, *Carro* pode ser definida como *equivalentClass* de *Automovel*. Desta forma, é possível deduzir que qualquer instância da classe *Carro* é um *Automovel*, e vice-versa.
equivalentProperty	Duas propriedades podem ser declaradas como equivalentes. Propriedades equivalentes relacionam um indivíduo com o mesmo conjunto de outros indivíduos. A propriedade *possui-lider* pode ser *equivalentProperty* de *possui-chefe*. Sendo assim, é possível deduzir que se X *possui-lider* Y, então X *possui*-chefe Y também é verdade.
sameAs	Dois indivíduos podem ser iguais. Esse construtor deve ser usado para criar diferentes nomes para referir um mesmo indivíduo. Por exemplo, o indivíduo Debora pode ser declarado como *sameAs* do indivíduo DeboraMcGuinnes.
Características de Propriedades	
inveseOf	Uma propriedade pode ser definida como inversa de outra propriedade. Se a propriedade P1 é declarada inversa da proprieadade P2, então se X se relaciona com Y por meio de P2, então Y se relacionado com X por meio de P1. Exemplo, se *possui-filho* é inversa de *possui-parente* e Debora *possui-parente* Luiza, então Luiza *possui-filho* Debora.
TransitiveProperty	Propriedades podem ser transitivas. Se uma propriedades é transitiva, então o par (x,y) é uma instância transitiva de da propriedade P, e o par (y,z) é uma instância de P, então (x,z) também é uma instância de P. Por exemplo, se *antepassado* é ma propriedade transitiva, e se Sara é *antepassado* de Luiza (*i.e.,* (Sara, Luiza) é uma instância da propriedade *antepassado*) e Luiza *antepassado* Debora (*i.e.,* (Luiza, Debora) é uma instância da propriedade *antepassado*), então de forma transitiva Sara é *antepassado* de Debora (*i.e.,* (Sara, Debora) é uma instância da propriedade *antepassado*).

Tabela 2.1 – Principais coneitos de OWL (cont.).

Sintaxe	Descrição
Características de Propriedades	
SymmetricProperty	Se uma propriedade é simétrica, então se o par (x, y) é uma instância da propriedade simétrica P, então o par (y, x) também é instância de P. Exemplo, *é-amigo* é uma propriedade simétrica. Então se Frank *é-amigo* Debora, Debora *é-amigo* Frank.
FunctionalProperty	Se uma propriedade é definida como *FunctionalPropery*, então ela não possui mais de um valor para cada indivíduo (podendo não ter valores para algum individuo). Esse construtor é uma forma de declarar que a propriedade possui cardinalidade mínima 0 e máxima 1.

Ademais, por ter a teoria semântica baseada em DL (*Description Logic*), algumas importantes características desta linguagem lógica, incorporadas à semântica de OWL, devem ainda ser observadas. Primeiramente assume-se a hipótese de mundo aberto (HMA), ou seja, não se pode concluir a falsidade de uma informação unicamente pela ausência de uma afirmação sobre sua veracidade. A veracidade da informação deve ser definida explicitamente num axioma ou ser deduzida a partir de outros axiomas existentes, caso contrário, nada se pode afirmar sobre ela.

Outra hipótese assumida é a de nomes não-únicos (HNNU), que significa que dois indivíduos com nomes diferentes podem representar o mesmo objeto do domínio. Em conjunto com a HMA, só se pode afirmar que dois nomes de indivíduos referem-se (ou não) ao mesmo objeto se eles forem explicitamente declarados iguais (ou diferentes), caso contrário, nada se pode afirmar. Finalmente, a monotonicidade significa que a adição de nova informação, supostamente desconhecida, não deve interferir na informação anteriormente derivada. Em outras palavras, o que é verdade em uma situação, deve continuar sendo independentemente da adição de outra informação no modelo. Assim, tais linguagens são tipicamente projetadas para representar cenários estáticos, em que a informação pode ser completada, mas não pode, de fato, mudar. (Hoekstra, 2009).

Capítulo 3

A Ferramenta *GoopHub*

Este capítulo apresenta GoopHub e discute aspectos relacionados ao seu desenvolvimento. Na Seção 3.1 é apresentado o objetivo do sistema. Na Seção 3.2 são apresentados resultados obtidos durante a análise de requisitos do sistema, incluindo o diagrama de casos de uso e o modelo estrutural do sistema. Na Seção 3.3 são discutidos alguns aspectos do projeto de sistema. A Seção 3.4 aborda a implementação do sistema e a Seção 3.5 apresenta algumas de suas telas. Por fim, a Seção 3.6 apresenta um exemplo de uso.

3.1 Objetivo de *GoopHub*

Atualmente, a busca por ontologias para reúso na construção de novas ontologias pode ser bastante custosa, pois embora haja muitos repositórios de ontologias na Web, muitos não são atualizados com frequência e deixam a desejar no que diz respeito às funcionalidades de busca pela ontologia mais adequada para reúso em uma situação particular. Isso contribui para tornar o reúso de ontologias pouco vantajoso, devido ao tempo gasto para buscar ontologias adequadas ser alto (Park, Oh e Ahn, 2011).

Visando prover apoio computacional a GO-FOR e promover seu uso, neste trabalho foi desenvolvido *GoopHub*, que tem como objetivo apoiar a criação, armazenamento e busca de GOOPs (*Goal-Oriented Ontology Patterns)*. *GoopHub* consiste em uma interface web para um repositório que armazena GOOPs (*i.e.,* o GOOPR de GO-FOR), permitindo que engenheiros de ontologias realizem *uploads* e *downloads* de GOOPs.

A partir do modelo de objetivos construído para a ontologia a ser desenvolvida, o engenheiro de ontologias pode utilizar *GoopHub* para buscar GOOPs seguindo uma abordagem *top-down*, buscando por objetivos mais gerais, decompostos em subobjetivos; ou *bottom-up*, procurando por objetivos mais específicos e integrando GOOPs a eles associados até que os objetivos mais gerais sejam atendidos.

3.2 Modelagem Conceitual

Nesta seção são apresentados o modelo de casos de uso e o modelo estrutural de *GoopHub*, que são resultados produzidos durante a Especificação e Análise de Requisitos do Sistema, com o propósito de definir *o que* o sistema deveria fazer.

3.2.1 Casos de Uso

O modelo de casos de uso visa apresentar as funcionalidades que um sistema provê aos usuários, assim como seus fluxos de execução. O ator que interage com GoopHub é o *Engenheiro de Ontologias*, que é a pessoa que utiliza o sistema, realizando consultas a GOOPs, inserindo GOOPs no repositório ou fazendo o download de GOOPs existentes.

A seguir são apresentados os casos de usos do sistema. Para cada caso de uso são apresentadas descrições breves.

Figura 3.1 – Diagrama de Casos de Uso do Sistema *GoopHub*.

Um Engenheiro de Ontologias pode utilizar *GoopHub* para: Realizar *upload* de GOOPs, Buscar GOOPs, Visualizar GOOPs ou Realizar o *download* de GOOPs. Esses casos de uso são considerados operações CRUD (*Create, Retrieve, Upload, Delete)*.

Para "Realizar *upload* de GOOPs", que consiste em inserir um novo GOOP no repositório, o Engenheiro de Ontologias deve preencher um formulário com algumas informações pessoais (e-mail, nome e organização). Após preencher o formulário, o Engenheiro de Ontologias seleciona o arquivo (em formato OWL) com o fragmento do modelo que deseja adicionar ao repositório, indica o objetivo (por meio de uma descrição textual) ao qual o fragmento se relaciona e faz o upload do arquivo OWL contendo a definição do GOOP. Caso o novo GOOP a ser inserido no repositório seja parte de um GOOP já existente, antes de inserir o novo GOOP, o Engenheiro de Ontologias procura pelo GOOP existente e indica que o GOOP a ser inserido está relacionado a ele.

No caso de uso "Buscar por GOOPs" o Engenheiro de Ontologias pode buscar por GOOPs usando busca padrão ou busca avançada. Na busca padrão, o Engenheiro de Ontologias faz a pesquisa usando o objetivo que ele deseja alcançar (ou seja, o objetivo ao qual o GOOP deve estar associado). O sistema irá retornar os GOOPs contidos no repositório e que satisfazem o objetivo informado. Na busca avançada, o Engenheiro de Ontologias utiliza a

linguagem SPARQL para consultar GOOPs no repositório utilizando todos os recursos disponíveis em SPARQL.

Uma vez feita a consulta a GOOPs, o Engenheiro de Ontologias pode realizar o caso de uso "Visualizar GOOPs" ou o caso de uso "Fazer download de GOOPs". Em "Visualizar GOOPs" o Engenheiro de Ontologias pode consultar as classes e propriedades que compõe o GOOP selecionado. Em "Fazer download de GOOPs" o Engenheiro de Ontologias faz o download do arquivo do GOOP (em OWL), devendo indicar onde o arquivo deve ser salvo.

3.2.2 Modelo Estrutural

O modelo estrutural de *GoopHub* pode ser entendido como um metamodelo, uma vez que descreve um domínio representativo de um domínio menos abstrato (*e.g.,* um modelo de modelos) e, ao mesmo tempo, é o núcleo de uma linguagem de modelagem para descrever intâncias (Henderson-Sellers, 2011). O metamodelo foi construído seguindo os princípios propostos por GO-FOR. Sendo assim, ele contém conceitos de OWL e GORE.

Como discutido nos capítulos anteriores, é possível utilizar modelagem de objetivos para explicitar o *design rationale* de ontologias. Desta forma, o metamodelo proposto tem como o objetivo relacionar os fragmentos de ontologias com os conceitos de GORE. Para a elaboração do metamodelo não foram considerados todos os elementos intencionais presentes em GORE. No contexto deste trabalhado, foram considerados atores, objetivos e suas formas de decomposição. A Figura 3.2 ilustra o metamodelo de *GoopHub*.

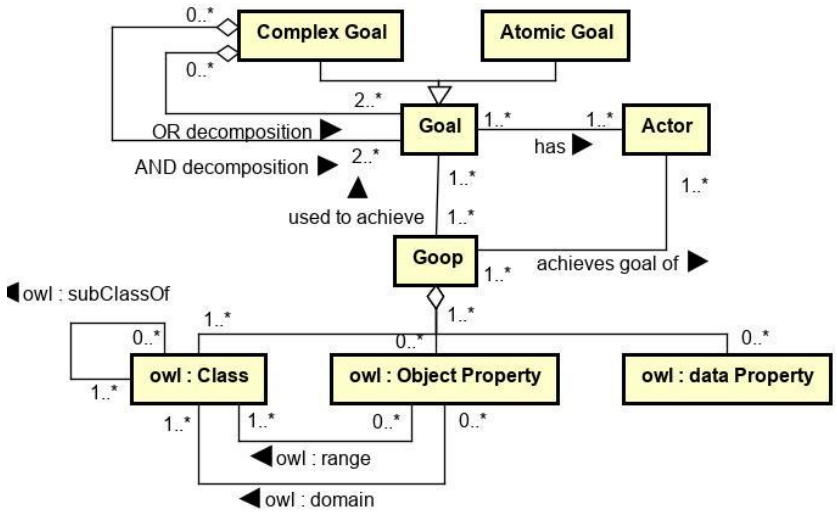


Figura 3.2 – Metamodelo do GoopHub.

No modelo, o conceito central é *Goop*, que realiza a conexão entre os conceitos ontológicos e os conceitos de GORE. Um *Goop* é composto por um fragmento de ontologia e satisfaz um objetivo de um determinado ator. Como a linguagem OWL foi desenvolvida para representar o conhecimento ontológico, os fragmentos de ontologias que compõe um *Goop* são representados pelos conceitos de OWL. *Class* representa as classes que fazem parte da ontologia, por exemplo, no domínio acadêmico, Professor seria uma classe. *Object Property* representa as relações que podem ocorrer entre classes. Por exemplo, Professor *leciona-para* Aluno. A propriedade *leciona-para* possui como domínio a classe Professor e como imagem a classe Aluno. As *Data Properties* são relações que ocorrem entre uma instância de uma classe e um tipo de dado. Por exemplo, a propriedade *tem-nome*(João, "João da Silva") significa que o indivíduo João se relaciona com a *String* "João da Silva" por meio da propriedade *tem-nome*.

A outra parte do modelo representa os conceitos de GORE, são eles: Objetivo, Ator, Objetivo Complexo e Objetivo Atômico. Um *Goop* se relaciona com Objetivo por meio da propriedade "usado para alcançar", isto é, um fragmento de ontologia é utilizado por um determinado ator para satisfazer um determinado objetivo. Atores são centrais para a natureza de modelagem social da linguagem. Os atores são entidades autônomas e ativas que visam atingir seus objetivos, exercendo seu know-how em colaboração com outros atores. Os objetivos podem ser do tipo Complexo e Atômico. Um Objetivo Complexo é composto de outro Objetivo associado por uma decomposição AND ou OR. Decomposições AND são usadas quando todos os subobjetivos devem ser satisfeitos para que o objetivo principal seja alcançado. Uma decomposição OR ocorre quando apenas um subobjetivo necessita ser satisfeito.

A Figura 3.3 exemplifica relações de decomposição entre objetivos. Ela representa um modelo de objetivos feito em iStar para descrever um domínio no qual um cliente deseja comprar um determinado produto pela internet.

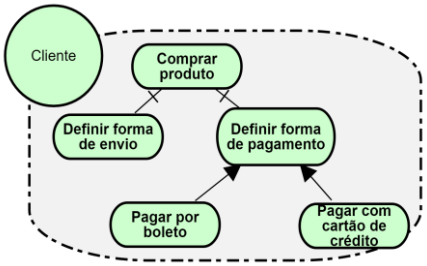


Figura 3.3 – Diagrama de objetivos – "Compra de produtos na Web" - contruido utilizando piStar (Pimentel e Castro, 2018)

O objetivo principal neste diagrama é "Comprar produto". Para que o cliente consiga alcançar esse objetivo é necessário que os objetivos "Definir forma de envio" e "Definir forma de pagamento" sejam satisfeitos. Isto se deve ao fato de que "Comprar produto" é um objetivo do tipo Complexo, decomposto em dois subobjetivos em uma decomposição AND. O subobjetivo "Definir forma de pagamento" também é do tipo Complexo, porém sua decomposição é do tipo OR, o que significa que, para que ele seja satisfeito, basta que "Pagar por boleto" ou "Pagar com cartão de crédito" sejam satisfeitos.

O metamodelo apresentado nesta seção foi implementado utilizando a linguagem OWL. Desta forma, todos os indivíduos criados são instâncias das classes contidas no metamodelo e são relacionados pelas propriedades definidas no mesmo. A vantagem de utilizar OWL é a possibilidade de realizar consultas SPARQL, utilizando os conceitos do metamodelo para filtrar as buscas por instâncias.

3.3 Projeto de Sistema

Uma vez definido *o que* o sistema deveria fazer, na fase de Projeto de Sistema definiu-se *como* deveria fazer. Para isso, foram levados em consideração aspectos tecnológicos, como: linguagem de programação, *frameworks* a serem utilizados, características de interface com o usuário, arquitetura de software e persistência de dados (Falbo, 2012).

As tecnologias aplicadas no desenvolvimento de *GoopHub* foram:

- *Linguagem de Programação Java:* linguagem orientada a objetos e independente de plataforma.
- *Spring Boot framework: Framework* que visa facilitar o processo de configuração e publicação de aplicações.
- *Apache Maven:* Ferramenta de gerência de projeto baseada em *Project object model* (POM) para simplificar o gerenciamento das dependências de bibliotecas de software.
- *Tomcat:* Servidor Web.
- *Stardog:* Banco de dados baseado em triplas.

A arquitetura de software do sistema *GoopHub* baseia-se no padrão MVC (*Model, View, Controller)*. A Figura 3.4 detalha o relacionamento entre as camadas deste padrão. Dessa forma, temos:

- *Model:* Camada responsável pela lógica de negócios e pela interação da aplicação com o banco de dados.
- *View:* Camada que interage diretamente com o usuário, exibindo dados por meio de XHTML, HTML, entre outros. Em *Spring Boot* há páginas HTML com código Java encapsulado.
- *Controller:* Realiza a intermediação entre as camadas anteriores. As requisições feitas no *browser* são processadas pelo *Controller*, após feito o processamento da requisição é feito o retorno para a *View*.

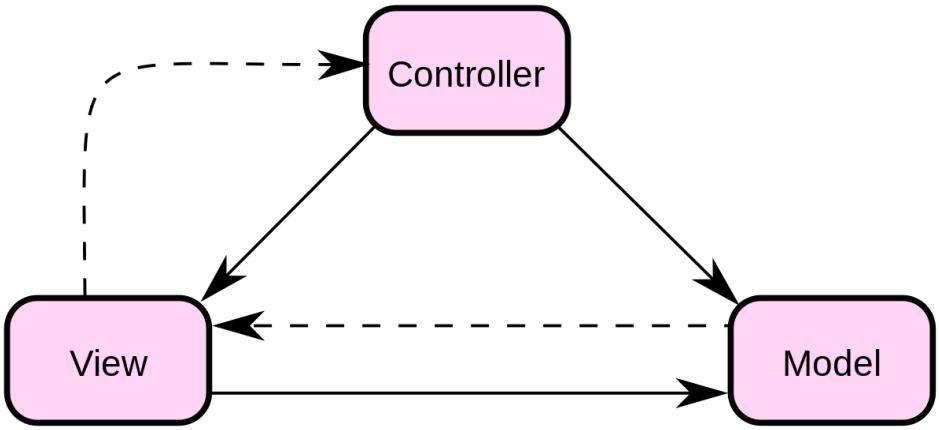


Figura 3.4 – Arquitetura MVC.

3.4 Implementação

Uma aplicação web utilizando o *Spring* é composta pela seguinte organização dos seus arquivos:

- *Service*: pasta na qual estão localizados os recursos e funções chamadas pelos *Controllers*. Realiza a comunicação entre a aplicação e o banco de dados.
- *Controller*: os métodos responsáveis pela comunicação entre cliente e aplicação estão neste pacote. É também onde são definidas as rotas disponíveis para o usuário.
- *Model*: pacote onde as entidades e suas propriedades e relacionamentos são definidos.
- *Resource*: contémnos layouts das páginas que serão exibidas para os usuários.

O sistema desenvolvido neste trabalho utiliza banco de dados de triplas, ou seja, não relacional. Consequentemente, o pacote *Model* não está incluso no projeto. Todo

armazenamento de informações é realizado utilizando o banco de dados de triplas, os arquivos inseridos pelos usuários são lidos, convertidos e inseridos no banco como instâncias do metamodelo apresentado anteriormente. Na Figura 3.5 é apresentada a estrutura dos arquivos presentes no projeto.

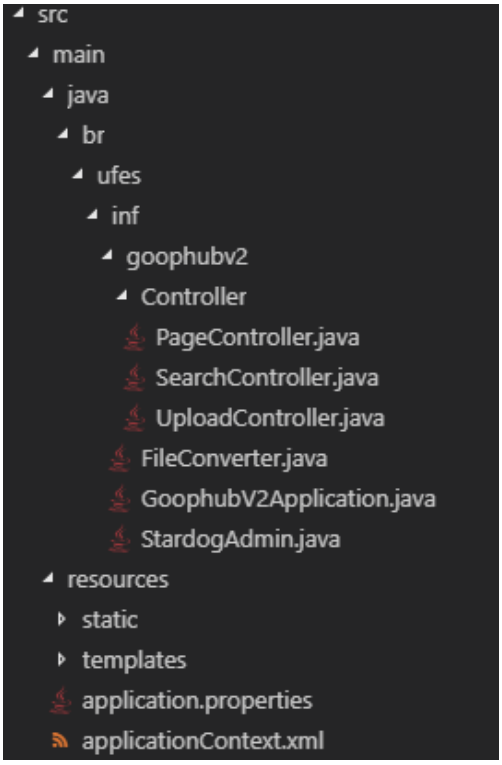


Figura 3.5 – Estrutura do projeto.

Ao lidar com o *Stardog* em aplicações *Spring*, é recomendado utilizar o arquivo de configuração *applicationContext*. Nele é possível definir a porta de comunicação entre aplicação e banco, o *bean* responsável pela comunicação e as configurações iniciais que serão carregadas junto à aplicação. Outro requisito é a classe *StardogAdmin*, encarregada de definir uma conexão com privilégio de administrador com o banco de dados.

No pacote *Controller* estão presentes as classes que realizam a comunicação entre cliente e aplicação. O controlador *PageController* é responsável por retornar as páginas que o usuário solicitar. Para realizar o retorno de uma página web utilizando *Spring* é necessário adicionar a dependência *ThymeLeaf*. Esta dependência é responsável por carregar as páginas localizadas na pasta *templates*, os arquivos estáticos (imagens, vídeos, GIFs, entre outros), e também, injetar código Java dentro do HTML quando necessário manipular os dados da aplicação. Já o *SearchController* é encarregado por todas as requisições que envolvem consultas no banco de dados, por exemplo: inserção de instâncias do metamodelo no banco, consulta por GOOPs e

qualquer consulta realizada no SPARQL Endpoint da aplicação. Por fim, o *UploadController* intermedia o upload de fragmentos de modelo no repositório, sendo responsável converter os arquivos em instâncias para que sejam adicionados ao banco de dados.

GoopHub possui quatro páginas que podem ser acessadas pelos usuários. A página principal, que apresenta *GoopHub*, explicando seus princípios, vantagens e como utilizar. O buscador, onde os usuários podem pesquisar por GOOPs a partir de objetivos utilizando linguagem natural. Após realizar a busca, os GOOPs são retornados e é possível realizar o *download* do fragmento de ontologia desejado. Existe também, uma página para inserção de novos GOOPs, sendo necessário realizar o *upload* do arquivo owl/rdf para que seja realizada a conversão. Por fim, o SPARQL Endpoint, onde é possível usar a linguagem SPARQL para obter resultados mais específicos na busca por GOOPs. Na próxima seção são apresentadas algumas telas de *GoopHub*.

3.4.1 Busca Literal

A busca por GOOPs é a principal funcionalidade da aplicação, por esse motivo, a escolha do banco de dados que seria usado foi importante. Com a busca literal implementada pelo *Stardog* é possível realizar uma pesquisa por todos os valores literais contidos no repositório. Internamente essa funcionalidade utiliza o *Apache Lucene* para realizar essa pesquisa. Assim, é possível utilizar todos os índices de busca fornecidos pelo *Lucene*. As Figuras 3.6 e 3.8 apresentam dois modos de como implementar essa funcionalidade utilizando a API fornecida pelo *Stardog*.

```
Searcher aSearch = connection
    .as(SearchConnection.class)
    .search()
    .limit(2)
    .query("man")
    .threshold(0.5);
```

Figura 3.6 – Fragmento utilizando Apache Lucene e Waldo API.

No fragmento de código acima, para realizar a busca, basta fornecer a classe que realiza a conexão com o banco de dados, logo após definir o limite de busca, o valor literal que será pesquisado e o *threshold*. O *threshold* funciona como um limitador, de acordo com o cálculo realizado pelo algoritmo que verifica a semelhança do termo fornecido com os valores no banco, o algoritmo ordena os valores mais semelhantes com um maior rank, que ele chama de

hits. Dessa forma, o *threshold* irá limitar os resultados para que apareçam apenas aqueles que contenham um valor de *hits* maior que 0.5. A Figura 3.7 é um exemplo de uma busca realizada utilizando este código.

```
API results:
antMan with a score of: 4.844814300537109
ironMan with a score of: 4.844814300537109
```

Figura 3.7 – Resultado da busca utilizando a API.

Outro modo de realizar esta consulta é o apresentado na Figura 3.8. O fragmento mostra a consulta feita utilizando a linguagem SPARQL. O resultado dessa consulta é o mesmo exibido anteriormente.

```
String aQuery = "SELECT DISTINCT ?s ?score WHERE {\n" +
        "\t?s ?p ?l.\n" +
        "\t( ?l ?score ) <" + SearchConnection.MATCH_PREDICATE + "> ( 'man' 0.5 2 ).
        "}";

SelectQuery query = connection
   .select(aQuery);
```

Figura 3.8 – Fragmento utilizando SPARQL.

A busca por GOOPs segue o padrão de implementação mostrado acima. Entretanto, para que a pesquisa ficasse mais eficiente foram adicionados alguns filtros à consulta, para evitar que seja preciso varrer todos os valores do banco e realizar a comparação. Com a busca testada e funcionando, a próxima etapa foi a implementação das rotas que iriam prover a comunicação entre a camada de visualização e a camada de serviços.

3.5 A Ferramenta *GoopHub*

A aplicação web funciona como um intermediador entre o usuário e o repositório, facilitando a busca por GOOPs. Ao acessar o endereço da aplicação, o usuário é direcionado à página principal, como mostra a Figura 3.9. Essa página tem como objetivo apresentar os princípios de GO-FOR, assim como as vantagens de se ter um repositório de padrões ontológicos. Partindo dela é possível acessar as funcionalidades que o *GoopHub* provê, sendo elas: busca, visualização e download de GOOPs, upload de modelos e um endpoint para realizar consultas mais aprofundadas. Os próximos parágrafos detalham a utilização da ferramenta.

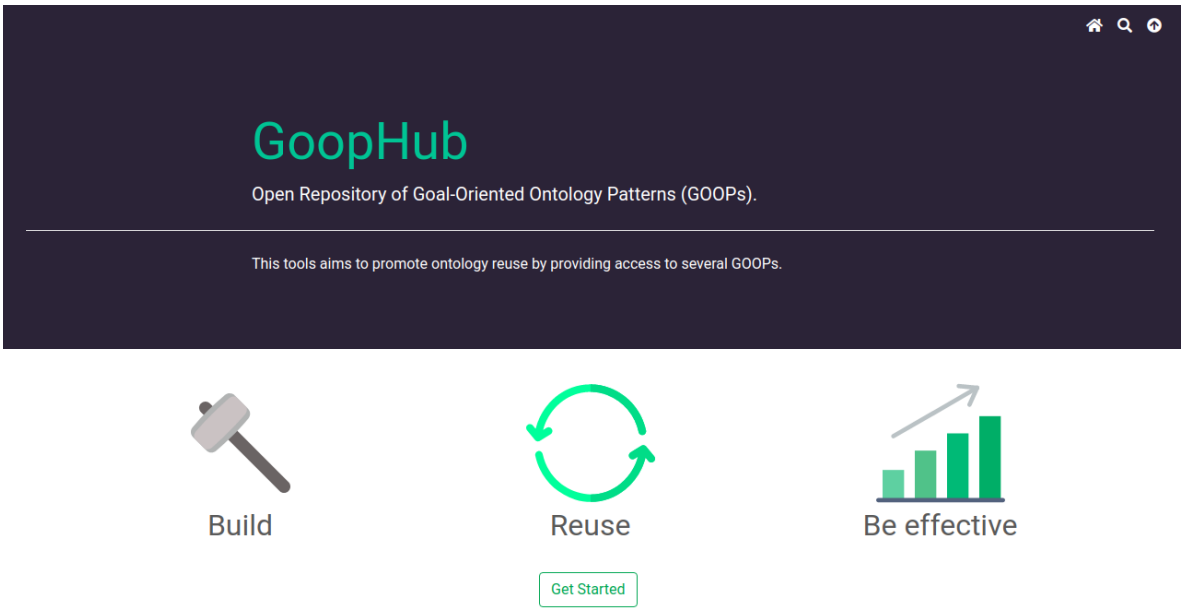


Figura 3.9 – Página inicial do GoopHub.

Para implementar os casos de uso "Buscar por GOOPs", "Visualizar GOOPs" e "Fazer download de GOOPs" foi criada uma página de busca. Usando um campo de busca, o usuário informa o objetivo para o qual deseja encontrar GOOPs. A ferramenta retorna para o usuário, na forma de *cards*, os GOOPs relacionados ao objetivo informado, exibindo uma descrição para cada um deles e informando se trata-se de um objetivo atômico ou complexo. GOOPs associados a objetivos complexos podem ser decompostos em outros. A Figura 3.10 ilustra um exemplo, no qual o usuário buscou por "describe location" e teve como retorno um *card* representado o GOOP que satisfaz esse objetivo "Describe Location".

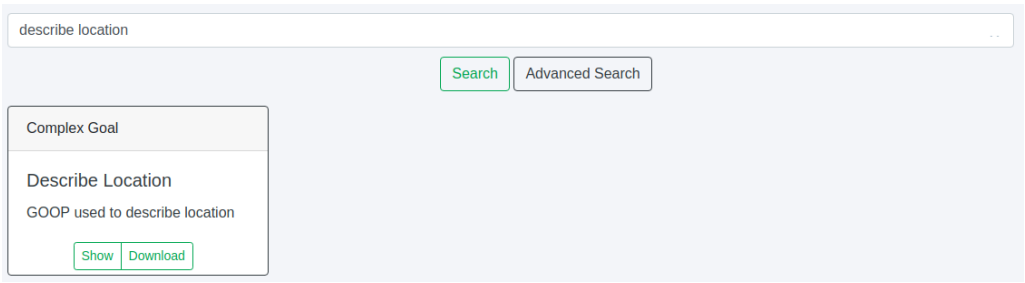


Figura 3.10 – Exemplo de busca feita pelo usuário.

Com o resultado da busca, o usuário pode visualizar os GOOPs (botão "Show" no *card* representando o GOOP) ou fazer o download do arquivo OWL/RDF (botão "Download" no card representando o GOOP).

Além da busca usando um campo simples de busca, como ilustrado na Figura 3.10, há a possibilidade de busca avançada usando o *Endpoint*, no qual é possível usufruir de todas as

funcionalidades que a linguagem de consulta SPARQL oferece. Essa busca é acessada a partir no boão "Advanced Search" na tela iustrada na Figura 3.10. Os resultados são retornados em forma de tabela, para melhor visualização dos dados. A Figura 3.11 exibe um exemplo de busca utilizando SPARQL e o resultado é mostrado na Figura 3.12.

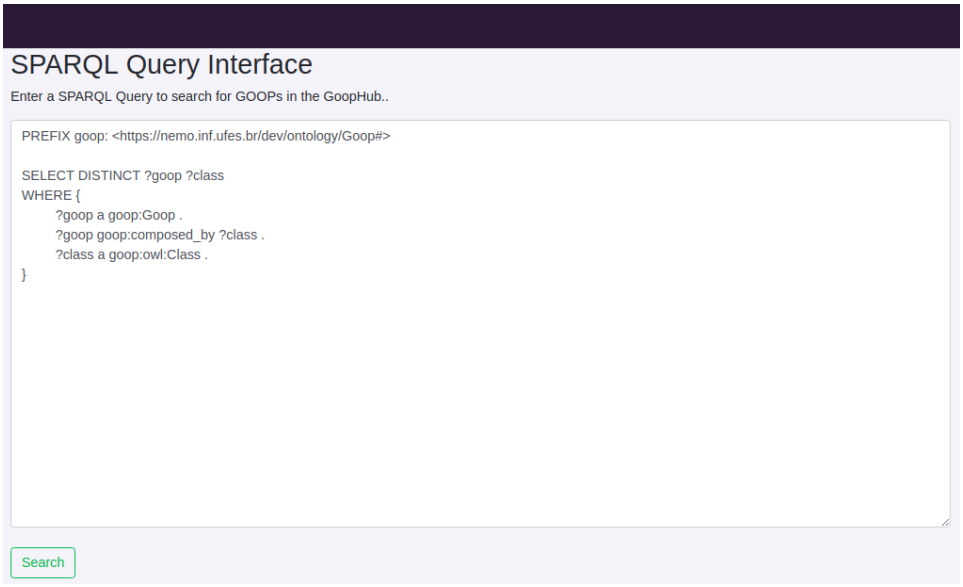


Figura 3.11 – Exemplo de consulta SPARQL.

goop	class
https://nemo.inf.ufes.br/dev/ontology/Goop#_Goop_	https://nemo.inf.ufes.br/dev/ontology/Goop#Bounding_Box
https://nemo.inf.ufes.br/dev/ontology/Goop#_Goop_	https://nemo.inf.ufes.br/dev/ontology/Goop#Area
https://nemo.inf.ufes.br/dev/ontology/Goop#_Goop_	https://nemo.inf.ufes.br/dev/ontology/Goop#Place
https://nemo.inf.ufes.br/dev/ontology/Goop#_Goop_	https://nemo.inf.ufes.br/dev/ontology/Goop#Polygon
https://nemo.inf.ufes.br/dev/ontology/Goop#_Goop_	https://nemo.inf.ufes.br/dev/ontology/Goop#Line

Figura 3.12 – Resultado da consulta SPARQL.

A última funcionalidade da ferramenta diz respeito ao caso de uso "Realizar *upload* de GOOPs" e trata da inserção de novos GOOPs no repositório. Para realizar essa funcionalidade o usuário necessita preencher um formulário e selecionar o arquivo owl/rdf do padrão (i.e., fragmento de ontologia) que deseja inserir. Para GOOPs que satisfaçam objetivos atômicos, o usuário indica o objetivo, seleciona o fragmento e realiza o upload. Caso ele queira associar esse fragmento a algum objetivo complexo, basta buscar por objetivos previamente cadastros utilizando o campo "*Search goals to add*". Após a busca, o usuário seleciona o objetivo desejado. A Figura 3.13 ilustra a inserção de um GOOP no repositório.

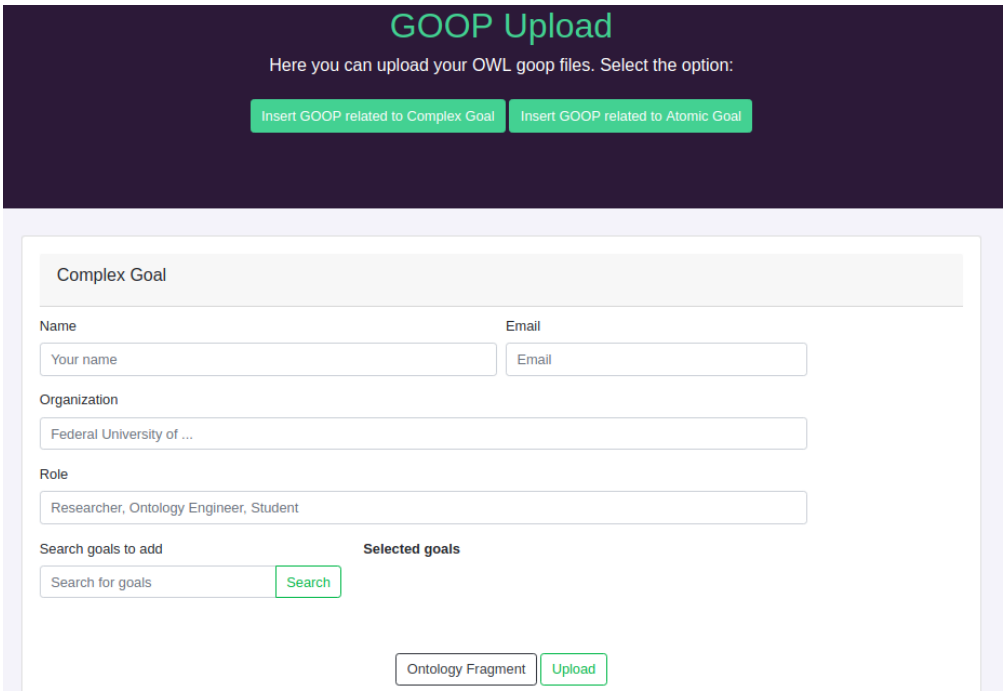


Figura 3.13 – Página para inserção de GOOPs no *GoopHub*.

3.6 Exemplo de Uso de GO-FOR e *GoopHub*[1]

GO-FOR foi aplicado no projeto científico que envolve o maior desastre Ambiental brasileiro, que ocorreu no dia 05 de novembro de 2015. A ruptura da barragem de Fundão, localizada na cidade de Mariana, em Minas Gerais (MG) derramou 55-62 milhões de m³ de minério de ferro, sendo carregado direto para umas das baías mais importantes do Sudeste brasileiro, a baía do Rio Doce.

Com o acontecimento, diversos grupos de pesquisados e entidades governamentais iniciaram projetos para calcular o impacto do desastre, produzindo um grande volume de dados em diversas áreas de conhecimento (hidrologia, geoquímica, biologia, entre outros). Sendo assim, um grupo de pesquisadores da UFES desenvolveu uma ontologia de qualidade da água para auxiliar na interoperabilidade de dados.

Como definido em GO-FOR, o escopo da ontologia foi definido utilizando modelagem de objetivos. Um ator importante no domínio de interesse é o Pesquisador, responsável por coletar amostras, documentar informações sobre as condições do ambiente, obter e analisar dados, entre outros. A Figura 3.14 mostra um fragmento do modelo de objetivos relacionado

[1] O exemplo apresentado nesta seção foi extraído de (Reginato *et al.,* 2019).

ao Pesquisador, focando no objetivo "Descrever Amostra de Peixe". O modelo foi feito utilizando a linguagem de modelagem iStar (Xavier Franch, Lidia López, Carlos Cares, 2015).

Para satisfazer esse objetivo é necessário descrever a coleta de um peixe no Rio Doce para verificar a qualidade da água, analisando as propriedades do peixe coletado. No modelo exibido na Figura 3.14, o objetivo principal é decomposto em três subobjetivos por uma decomposição AND: "Descrever Data", "Classificar Amostra de Peixe" e "Descrever Localidade". Isso significa que para "Descrever Amostra de Peixe", esses três subobjetivos devem ser satisfeitos. O subobjetivo "Descrever Localidade" é decomposto por outros dois objetivos em uma relação OR, "Descrever Ponto Geográfico" e "Descrever Nome de Lugar". Sendo assim, esse objetivo pode ser alcançado caso um dos subobjetivos seja atendido.

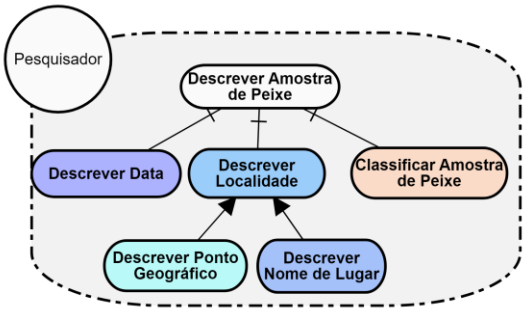


Figura 3.14 – Fragmento do modelo de objetivos relacionado a ontologia de qualidade da água.

Uma vez que o modelo de objetivos foi definido, a próxima etapa foi buscar no *GoopHub* por GOOPs que atendessem esses objetivos. O GOOP apresentado na Figura 3.15 foi um candidato encontrado para satisfazer o objetivo "Descrever Data". Este GOOP foi extraído da *Time Ontology* (*https://www.w3.org/TR/owl-time/*) e armazenado no *GoopHub* durante o desenvolvimento de outra ontologia, no qual o fragmento foi utilizado para satisfazer o objetivo "Descrever Data".

O GOOP foi extraído da *Time Ontology* para reúso, e neste contexto, utilizado no desenvolvimento com reúso. Na Figura 3.15, *Temporal Position* é o conceito mais genérico que possui propriedades para indicar o sistema temporal empregado. *Time Position* tem propriedades que proveem diversas formas de descrever uma posição temporal utilizando números (*e.g.,* coordenada temporal) ou valores nominais (*e.g.,* período geológico, nome dinástico, era arqueológica). *General Date Time Description* especifica o tempo e data usando elementos de relógio e calendário. Por fim, *Date Time Description* transforma o sistema temporal para calendário Gregoriano. A Figura 3.15 exibe esses conceitos, as propriedades foram ocultadas para simplificar o modelo.

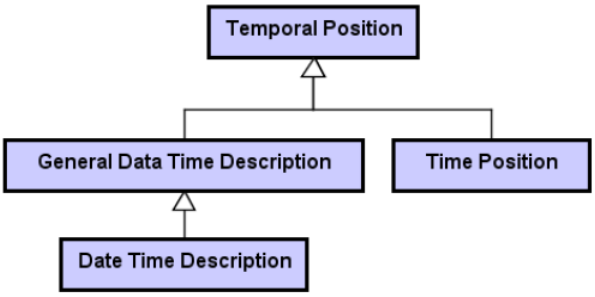


Figura 3.15 - GOOP relacionado a "Descrever Data".

Ao procurar por GOOPs para o objetivo "Descrever Amostra de Peixe" e "Descrever Localidade", nenhum foi encontrado. Portanto, foi preciso desenvolver os fragmentos a partir do zero ou utilizar outras ontologias para satisfazer os objetivos. O caminho escolhido foi extrair novos GOOPs de ontologias já existentes.

Para atender o objetivo "Descrever Localidade", retirou-se um fragmento da *Place Names Ontology* (*http://www.linked.data.gov.au/def/placenames/*). O fragmento é exibido na Figura 3.16. No modelo, *Geometry* possui *Data Properties* que podem ser utilizadas para especificar a latitude e longitude de um lugar. *Place Name*, descreve um lugar usando seu nome. Para distinguir a quais objetivos determinado fragmento está relacionado, os fragmentos foram coloridos com cores distintas. O fragmento completo exibido na Figura 3.16 é necessário para realizar o objetivo "Descrever Localidade". Os conceitos em azul escuro satisfazem o objetivo "Descrever Ponto Geográfico". De forma similar, se o objetivo for "Descrever Lugar pelo Nome", é necessário o fragmento com conceitos na cor azul claro. Esse exemplo representa a relação *part of* entre GOOPs resultante da relação de composição entre os objetivos.

Na figura, existem três GOOPs. Um relacionado ao objetivo "Descrever Lugar pelo Nome", outro relacionado a "Descrever Ponto Geográfico" e um outro que é composto pelos dois anteriores, que atende o objetivo "Descrever Localidade".

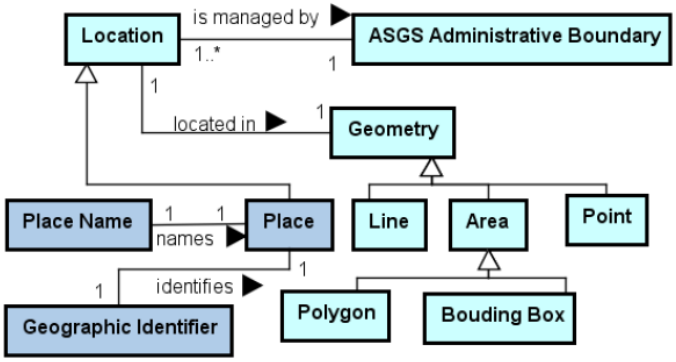


Figura 3.16 – GOOP relacionado a "Descrever Localidade".

Para satisfazer o objetivo "Descrever Amostra de Peixe", foi extraído um fragmento da ontologia *NCBITaxon* (*http://purl.obolibrary.org/obo/ncbitaxon.owl*). Após a extração, o fragmento foi transformado em um GOOP.

A Figura 3.17 ilustra o fragmento da ontologia construída a partir do reúso de GOOPs. Alguns conceitos não foram apresentados para melhor visualização do modelo. Por exemplo, as classes que são instanciadas para descrever uma amostra de peixe não foram incluídas no modelo (foram inclusas apenas as classes relacionadas a hierarquia da taxonomia). As cores dos conceitos indicam os objetivos a que eles se relacionam. Sendo assim, os conceitos associados a seus objetivos representam fragmentos de GOOPs reutilizados para construir a ontologia. O conceito em branco foi adicionado na ontologia para satisfazer seus requisitos.

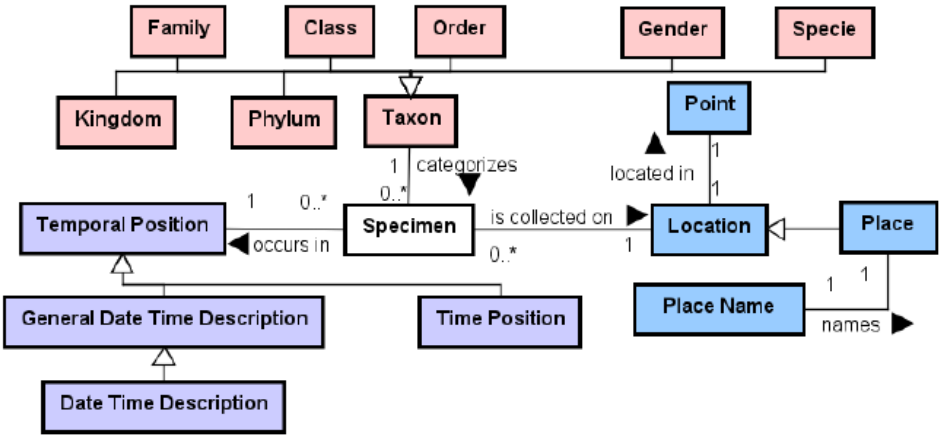


Figura 3.17– Fragmento da ontologia Qualidade da Água – foco em Amostra de Peixes.

Os GOOPs relacionados aos objetivos "Descrever Amostra de Peixe" e "Descrever Localidade", os quais foram criados durante o desenvolvimento da ontologia, foram adicionados e disponibilizados no *GoopHub*, sendo possível consultar as classes e propriedades que os compõem, e também o arquivo do fragmento em formato OWL. Os conceitos reusados podem ser incosporados diretamente no modelo da ontologia sendo desenvolvida ou pode-se criar novos conceitos no modelo, relacionando-os aos originais através das relações, *owl:equivalentClass*, *rdfs:subClassOf*, etc.

Capítulo 4
Conclusão

Neste capítulo são realizadas as considerações finais deste trabalho, sendo apresentadas suas principais contribuições e perspectivas de trabalhos futuros.

4.1 Considerações Finais

GO-FOR (*Goal-Oriented Frameworl for Ontology Reuse*) foi proposto com o objetivo de facilitar e apoiar o reúso de ontologias. Neste trabalho foi desenvolvido *GoopHub,* uma ferramenta que promove o uso de GO-FOR apoiando o armazenamento, busca e recuperação de GOOPs (*Goal-Oriented Ontology Patters*).

GO-FOR propõe a associação de fragmentos de ontologias aos objetivos que eles atendem, dando origem a padrões ontológicos orientados a objetivos (*goal-oriented ontology patterns, GOOPs*). GoopHub foi desenvolvido para servir como repositório de GOOPs. Uma vez que *GoopHub* esteja povoado, reúso poderá ser prática mais frequente, gerando um ciclo virtuoso no desenvolvimento de ontologias com e para reúso.

Para o desenvolvimento do trabalho, foram realizadas atividades do processo de desenvolvimento de software, envolvendo levantamento e análise de requisitos, projeto, implementação, testes e implantação. Para a etapa de implementação foi importante estudar as tecnologias que seriam usadas no desenvolvimento da aplicação, sendo necessário estudar *frameworks*, bancos de dados, entre outras tecnologias. O aprendizado na área de desenvolvimento e reúso de ontologias também foi importante para execução do projeto. Além disso, a disciplina Desenvolvimento Web e Web Semântica foi de grande ajuda, pois apresentou e detalhou como funciona o desenvolvimento de uma aplicação Web utilizando Java, e também, introduziu o uso de SPARQL para consulta de arquivos rdf, owl, entre outros.

Entre as dificuldades encontradas, destaca-se a implementação utilizando *Stardog*, devido a não haver muito material para sanar dúvidas na internet, estando disponíveis apenas o fórum da própria ferramenta, alguns tutoriais no próprio site da ferramenta e o GitHub.

No Capítulo 1 foram identificados os objetivos a ser alcançados neste trabalho. Na Tabela 4.1 é apresentado para cada objetivo específico o produto que serve como evidência de seu alcance.

Tabela 4.1 – Objetivos específicos e sua situação na conclusão da monografia.

Objetivo	Status	Resultado
Analisar o problema e projetar uma solução para a criação, armazenamento e recuperação de GOOPs	Atendido	Elaboração do projeto de sistemas para a solução proposta (vide Capítulo 3).
Implementar a solução projetada.	Atendido	Implementação do repositório conforme o projeto de sistemas (vide Seção 3.3, 3.4 e 3.5).
Definir um conjunto de GOOPs e disponibilizá-los para reúso.	Atendido	Adição de três GOOPs no repositório, sendo eles: "Descrever Data", "Descrever Localidade" e "Classificar Amostra de Peixe" (vide Seção 3.6).
Aplicar a solução implementada para desenvolvimento de ontologias com reúso.	Atendido	Ontologia para classificação de amostrs de peixe desenvolvida aplicando o *framework* GO-FOR junto com o repositório GoopHub (vide Seção 3.6).

4.2 Contribuções e Trabalhos Futuros

Este trabalho foi desenvolvido no âmbito de um projeto de pesquisa mais amplo, no qual se propôs GO-FOR. A principal contribuição deste trabalho é a ferramenta *GoopHub*, que apoia a utilização de GO-FOR fornecendo um repositório para armazenamento, busca e recuperação de GOOPs.

A utilização de GO-FOR e *GoopHub* no desenvolvimento de uma ontologia de qualidade da água mostrou que seu uso é viável. O esforço necessário para criar GOOPs pode ser compensado pelos benefícios de reusá-los. Entretanto, novos estudos e aplicações são necessárias. Como trabalho futuro, pretende-se povoar o repositório com mais GOOPs e realizar experimentos e estudos de casos para avaliar os efeitos de utilizar GO-FOR no processo de engenharia de ontologias. Ainda, como forma de aprimorar GO-FOR, pretende-se utilizar outros elementos presentes nos modelos de objetivos (*e.g.,* tarefas), permitindo realizar um detalhamento mais preciso no modelo de objetivos, tornando a busca por GOOPs mais refinada. Esses trabalhos futuros serão realizados no âmbito do projeto de pesquisa mencionado.

Especificamente para o *GoopHub*, planeja-se associar propriedades aos GOOPs (por exemplo, taxa de reúso, qualidade, avaliação pelo usuário). Dessa forma, quando mais de um GOOP for retornado na busca, será possível ranqueá-los utilizando essas propriedades. Além disso, funcionalidades de cadastro de usuários, controle de versão e download do arquivo OWL são funcionalidades que se pretende adicionar em breve. Outro trabalho futuro diz respeito à inclusão de funcionalidades para suportar a integração de GOOPs, com o objetivo

de gerar novos GOOPs e auxiliar o engenheiro de ontologias na integração de GOOPs em sua ontologia.

Por fim, um trabalho futuro que merece destaque é a implementação (ou integração) de um editor de ontologias em *GoopHub* para facilitar a visualização dos fragmentos e gerar os arquivos das ontologias modeladas utilizando a integração de diversos GOOPs.

De modo geral, a perspectiva é que *GoopHub* seja evoluído para ser adotado por engenheiros de ontologias que tenham interesse em adotar uma abordagem orientada a objetivos. Espera-se que o apoio automatizado provido pela ferramenta contribua para difundir o desenvolvimento de ontologias com e para reúso, melhorando a qualidade e a produtividade do processo engenharia de ontologias.

Referências Bibliográficas

Anton, A. I. (2002) 'Goal-based requirements analysis', in. doi: 10.1109/icre.1996.491438.

Antoniou, G. e Van Harmelen, F. (2003) 'OWL Web Ontology Language', *Handbook on Ontologies in Information Systems*, pp. 67–91. doi: 10.1145/1295289.1295290.

Bontas, E. P., Mochol, M. e Tolksdorf, R. (2005) 'Case Studies on Ontology Reuse', in *Proceedings of the IKNOW05 International Conference on Knowledge Management*.

Bresciani, P. *et al.* (2004) 'Tropos: An agent-oriented software development methodology', *Autonomous Agents and Multi-Agent Systems*. doi: 10.1023/B:AGNT.0000018806.20944.ef.

Buschmann, F., Henney, K. e Schmidt, D. C. (2007) *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*, *Library*. doi: 10.1093/intimm/dxu027.

Corcho, O., Fernández-López, M. e Gómez-Pérez, A. (2006) 'Ontological engineering: Principles, methods, tools and languages', in *Ontologies for Software Engineering and Software Technology*. doi: 10.1007/3-540-34518-3_1.

Dalpiaz, F., Franch, X. e Horkoff, J. (2016) 'iStar 2.0 Language Guide', pp. 1–15. Available at: http://arxiv.org/abs/1605.07767.

Ding, Y. e Fensel, D. (2001) 'Ontology Library Systems: The key to successful Ontology Reuse.', *SWWS*.

Falbo, R. *et al.* (2013) 'Organizing ontology design patterns as ontology pattern languages', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. doi: 10.1007/978-3-642-38288-8-5.

Falbo, R. A. *et al.* (2016) 'An ontology pattern language for service modeling', in. doi: 10.1145/2851613.2851840.

Falbo, R. de A., Guizzardi, G. e Duarte, K. C. (2004) 'An ontological approach to domain engineering', in. doi: 10.1145/568760.568822.

Fernandes, P. C. B., Guizzardi, R. S. S. e Guizzardi, G. (2011) 'Using Goal Modeling to Capture Competency Questions in Ontology-based Systems', *Journal of Information and Data Management*.

Gangemi, A. e Presutti, V. (2009) 'Ontology Design Patterns', in *Handbook on Ontologies*. doi: 10.1007/978-3-540-92673-3_10.

Grau, B. C. *et al.* (2008) 'OWL 2: The next step for OWL', *Web Semantics*. doi: 10.1016/j.websem.2008.05.001.

Greenfield, J. (2010) 'Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools', in. doi: 10.1007/978-3-540-28630-1_19.

Gruber, T. R. (1991) 'The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases', in *Principles of knowledge representation and reasoning: Proceedings of the Second International Conference*.

Gruber, T. R. (1993) 'A translation approach to portable ontology specifications', *Knowledge Acquisition*. doi: 10.1006/knac.1993.1008.

Guarino, N. (1997) 'Understanding, building and using ontologies', *International Journal of Human Computer Studies*. doi: 10.1006/ijhc.1996.0091.

Guarino, N. (1998) 'Formal Ontology and Information Systems', *Formal ontology in information systems: Proceedings of the first international conference*.

Henderson-Sellers, B. (2011) 'Bridging metamodels and ontologies in software engineering', *Journal of Systems and Software*. doi: 10.1016/j.jss.2010.10.025.

Hepp, M. (2008) 'GoodRelations: An ontology for describing products and services offers on the web', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. doi: 10.1007/978-3-540-87696-0-29.

Horkoff, J. *et al.* (2017) 'Goal-oriented requirements engineering: an extended systematic mapping study', *Requirements Engineering*. doi: 10.1007/s00766-017-0280-z.

Horrocks, I., Patel-Schneider, P. F. e Van Harmelen, F. (2003) 'From SHIQ and RDF to OWL: The making of a Web Ontology Language', *Web Semantics*. doi: 10.1016/j.websem.2003.07.001.

Jureta, I. J. *et al.* (2010) 'Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling', in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010*. doi: 10.1109/RE.2010.24.

Katsumi, M. e Grüninger, M. (2016) 'What is ontology reuse?', in *Frontiers in Artificial Intelligence and Applications*. doi: 10.3233/978-1-61499-660-6-9.

Kavakli, E. (2002) 'Goal-oriented requirements engineering: A unifying framework', *Requirements Engineering*. doi: 10.1007/PL00010362.

van Lamsweerde, A. (2002) 'Goal-oriented requirements engineering: a guided tour', in. doi: 10.1109/isre.2001.948567.

Van Lamsweerde, A. (2009) 'Reasoning about alternative requirements options', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. doi: 10.1007/978-3-642-02463-4_20.

Lapouchnian, A. (2005) 'Goal-Oriented Requirements Engineering : An Overview of the Current Research', *Requirements Engineering*. doi: 10.1007/s00766-003-0178-9.

López de Vergara, J. E. *et al.* (2003) 'Semantic management: Application of ontologies for the integration of management information models', in *IFIP Advances in Information and Communication Technology*. doi: 10.1007/978-0-387-35674-7.

McGuinness, D. L. e Harmelen, F. Van (2004) 'OWL Web Ontology Language Overview', *W3C recommendation*.

Negri, P. P. *et al.* (2017) 'Towards an ontology of goal-oriented requirements', in *CIbSE 2017 - XX Ibero-American Conference on Software Engineering*.

Noy, N. F. (2004) 'Semantic Integration: A Survey Of Ontology-Based Approaches', *Newsletter ACM SIGMOD Record*.

Park, J., Oh, S. e Ahn, J. (2011) 'Ontology selection ranking model for knowledge reuse', *Expert Systems with Applications*. doi: 10.1016/j.eswa.2010.10.002.

Poveda-Villalón, M., Carmen Suárez-Figueroa, M. e Gómez-Pérez, A. (2010) 'Reusing ontology design patterns in a context ontology network', in *CEUR Workshop Proceedings*.

Presutti, V. *et al.* (2009) 'Extreme design with content ontology design patterns', in *CEUR Workshop Proceedings*.

Pimentel, João and Castro, Jaelson. piStar Tool – A Pluggable Online Tool for Goal Modeling. 2018 IEEE 26th International Requirements Engineering Conference, pp. 498-499.

Reginato, C. C. *et al.* (2019) 'GO-FOR : A Goal-Oriented Framework for Ontology Reuse', *IEEE IRI 2019*.

Ruy, F. B. *et al.* (2015) 'Ontology engineering by combining ontology patterns', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. doi: 10.1007/978-3-319-25264-3_13.

Salamon, J. S. *et al.* (2017) 'Using goal modeling and ontoUML for reengineering the good relations ontology', in *CEUR Workshop Proceedings*.

Salamon, J. S. (2018) 'Uma Abordagem Orientada a Objetivos para Desenvolvimento de Ontologias baseado em Integração'.

Scherp, A. *et al.* (2011) 'Designing core ontologies', *Applied Ontology*. doi: 10.3233/AO-2011-0096.

Suárez-Figueroa, M. C. *et al.* (2012) *Ontology engineering in a networked world*, *Ontology Engineering in a Networked World*. doi: 10.1007/978-3-642-24794-1.

Welty, C., Fikes, R. e Makarios, S. (2006) 'A reusable ontology for fluents in OWL', in *Formal Ontology in Information Systems. IOS Press*.

Xavier Franch, Lidia López, Carlos Cares, D. C. (2015) 'The i* Framework for Goal-Oriented Modeling Xavier', (Byass 2014).