COMPUTING 2nd Edition Volume I

# Representação do Conhecimento de Forma a Buscar Oportunidades de Refatoração Através de uma Ontologia

**Fernando Quatrin Campagnolo**[1], **Eduardo Kessler Piveta**[1]

[1] Universidade Federal de Santa Maria (UFSM) RS – Brazil

{fcampagnolo, piveta} @inf.ufsm.br

***Abstract.*** *During life cycle of software systems, they can be modified and adapted to the new features. These modifications can increase the complexity of software systems and decrease your quality. One way to improve your quality is to apply transformations in programs, more specifically, refactoring. One of the activities in the refactoring process proposed by Piveta (2009) is Select Refactoring Patterns. This activity selects a set of refactoring patterns to apply in a software system. From that scenario, this survey propose: (i) Refactoring knowledge representation through OWL ontology; (ii) Using the proposed ontology as database for selecting a set of refactoring to be applied in software system. For this a Java application was developed using Jena API. This application makes it easier the activity Select Refactoring Patterns.*

***Resumo.*** *Durante o ciclo de vida dos sistemas de software, eles precisam ser melhorados, modificados e adaptados a novas funcionalidades. Tais modificações podem aumentar a complexidade e diminuir sua qualidade. Uma das maneiras de melhorar sua qualidade é aplicar transformações, mais especificamente, refatorações. Uma das atividades do processo de refatoração proposto por Piveta (2009), é a seleção do conjunto de refatorações a serem aplicadas em um sistema de software. A partir desse cenário, este trabalho tem como objetivo: (i) Representar o conhecimento de refatoração através de uma ontologia OWL; (ii) Utilizar a ontologia proposta como base de dados para selecionar o conjunto de refatorações a serem aplicadas. A fim de facilitar a seleção, desenvolveu-se uma aplicação Java utilizando a API Jena. Essa possibilita filtrar as refatorações por algumas características como vantagem e domínio.*

## 1. Introdução

Atualmente empresas como Google, Microsoft e IBM vem adotando métodos ágeis em alguns dos seus projetos. Eles surgiram na década de 90 a fim de desburocratizar os processos tradicionais de desenvolvimento de software. Uma das práticas utilizadas para auxiliar no desenvolvimento, manutenção e na gerência dos sistemas de software, por essas metodologias, é a refatoração [Stellman and Greene 2013].

De acordo com Mens and Tourwe (2004), os sistemas de software tendem a evoluir em curtos períodos de tempo, buscando atender novos requisitos. Para isso, tais sistemas precisam ser melhorados, modificados e adaptados a esses novos requisitos. Essas modificações tendem a aumentar a complexidade desses sistemas e diminuir a sua qualidade e quanto mais complexo for um sistema de software, mais custosa será sua

manutenção e implementação de novos requisitos. Uma das maneiras de auxiliar na gerência desta complexidade e manter um sistema de software atualizado é a aplicação de transformações em programas, mais especificamente, a aplicação de refatorações, uma das práticas prevista e utilizada em XP. De acordo com Opdyke (1992), uma refatoração é definida como uma transformação de uma forma de representar o código para outra, a qual, modifica-se seu comportamento interno sem modificar seu comportamento externamente observável [Mens and Tourwe 2004] [Opdyke 1992].

Realizando uma busca por refatorações foram encontrados diversos catálogos de refatoração que podem ser aplicados em diversas situações. Essas refatorações variam de acordo com o seu domínio e propósito por exemplo, o catálogo de refatorações descrito por Fowler et al. (1999), um conjunto de mais de 70 refatorações que podem ser aplicadas em linguagens de programação orientadas a objetos (OO), como Java e C#. Tais refatorações têm como objetivo tornar o código mais legível, manutenível e reutilizável [Fowler et al. 1999].

Piveta (2009) propôs um processo que descreve as atividades necessárias para aplicar um conjunto de refatorações num sistema de software. Esse processo inicia pela atividade de seleção de métricas e finalizada pela atividade de aplicação das refatorações. Uma das atividades propostas nesse processo é a seleção de um conjunto de refatorações que serão aplicadas num sistema de software. Até o presente momento, esta seleção deve ser feita manualmente, ou seja, deve-se buscar os catálogos que contém as refatorações desejadas e selecioná-las uma a uma. Dependendo das métricas, dos catálogos de refatorações e do domínio, essa atividade pode demandar um grande tempo e esforço.

A fim de otimizar esta atividade, este trabalho tem como objetivo desenvolver uma ontologia capaz de representar informações sobre as refatorações que servirá como base para a seleção de um conjunto de refatorações a serem aplicadas em um determinado sistema de software. Sendo assim esta ontologia deve ser capaz de representar as mais diversas refatorações junto a algumas características excrecências, como por exemplo, seu domínio, as vantagens, catálogo que ela pertence, nome, resumo, motivação, mecânica e exemplos.

Os catálogos de refatorações descritos por Fowler et al. (1999), Kerievsky (2008), Júnior (2014) e Harold (2009) serão utilizados como bases para desenvolver e popular esta ontologia. Para facilitar a seleção das refatorações desejadas será desenvolvida uma ferramenta em Java utilizando a *API* (*Application Programming Interface*) Apache Jena [Jena 2015].

As seções deste trabalho estão estruturadas da seguinte forma: Na seção 2 são apresentados os assuntos que embasaram este trabalho, refatoração e ontologia. Na seção 3 a ontologia proposta é explicada justificando algumas escolhas tomadas. Na seção 4, demonstra-se como as buscas foram realizadas utilizando o plugin SPARQL no Protégé e a ferramenta desenvolvida em Java utilizando a API Jena. Na seção 5 alguns trabalhos relacionados são abordados. Por fim, na seção 6, as conclusões deste trabalho são apresentadas.

## 2. Referencial Teórico

Nesta seção serão abordados alguns conceitos essenciais que foram utilizados para desenvolver este trabalho. Iniciando pelos conceitos de refatoração, a descrição do processo

utilizado, os conceitos de ontologia e seus componentes, e, finalizando, os conceitos de OWL *Ontology Web Language*.

## 2.1. Refatoração

Os sistemas de software passam por diversas mudanças durante seu ciclo de vida. Essas mudanças inflam os sistemas de software e os tornam mais complexos e de baixa qualidade. Uma das maneiras de reduzir esta complexidade e manter o software atualizado é utilizar refatorações de código. De acordo com Opdyke (1992), a refatoração é o processo de melhoria do software que modifica sua estrutura e/ou seu comportamento interno sem modificar seu comportamento externamente observável. Para Fowler et al. (1999) a refatoração é o processo de alterar um sistema de software de um estado A para B, sem alterar seu comportamento externo e ainda melhora a estrutura interna. Por exemplo, a refatoração Renomear Método, que consistem em renomear um método que não é entendível e todas as suas ocorrências no sistema de software, melhorando o entendimento do código sem mudar seu comportamento [Fowler et al. 1999].

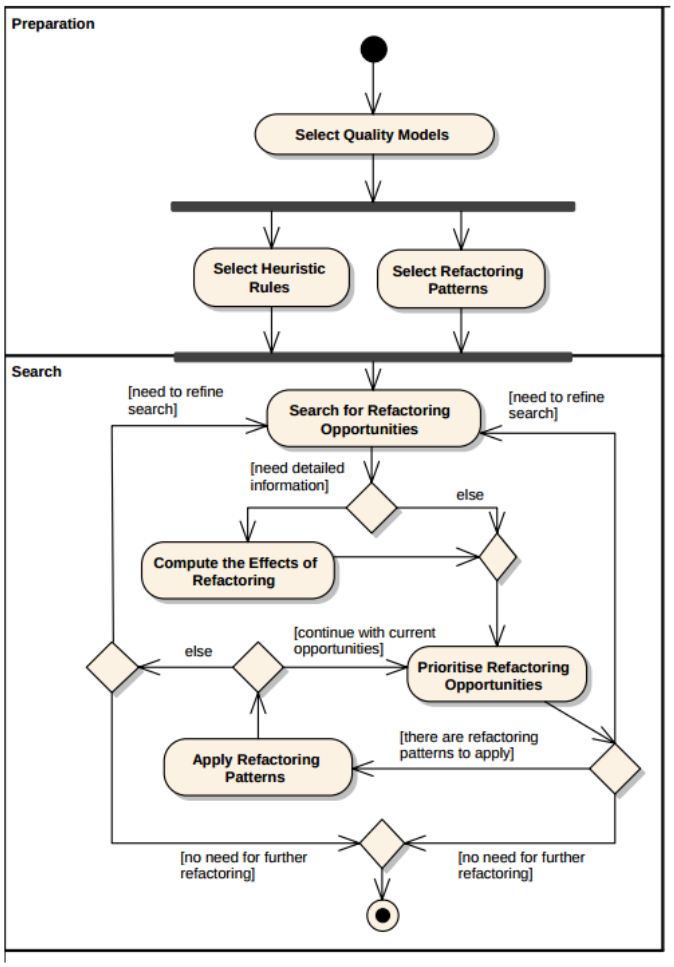


**Figura 1. Processo proposto por [Piveta 2009].**

Piveta (2009) propôs um processo detalhado para a aplicação de refatorações. Esse inclui as principais atividades para aplicar as refatorações num sistema de software, são elas: Seleção e criação de métodos de qualidade, padrões de refatoração e funções

heurísticas; Busca e priorização de oportunidades de refatoração; Avaliação dos efeitos da refatoração na qualidade de software; Análise de vantagens e desvantagens e a aplicação de padrões de refatoração. A Figura 1 mostra as atividades utilizando um diagrama de atividade UML (*Unified Modeling Language*). Uma das atividades propostas nesse processo é *Select Refactoring Patterns*. Nessa atividade o especialista do domínio seleciona as refatorações a serem aplicadas no sistema de software, uma tarefa que pode ser custosa se o desenvolvedor conhecer pouco do assunto. Este trabalho tem como objetivo facilitar a escolha destas refatorações utilizando alguns filtros como domínio e benefícios da refatoração [Piveta 2009].

### 2.2. Ontologia

O termo ontologia é emprestado da filosofia. Ele representa um relato sistemático da existência, ou seja, representa formalmente um conhecimento consensual que pode ser reutilizado e compartilhado. Uma ontologia pode ser definida como uma especificação formal e explícita de um conceito compartilhado [Gruber 1993]. Ou ainda, pode ser defina como uma formalização do conjunto de termos de um domínio específico e compartilhado por uma comunidade de usuários [W3C 2015]. Os usuários especificam as definições dos termos e descrevem suas relações um com os outros. Esse formalismo serve de base para a comunicação e entendimento entre humanos e máquina. Para formalizar a rede conceitual usa-se cinco tipos de componentes, são eles [Guarino 1998]:

- Classes: Mecanismo de abstração para o agrupamento de recursos com características semelhantes, ou seja, a representação de conceitos do domínio;
- Relações: Representam as conexões entre conceitos do domínio.
- Funções: Caso especial de relações onde o enésimo elemento é único para os vários elementos precedentes.
- Axiomas: São regras que declaram as relações que os elementos de uma ontologia devem cumprir. Inferem conhecimento que não estão indicados na ontologia.
- Instâncias: Representação de objetos ou conceitos na ontologia proposta.

A OWL é uma linguagem que facilita a descrição de ontologias para a web semântica. Com ela é possível definir, descrever e instanciar classes, propriedades e relacionamentos da ontologia. Para relacionar as classes com suas propriedades e/ou atribuir outras informações, são utilizados axiomas, os quais contêm restrições e conjuntos de valores necessários e/ou permitidos [Guarino 1998] [OWL 2015] [W3C 2015].

## 3. Representando o Conhecimento de Refatoração

Nesta seção será demonstrado a visão geral atividade abordada, onde a ferramenta e a ontologia desenvolvida é utilizada e o processo de desenvolvimento e pesquisa da ontologia proposta. Quanto a ontologia, primeiramente foi realizada a análise dos 4 catálogos escolhidos a fim de identificar e selecionar as informações necessárias para representar as refatorações. Em um segundo momento essas informações foram mapeadas para a ontologia proposta utilizando o Protégé. Por fim, foram instanciadas algumas refatorações na ontologia gerada, dessas há, pelo menos, uma representação, ou seja, uma refatoração de cada catálogo.

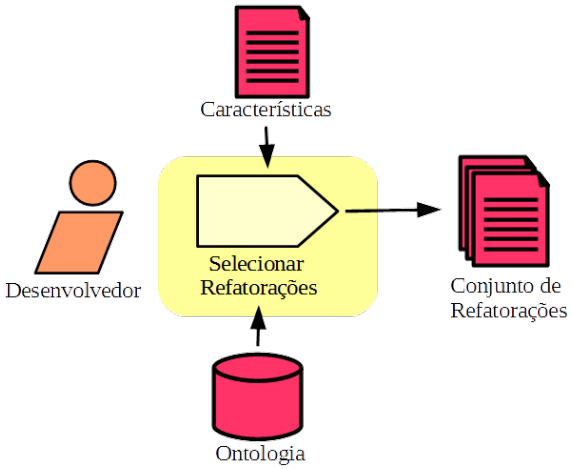


**Figura 2. Relação dos artefatos da atividade Selecionar Refatorações**

### 3.1. Visão Geral

A Figura 2 demonstra um modelo com a estrutura e o relacionamento entre os artefatos da atividade Selecionar Refatorações, a qual a ferramenta desenvolvida auxiliará. Inicialmente o desenvolvedor deve seleciona algumas características, como vantagens e domínio, das refatorações que poderão ser aplicadas no sistema de software. Tais características são apresentadas e buscadas de acordo com os dados mapeados e informados na ontologia proposta. A seção 3.3 mostrará com detalhes a ontologia. Após informar tais características a ferramenta seleciona um possível conjunto de refatorações a serem aplicadas naquele domínio mostrando algumas informações, a referência dos catálogos e alguns exemplos de como aplicar tais refatorações nos sistemas. A ferramenta desenvolvida ainda não atua nas buscas por oportunidades de refatorações e nem nas suas aplicações. A responsabilidade pela execução destas atividades fica a critério do desenvolvedor.

### 3.2. Análise dos Catálogos Escolhidos

A fim de selecionar as informações que são essências para representar o domínio da refatoração de software em uma ontologia, foram utilizados quatro catálogos. Três desses catálogos estão contidos em livros consolidados no meio acadêmico e empresarial e um desses catálogos propões refatorações para usufruir dos benefícios do Java 8.

A obra de Fowler et al. (1999) é um dos primeiros catálogos de refatoração a ser formalizado. Além de discutir as várias técnicas da refatoração ele fornece um catálogo com mais de 70 refatorações. Essas refatorações possuem instruções que indicam quando elas devem ser aplicadas. Elas podem ser utilizadas em qualquer linguagem orientada a objetos, como Java, C# e C++. Algumas melhorias que se obtém aplicando essas refatorações são: legibilidade, manutenibilidade e reusabilidade [Fowler et al. 1999].

Kerievsky (2008) descreve um catálogo de 27 refatorações direcionadas a padrões de projetos utilizando exemplos de casos reais, experiências vividas pelo autor, para aplicá-las e exemplificá-las. Ele utiliza, além do raciocínio lógico, algumas das refatorações propostas por Fowler et al. (1999) para implementar os padrões de projeto, como por exemplo, a refatoração substituir construtores por métodos de criação faz o uso das refatorações extrair método, mover método e internalizar método. Demonstra também múltiplas maneiras de implementar o mesmo padrão e quando usá-los [Kerievsky 2008].

Harold (2009), em sua obra, demonstra um conjunto de refatorações que tem por objetivo melhorar os projetos de aplicações web. Essas refatorações procuram melhorar a confiabilidade, o desempenho, a usabilidade, a segurança, a acessibilidade, a compatibilidade e o posicionamento nos sistemas de busca, potencializando os benefícios e minimizando os esforços [Harold 2009].

Júnior (2014) descreve um conjunto de 20 refatorações, incluindo refatorações inversas, direcionadas as novas funcionalidades da linguagem Java. Tais refatorações estão relacionadas as expressões lambda e tem como principal objetivo atualizar os sistemas de software implementados nas versões anteriores do Java para o Java 8 [Júnior 2014].

Analisando os catálogos selecionados verificou-se que eles utilizam como base a estrutura que Fowler et al. (1999) propôs e utilizou para descrever suas refatorações. Essa estrutura deve conter um nome, um resumo, uma motivação, uma mecânica e um ou mais exemplos de aplicação. Uma característica relevante é a possibilidade de uma refatoração conter inversas, por exemplo, nas refatorações propostas por Júnior (2014) existem refatorações para atualizar o código das versões anteriores do Java para Java 8 e existe as refatorações inversas, que realizam a desatualização do código. Kerievsky (2008) propõe algumas mudanças nessa estrutura, como, deixar explícito as vantagens e desvantagens de aplicar uma determinada refatoração, Fowler et al. (1999) demonstra isso implícito na motivação de determinada refatoração. A estrutura de catálogo utilizado por Harold (2009) e Júnior (2014) é idêntica a proposta por Fowler et al. (1999), ou seja, não possui modificações na sua estrutura.

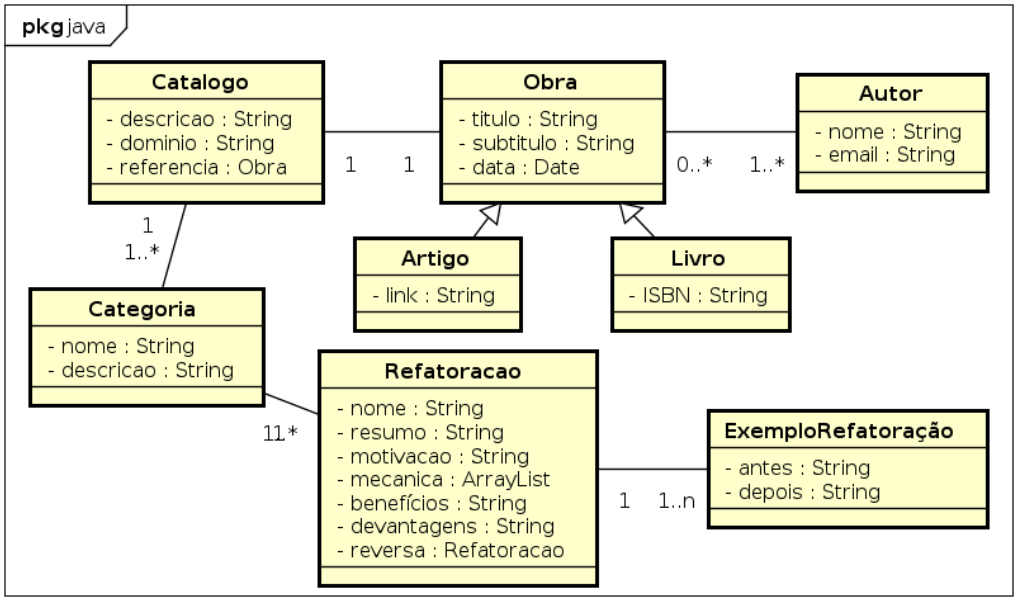


**Figura 3. Diagrama de Classe para representar o domínio da refatoração.**

Após realizada a análise dos catálogos selecionados, foi desenvolvido um diagrama de classe, no Astah Community [Astah 2015], a fim de organizar e compreender as informações que serão mapeadas para a ontologia. Na Figura 3 podemos observar o diagrama de classe que representa o domínio das refatorações. Ele possui as informações consideradas importantes e que devem ser mapeadas referentes as refatorações e as obras que elas estão contidas.

Como observado na Figura 3, as obras possuem um catálogo, o qual possui um determinado domínio. Por exemplo, Kerievsky (2008) descreve um catálogo de padrões para linguagens orientadas a objetos. Cada catálogo possui várias categorias. Elas, por sua vez, possuem um conjunto de refatorações que se assemelham, de alguma forma, quanto as modificações que serão realizadas no sistema de software. Por exemplo, uma das categorias que Kerievsky (2008) utiliza é a criação, o conjunto de refatorações desta categoria têm como alvo alguns dos problemas mais comuns de criação que vão desde construtores extremamente complexos até *singletons* desnecessários.

### 3.3. Ontologia Proposta

Após desenvolver o diagrama de classe citado na seção anterior, começou-se a desenvolver a ontologia proposta no Protégé 3.4.8. No momento que este projeto foi desenvolvido o Protégé encontrava-se na versão 5.0, porém, utilizou-se uma versão mais antiga pois ela suporta todos os recursos utilizados para representar a ontologia e demonstrou ser mais simples e amigável para realizar tais operações, por exemplo, criação das classes, criação dos relacionamentos e instanciação das refatorações. O Protégé é uma ferramenta *open source* com a finalidade de facilitar a construção de ontologias. É mantido pela Universidade de Stanford no *Stanford Center for Biomedical Informatics Research* (BMIR) desde 2000, ano de sua primeira versão, até os dias atuais [Protégé 2015].

A Figura 4 demonstra o grafo gerado pela ferramenta Protégé utilizando o *plugin Jambalaya* [Jambalaya 2015]. Podemos observar as classes e seus relacionamentos que são utilizados para representar o conhecimento sobre refatoração. As classes utilizadas são: *Advantage*, *Author*, *Catalog*, *Category*, *Disvantages*, *Domain*, *Example*, *Refactoring*, *Work*, *Article* e *Book*. As classes *Article* e *Book* são filhas da classe *Work* e por não possuírem relações com outras classes elas foram suprimidas deste diagrama.

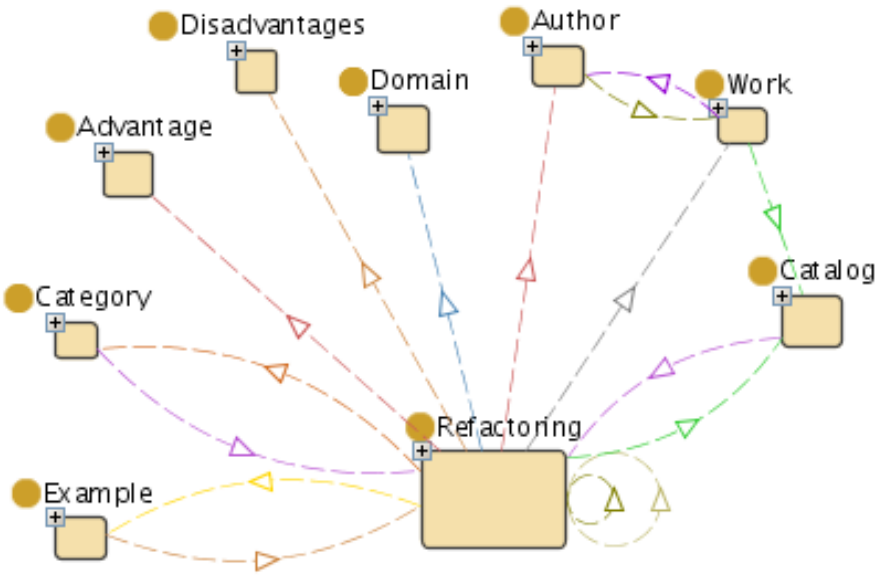


**Figura 4. Ontologia proposta para representar as refatorações.**

Fazendo um comparativo entre as Figuras 3 e 4, observa-se algumas diferenças. Uma delas é a quantidade de classes, a ontologia possui um total de 11 classes, 3 a mais

do que o diagrama de classes. Essa modificação se faz necessária para melhor representar o conhecimento das refatorações. As três classes adicionadas são: *Advantage*, *Disvantages* e *Domain*, estas eram representadas como propriedades das classes Refatoração e Catálogo, respectivamente.

Dentre as classes representadas pela ontologia, a mais relevante é *Refactoring*. Ela interliga todas as informações deste domínio e será utilizada como base exemplificar as relações, instâncias e propriedades. Nas ontologias há possibilidade de descrever alguns tipos de dados para as nossas classes. A Tabela 1 demonstra todos os atributos utilizados para a classe *Refactoring*. Da esquerda para direita, a primeira coluna contém o nome do atributo e a última sua descrição.

**Tabela 1. Os tipos de dados e suas respectivas descrições da classe *Refactoring***

| Tipo de dado | Descrição |
|---|---|
| *Resume* | Descreve um breve resumo da refatoração a ser aplicada |
| *Mechanics* | Descreve uma sequência de passos para aplicar a refatoração |

Quanto as relações, nota-se que elas formam grafos que interligam as classes, diferente das relações representadas no diagrama de classes. A classe *Refactoring* possui relações diretas com todas as outras classes, exceto as classes *Article* e *Book*, que são relacionadas através da classe *Work*. Cada uma dessas relações está descrita na Tabela 2 onde, da esquerda para a direita, a primeira coluna contém o nome das relações, a segunda possui as classes que podem ser relacionadas e na última a descrição desta relação.

**Tabela 2. As relações e suas respectivas descrições da classe *Refactoring***

| Relação | Classe Relação | Descrição |
|---|---|---|
| *hasCatalog* | *Catalog* | Refatoração dever estar em um catálogo |
| *hasExample* | *Example* | Refatoração possui um ou mais exemplos de aplicação |
| *hasCategory* | *Category* | Uma refatoração pode pertencer a uma categoria |
| *hasReverse* | *Refactoring* | Uma refatoração pode possuir uma refatoração inversa |
| *createBy* | *Auhtor* | Uma refatoração é descrita por um autor |
| *hasDesvantagens* | *Desvantagens* | Representam as desvantagens em aplicar essa refatoração, caso existir |
| *hasWork* | *Work* | Em qual obra que essa refatoração está descrita, esta pode ser um Article ou Book |
| *hasAdvanteges* | *Advantages* | Representam as vantagens em aplicar essa refatoração |
| *hasDomain* | *Domain* | Qual(is) o(s) domínio(s) que esta refatoração pertence |
| *useOtherRefactorings* | *Refactoring* | Quais as outras refatorações que ela utiliza. |

Após a implementação da ontologia ser implementa no Protégé, foram incluídas algumas refatorações. Sendo assim buscou-se incluir pelo menos uma refatoração, com suas respectivas informações, de cada catálogo na ontologia proposta. Como as refatorações seguem a mesma estrutura conclui-se que todas elas poderão ser incluídas nesta ontologia. O objetivo deste trabalho não era a quantidade de refatorações inseridas, mas sim a quantidade de refatorações possíveis de serem representadas. Dessa forma, considerou-se essa amostra suficiente para validar a ontologia proposta e realizar as buscas pelas refatorações. Futuramente as refatorações que ainda não foram incluídas serão ser adicionadas a fim de aumentar o conjunto de refatorações da ontologia.

## 4. Realizando as Buscas

Após implementar e popular a ontologia com algumas refatorações buscou-se maneiras de realizar consultas nesta ontologia. Uma das formas de realizar buscas em uma ontologia OWL é utilizando a SPARQL (*SPARQL Protocol and RDF Query Language*). Ela é uma linguagem utilizada para realizar consultas e manipula documentos RDF (*Resource Description Framework*). Em 21 de março de 2013 a SPARQL 1.1 tornou-se oficialmente recomendado pela W3C [SPARQL 2015].

Inicialmente fez-se o uso do *plugin SPARQL Query* presente na ferramenta Protégé para realizar as consultas. Observa-se um exemplo na Figura 5. Neste exemplo é feita a seleção da refatoração, do domínio, do nome do autor e sua refatoração inversa, se ela existir.

```
1  SELECT ?refactoring ?domain ?name_author ?reverse
2  WHERE {
3    ?refactoring  a           :Refactoring.
4    ?refactoring  :hasDomain  ?domain.
5    ?refactoring  :createBy   ?author.
6    ?author       :name       ?name_author.
7    ?domain      rdfs:label  ?l  FILTER REGEX(?l, "lambda", "i").
8    OPTIONAL{ ?refactoring :hasReverse ?reverse. }
9  }
10 ORDER BY ?name_author
```

**Figura 5. Exemplo de consulta SPARQL.**

Para isso inicia-se buscando todas as instâncias da classe *Refactoring*, como demonstrado na linha 3 da Figura 5. Posteriormente busca-se os domínios destas instâncias bem como seu respectivo autor e nome, como demonstrado nas linhas 4, 5 e 6, respectivamente. Na linha 7 realiza-se um filtro nas ocorrências utilizando a função *Filter Regex* do SPARQL. Ela possui três parâmetros a serem informados, são eles: variável onde o filtro será aplicado, expressão regular a ser aplicada, neste caso, variáveis que contém a palavra ”lambda”, e algumas opções extras, neste caso utiliza-se a opção ”i”. Esta opção serve para ignorar a diferença entre letras maiúsculas e minúsculas. Por fim busca-se, opcionalmente, as refatorações inversas, demonstrado na linha 8. Caso não existir inversa a ocorrência continuará nos resultados da consulta, porém, o conteúdo da coluna estará vazio.

A figura 6 demonstra o resultado das ocorrências encontradas quando a consulta da figura 5 é executada. Para esta consulta foram encontradas duas ocorrências ambas do catálogo proposto por Júnior (2014).

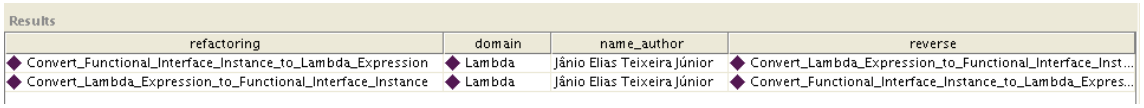


**Figura 6. Ontologia proposta para representar as refatorações.**

Para realizar as consultas no Protégé, o especialista de domínio, que seleciona o conjunto de refatorações a serem aplicadas em um sistema de software, precisa conhecer: (i) a linguagem de busca SPARQL e (ii) a ontologia proposta. A fim de poupar esforços e facilitar a busca pelas refatorações, foi desenvolvida uma aplicação Java utilizando a API Apache Jena para ler e realizar as consultas na ontologia proposta [Jena 2015]. Jena é uma API *open-source* para linguagem Java cujo objetivo é auxiliar o desenvolvimento de aplicações e ferramentas que façam uso de *Semantic Web* e *Linked-Data*. Foi criada por pesquisadores da HP Labs no ano de 2000 e no ano de 2010 a equipe integrou-se à *Apache Software Foundation* [Jena 2015].

A Figura 7 demonstra a interface da aplicação desenvolvida. Observa-se que ela possui 3 colunas na parte superior. Da esquerda para a direita, a primeira delas seleciona-se quais são as melhorias que se deseja buscar, a segunda coluna seleciona-se um ou mais domínios que deseja-se filtrar e a última coluna possui o botão *Search* para realizar as consultas de acordo com os parâmetros selecionados. O botão *Get Example*, quando clicado, mostra os exemplos da aplicação da refatoração selecionada na tabela abaixo. Essa informação é demostrada através de um *pop up*. A opção *Revese* que serve para filtrar as buscas por refatorações que possuem, ou não, inversas.

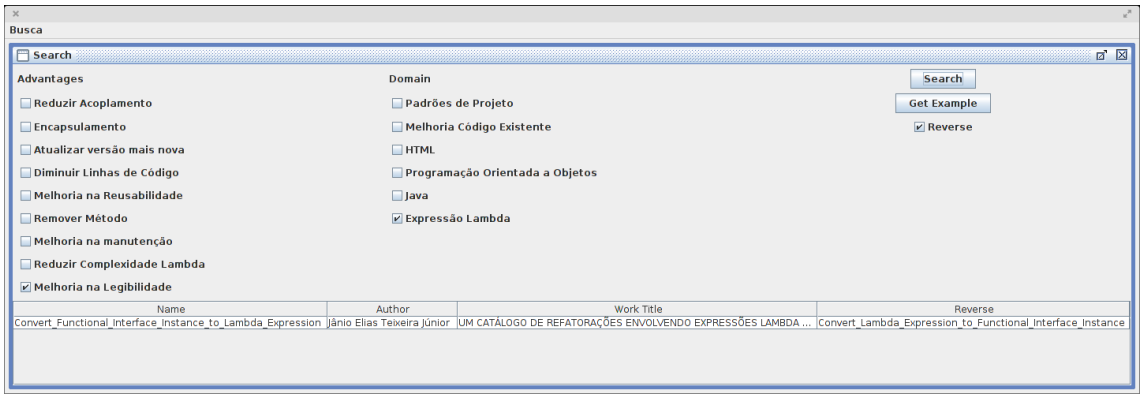


**Figura 7. Interface da aplicação desenvolvida em Java para realizar as consultas**

A aplicação facilita as buscas, porém, a informação mostrada ainda está incompleta quando comparada com a ontologia proposta. Algumas informações que foram mapeadas não estão sendo mostradas nos resultados da busca, como por exemplo, qual obra que contém a refatoração, seu resumo, sua mecânica, o catálogo que ela pertence, quais as outras refatorações que ela utiliza e quais as desvantagens de aplicar essa determinada refatoração. Essas opções foram planejadas porém ainda não estão sendo mostradas ficando como melhorias para uma próxima versão da ferramenta. Tanto a ferramenta, que foi desenvolvida no eclipse, quanto a ontologia proposta, que foi implementada no Protégé, podem ser localizadas no endereço https://github.com/fcampagnolo/OWLRefactoring. Elas estão disponíveis para serem uti-

lizadas, estudadas, modificadas e aperfeiçoadas.

## 5. Trabalhos Relacionados

Alguns trabalhos relacionados a refatoração e ontologia foram utilizados como inspiração para desenvolver o este trabalho. Um deles foi o trabalho de Cantarelli (2012). Ele definiu uma ontologia para representar a estrutura de um programa OO a fim de buscar por oportunidades de refatorações nesse sistema. Para isso ele mapeia, do sistema para a ontologia, as seguintes informações: anotações, classes, atributos, exceções e métodos. Tendo o programa representado na ontologia ele realiza buscas utilizando o SPARQL, *plugin* nativo do Protégé, para buscar oportunidades de refatoração neste sistema de software [Cantarelli 2012].

Outro trabalho utilizado foi o desenvolvido por Gerd Groner and Staab (2010). Ele selecionou um conjunto de refatorações propostas por Fowler et al. (1999), formalizou os algorítimos de busca para ontologias e aplicou-os em duas ontologias, a DOLCE Lite Plus ontology e a Bio-medical ontology OBI 3. Como resultado foram encontradas diversas oportunidades de refatoração [Gerd Groner and Staab 2010]

Ostrowski (2008) também utiliza as refatorações propostas por Fowler et al. (1999) como base para realizar buscas por oportunidades de refatorações nas ontologias. Ele definiu algumas regras para OWL-DL utilizando Jena API. Essas regras buscam remover propriedades, relações e instancias desnecessárias, inferir novas regras e novos dados [Ostrowski 2008].

Observando os esses trabalhos, que foram os mais relevantes para esta pesquisa, observa-se que nenhum deles tem como objetivo representar as refatorações e sim utilizar técnicas de refatoração para melhorar e atualizar as ontologias ou programas existentes. O principal objetivo deste trabalho foi, justamente, representar o domínio de conhecimento sobre refatoração através de uma ontologia OWL e propor uma ferramenta para facilitar a seleção das refatorações.

## 6. Conclusão

Este trabalho apresentou a proposta de uma ontologia para representar um domínio específico, a refatoração. Para isso quatro obras consolidadas no meio acadêmico foram analisadas, são elas: Fowler et al. (1999), Kerievsky (2008), Harold (2009) e Júnior (2014). Verificou-se que todas elas seguem a estrutura de refatoração proposta por Fowler et al. (1999), a qual cada refatoração possui: nome, resumo, motivação, mecânica e exemplos. Kerievsky (2008), propôs algumas alterações nesta estrutura. Uma delas é a adição de vantagens e desvantagens para cada refatoração.

O objetivo principal deste trabalho era representar o domínio do conhecimento da refatoração através de uma ontologia e utilizá-la como base de consulta para facilitar a buscar e seleção de um conjunto de refatorações a serem aplicadas em um sistema de software. Conclui-se que este objetivo foi alcançado no momento em que foi possível representar um conjunto significativo de refatorações na ontologia proposta. Essa foi utilizada como base de consulta na ferramenta desenvolvida, a qual facilita e simplifica as buscas por refatorações, reduzindo os esforços na atividade de selecionar o conjunto de refatorações a ser aplicado no sistema de software. Outro beneficio desta ferramenta é

que o usuário não precisa ser especialista na área de refatoração para selecionar um conjunto de refatorações. Uma vez que as refatorações estiverem representadas na ontologia, o usuário apenas seleciona quais são as características que ele deseja melhorar em seu sistema de software. A ferramenta buscará e apresentará as possíveis refatorações que poderão ser utilizadas para melhorar tais características.

Fica como trabalhos futuros a possibilidade de desenvolver uma funcionalidade que seja capaz de ler um arquivo de refatorações e as adicione automaticamente na ontologia. Um dos motivos desta amostra não ser muito grande é o custo de tempo necessário para transcreve-las manualmente dos livros para a ontologia. Como foi possível adicionar pelo menos uma refatoração de cada catálogo, certamente as outras poderão ser adicionas na ontologia sem a necessidade de modificá-la. Outra oportunidade é ampliar as opções de buscas e visualização das refatorações na ferramenta desenvolvida, as quais podem ser adicionadas desvantagens, obras e autores. Por fim, há possibilidade expandir a ontologia para que, através dela, possamos buscar por oportunidades de refatorações em sistemas de softwares, ou seja, desenvolver uma ferramenta integrada a uma IDE (*Integrated Developmente Environment*) que seja capaz de mapear os programas e, utilizando informações da ontologia, buscar oportunidades de refatorações nos sistemas.

## Referências

Astah (2015). Astah community. `http://astah.net/editions/community`. Acesso em: 15 dez. 2015.

Cantarelli, G. S. (2012). Um processo para o uso de linguagens de consulta em código fonte. Master's thesis, Universidade Federal de Santa Maria-RS, Brasil, Cidade Universitária, Bairro Camobi, Santa Maria - RS, Brasil.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Gerd Groner, F. S. P. and Staab, S. (2010). Semantic recognition of ontology refactoring. *The International Semantic Web Conference*.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Technical Report KSL 92-71*, pages 199–220.

Guarino, N. (1998). Formal ontology and information systems. *International Conference on Formal Ontology in Information Systems*, pages 3–15.

Harold, E. (2009). *Refatorando HTML. Como Melhorar o Projeto de Aplicações Web Existentes - Em Portuguese do Brasil*. Bookman.

Jambalaya (2015). Jambalaya overview. http://www.thechiselgroup.org/jambalaya/. Acesso em: 15 dez. 2015.

Jena, A. (2015). Apache jena: Java framework. `https://jena.apache.org/`. Acesso em: 12 dez. 2015.

Júnior, J. E. T. (2014). Um catálogo de refatorações envolvendo expressões lambda em java. Master's thesis, Universidade Federal de Santa Maria-RS, Brasil, Cidade Universitária, Bairro Camobi, Santa Maria - RS, Brasil.

Kerievsky, J. (2008). *Refatoração Para Padrões - Em Portuguese do Brasil*. BOOKMAN - GRUPO A.

Mens, T. and Tourwe, T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering*, 20:126–139.

Opdyke, W. (1992). *Refactoring Object-oriented Frameworks*. PhD thesis, University of Illinois.

Ostrowski, D. A. (2008). Ontology refactoring. *ICSC 08 Proceedings of the 2008 IEEE International Conference on Semantic Computing*, pages 476–479.

OWL (2015). Owl 2 web ontology language document overview (second edition). http://www.w3.org/TR/owl2-overview/. Acesso em: 15 dez. 2015.

Piveta, E. K. (2009). *Improving the Search for Refactoring Opportunities on Object-oriented and Aspect-oriented Software*. PhD thesis, Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação.

Protégé (2015). About protégé. http://protege.stanford.edu/. Acesso em: 15 dez. 2015.

SPARQL (2015). Sparql 1.1 query language - w3c recommendation 21 march 2013. http://www.w3.org/TR/sparql11-query/. Acesso em: 15 dez. 2015.

Stellman, A. and Greene, J. (2013). *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*. O'Reilly Media.

W3C (2015). Comunidade word wide web. `http://www.w3c.br/Home/WebHome`. Acesso em: 20 mai. 2015.