UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL**

VICTOR RONCHI GARCIA

ANÁLISE DE COBERTURA EM ARQUIVOS RDF

Bandeirantes
2014

Victor Ronchi Garcia

ANÁLISE DE COBERTURA EM ARQUIVOS RDF

Trabalho de Conclusão de Curso submetido a Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. José Reinaldo Merlin

Bandeirantes

2014

Victor Ronchi Garcia

Análise de cobertura em arquivos RDF

Trabalho de Conclusão de Curso submetido a Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____
Prof. Me. José Reinaldo Merlin
UENP – *Campus* Luiz Meneghel

_____
Prof. Me. Rodrigo Tomaz Pagno
UENP – *Campus* Luiz Meneghel

_____
Prof. Wellington Della Mura
UENP – *Campus* Luiz Meneghel

Bandeirantes, 13 de junho de 2014

Dedico este trabalho aos meus pais que
nunca mediram esforços para eu poder estudar.

AGRADECIMENTOS

Primeiramente agradeço a Deus por ter me dado a vida, a oportunidade e as condições para conquistar esse sonho que trago desde criança.

Agradeço aos meus pais Vilson Garcia e Fátima Ronchi, heróis que apesar de todas as dificuldades me deram apoio, incentivo nas horas difíceis, de desânimo e cansaço.

Ao meu irmão Vinícius R. Garcia, pela força e pela vibração em relação a esta jornada.

Ao professor Me. José Reinaldo Merlin, pela paciente orientação, apoio e confiança.

Aos professores da banca avaliadora Wellington Della Mura e Rodrigo Tomaz Pagno, pelas correções e incentivos.

Aos colegas de sala, em especial Elielson Barbosa, Fernando Mossatto, Paulo César Barbieri e Guilherme Tavares Basseto, que tive a honra de formar grupos para a realização das atividades e dos trabalhos decorrentes do curso.

Aos amigos da empresa Setup Júnior, na qual tive o prazer de conhecer e compartilhar grandes momentos de alegria e de aprendizado.

Por todos os professores do Centro de Ciências Tecnológicas da UENP, em especial os professores Carlos Eduardo Ribeiro, Daniela de Freitas Guilhermino Trindade e Ederson Marcos Sgarbi pelos trabalhos desenvolvidos e os ensinamentos transmitidos durante o período de graduação.

A todos os funcionários da UENP que direta ou indiretamente fizeram parte da minha passagem pela universidade.

Muito Obrigado!

“Cada sonho que você deixa pra trás, é
um pedaço do seu futuro que deixa de existir”
Steve Jobs

RESUMO

A atividade de teste é reconhecidamente tida como uma fase indispensável à qualidade de software. Para cada tipo de software existente é requerido uma abordagem de teste diferente, pois um software de sistema crítico exige um procedimento de teste diferente de um software para controle de estoque. O tipo de software objeto deste trabalho são os programas que manipulam bases de conhecimento descritas em *Resource Description Framework* (RDF). Uma das atividades realizadas em teste de software é análise de cobertura que consiste em demonstrar de forma destacada quais elementos de código foram exercitados durante os testes. Neste trabalho é apresentado um método para possibilitar a instrumentação de arquivos RDF. Uma vez que é plausível a instrumentação do arquivo RDF torna-se possível realizar a análise de cobertura destes arquivos para auxiliar na atividade de testes de sistemas baseado em conhecimento. Para o experimento foi construído um protótipo de aplicação que tem por objetivo realizar a consulta em uma ontologia e também foi proposto um método para instrumentação e a análise de cobertura. O método proporcionou uma forma para visualizar de forma destacada quais partes da ontologia foi consultada com um determinado caso de teste.

Palavras chave: Teste de software. Análise de cobertura. RDF.

ABSTRACT

The testing activity phase is admittedly regarded as a prerequisite for software quality. For each type of existing software diferente testing approach is required, because a critical system software requires a testing procedure other than a software for inventory control. The type of software object of this work are programs that manipulate knowledge bases described in Resource Description Framework (RDF). One of the activities in software testing is coverage analysis to demonstrate that highlight code elements which were exercised during testing. This work presents a method to allow the instrumentation of RDF files. Since it is plausible instrumentation file RDF becomes possible to perform the coverage analysis of the files to assist in the activity testing of systems based on knowledge. For the experiment we built a prototype application which aims to make the query into an ontology and a method was proposed for instrumentation and coverage analysis. This method provides a way to display emphasized which parts of the ontology was consulted with a particular test case.

Keywords: Software Testing. Coverage analysis. RDF.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Durante a fase de desenvolvimento de software, a preocupação da equipe desenvolvedora é entregar um produto de software livre de erros. A Engenharia de Software (ES) vem apresentando preocupação com a qualidade de software, no entanto, mesmo com todas as técnicas, recursos e padrões disponibilizados pela tal, ainda assim é possível encontrar erros intrínsecos em algoritmos de programas computacionais (NICÁCIO, 2010).

A atividade de teste é uma ação que possui técnicas bem definidas para encontrar os erros contidos no código dos programas, sendo assim, utiliza-se dessa prática para a garantir a entrega de um produto final livre de erros. Para que o procedimento de teste possa ser bem desempenhado, deve-se elaborar critérios de testes próprios para cada tipo de aplicação, pois a execução de teste de um sistema de tempo real, por exemplo, possui critérios de teste diferentes de um sistema de conhecimento.

Os Sistemas de Conhecimento (SCs) são sistemas que manipulam conhecimento representado explicitamente em uma base de conhecimento. Segundo Costa e Silva (2005), uma base de conhecimento procura reunir conhecimento específico de um domínio, obtido por intermédio de um profissional especialista no assunto, ou até mesmo por meio de bibliografias existentes na área.

Existem diversas maneiras de se representar o conhecimento, cada qual com suas formas e padrões para tal finalidade. Merlin (2011) menciona que nos últimos anos a técnica mais utilizadas para representação o conhecimento foram as ontologias. Gruber (1995) explica que uma ontologia é uma descrição de conceitos e relacionamentos que ligam os conceitos uns aos outros. Existem diversas linguagens de representações de conhecimento através das ontologias. A representação em arquivos RDF/OWL é um tipo de representação de conhecimento, que utiliza estruturas da Web Semântica para tal finalidade.

A principal característica que diferem os sistemas de conhecimento para os sistemas tradicionais (como controle de estoque, de pedido, entre outros) pode ser notada facilmente. Um sistema convencional possui toda sua lógica de tomada de decisão incorporada ao código, os softwares baseados em conhecimento utilizam uma base de conhecimento externa à codificação do software para que seja possível realizar as inferências e chegar a uma determinada resposta (MERLIN, 2011).

O desenvolvimento de um Sistema de Conhecimento não é tão trivial quanto o desenvolvimento de um sistema tradicional, pois esses não são centrados em software e sim em conhecimento, mas mesmo assim utilizam programas computacionais para desempenhar as atividades de consulta (MERLIN, 2011).

Há uma grande dificuldade quando se trata de desenvolvimento desses tais sistemas, pois os desenvolvedores muitas vezes não estão acostumados com o desenvolvimento deste tipo de sistema e acabam encontrando dificuldades em saber se o algoritmo que está sendo escrito, está ou não desempenhando as atividades de forma correta. Como todo software é de fundamental importância garantir a qualidade desses sistemas para que não se torne mais um sistema cheio erros impregnados e inutilizáveis.

Na maioria das vezes os desenvolvedores se deparam com perguntas do tipo: “será que o algoritmo está realizando as consultas de forma correta na base de conhecimento? ”, ou até mesmo “o software contempla todas as consultas? ”, essas perguntas demonstram as dúvidas que os programadores encontram durante o desenvolvimento de tais sistemas, por esse motivo a garantia de qualidade de software pode ser violada por não saberem se de fato o sistema está funcionando como planejado.

Com base nesse problema, é proposto um método de análise de cobertura para arquivos RDF/OWL.

1.1 CONTEXTUALIZAÇÃO

Alavi e Leidner (2001) apud Merlin (2011) definem:

> “sistemas de conhecimento como sistemas baseado em tecnologia da informação, desenvolvidos para apoiar processos de criação, armazenamento, recuperação, transferência e aplicação do conhecimento. ”

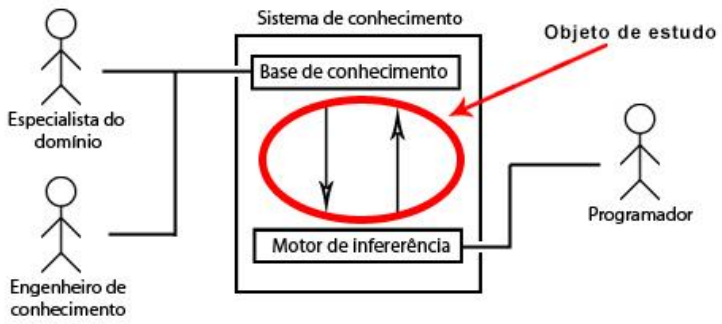
A definição mais usual de um sistema de conhecimento pode ser entendida como sendo um sistema de computador que manipula conhecimento representado explicitamente em uma base de conhecimento.

O desenvolvimento de um sistema de conhecimento é um processo que envolve diversas pessoas com papéis variados a fim de atingir tal objetivo. Podem-se destacar os papéis de programadores, engenheiros do conhecimento e o especialista no domínio. A seguir são descritas as características e responsabilidades de cada um:

a) Especialista do domínio: É a pessoa responsável por disponibilizar o conhecimento do domínio do problema, um exemplo seria de um agrônomo especialista em doenças de cajuzeiro;
b) Engenheiro de conhecimento: Sua responsabilidade é fazer com que todo o conhecimento do especialista do domínio do problema, seja representado em linguagem computacional por meio de formalismos, dando origem a base de conhecimentos; e
c) Programador: Sua função é de escrever algoritmos (muitas vezes chamados de motores de inferência) para que possa realizar consultas na base de conhecimento a fim de atingir uma determinada resposta.

Os programadores podem encontrar algumas dificuldades em escrever tais algoritmos para realizar consultas nas bases de conhecimento, devido os mesmos não serem da área da engenharia do conhecimento e também por não possuir um pleno conhecimento sobre domínio do problema. Muitas vezes os programadores se deparam com situações de não saberem se o algoritmo que está sendo desenvolvido está realizando consultas na base de conhecimento. O objeto de estudo deste trabalho encontra-se no processo de desenvolvimento de um sistema de conhecimento. Na Figura 1, é mostrado em destaque qual parte do processo será desenvolvido o estudo.

Figura 1 - Objeto de estudo do trabalho



Fonte: O autor.

1.2 Formulação e Escopo do Problema

A qualidade de software é um fator crucial que irá definir se tal software pode ser liberado para uso ou não. Atualmente é difícil garantir a qualidade de software, pois as estruturas dos sistemas computacionais estão cada vez mais complexas dificultando ainda mais na hora de garantir a qualidade desejada.

Os sistemas de conhecimento são sistemas que se encontram inseridos nesse contexto, pois manipulam uma base de conhecimento explícito externo ao código fonte do programa.

Durante o desenvolvimento dos sistemas de conhecimento torna-se difícil para o desenvolvedor saber se tal sistema está realizando corretamente as devidas manipulações na base de conhecimento, dificultando ainda mais a garantia de qualidade do software a ser desenvolvido.

1.3 Justificativa

Análise de cobertura em teste de software é importante, pois permite saber o quanto dos elementos requeridos foi exercitado por um conjunto de casos de testes. Desta maneira, pode-se decidir se há a necessidade de testes adicionais ou não. Essa pesquisa se justifica por não se encontrar na literatura estudos específicos sobre análise de cobertura em arquivos RDF.

1.4 Objetivos

O objetivo desse trabalho consiste em investigar por meio do desenvolvimento de um protótipo a possibilidade de realizar a prática de análise de cobertura em ontologias expressas em RDF, a fim de permitir o desenvolvedor saber de fato se o algoritmo que está sendo testado está ou não realizando consultas na base de conhecimento utilizada.

1.4.1 Objetivos Específicos

Os objetivos específicos do trabalho são:

a) Desenvolver um protótipo em linguagem Java de um sistema que realiza consultas em uma base de conhecimentos descrita em RDF;

b) Realizar um estudo bibliográfico sobre análise de cobertura;
c) Desenvolver técnicas de instrumentação de arquivos RDF/OWL;
d) Desenvolver algoritmos para realizar análise de cobertura em arquivos RDF instrumentados; e
e) Analisar e avaliar os algoritmos.

1.5 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado como segue. Na seção 2 é apresentado um levantamento bibliográfico dos principais conceitos da atividade de teste de software, mostrando os principais tipos de teste e a técnica de análise de cobertura, bem como uma visão geral sobre ontologias e linguagens de representação do conhecimento. Também nessa seção é abordada a análise de cobertura em arquivos RDF/OWL, cujo assunto é o foco deste trabalho. Na seção 3 está descrita a metodologia que foi seguida para a realização e conclusão desta pesquisa. Na seção 4 é descrita a fase de desenvolvimento na qual são abordados todos os pontos relevantes do processo de instrumentação e análise de cobertura desenvolvidos. A seção 4 ainda traz consigo os resultados e discussões alcançados por esta pesquisa. Por fim, as considerações finais e trabalhos futuros são discutidos na seção 5.

2 FUNDAMENTAÇÃO TEÓRICA

A Engenharia de Software possui uma grande preocupação em desenvolver um produto livre de erros, no entanto, mesmo utilizando todas as técnicas disponibilizadas pela mesma, é possível que permaneça erros no produto final, e até mesmo tornando-o inutilizável. As atividades de verificação e validação são atividades que visam encontrar os erros intrínsecos ao programa, pois são caracterizadas como sendo da atividade de teste e essas são executadas durante o processo de desenvolvimento.

Quando se fala em teste, deve-se tomar um cuidado muito grande em relação à abordagem de teste, pois é necessário definir as técnicas e os critérios de teste levando em consideração o tipo do sistema que irá ser testado. Por exemplo, para testar um sistema de tempo real, as formas de testes para um sistema desse tipo são diferentes das formas de teste de sistema baseado em conhecimento, pois cada tipo de sistema possui características próprias.

Sistemas baseado em conhecimento possuem um outro fator que dificulta ainda mais a atividade de teste, por serem sistemas que manipulam conhecimento representado explicitamente em um componente externo ao programa denominado base de conhecimento, e muitas vezes o desenvolvedor não consegue ter a certeza de que o seu programa está realizando de forma correta as inferências na base de conhecimento.

Nessa seção, serão apresentados os conceitos da área de teste, descrevendo as modalidades dessa atividade, bem como a prática de análise de cobertura em teste de software. Serão também apresentados conceitos sobre arquivos RDF e sua utilização, e enfim a tarefa análise de cobertura em arquivos RDF.

2.1 TÉCNICAS E CRITÉRIOS DE TESTES

A atividade de teste visa contemplar as possíveis entradas que o usuário poderá inserir no algoritmo. Uma grande vantagem seria de realizar os testes de um programa contemplando todas as entradas possíveis, no entanto é impossível realizar essa atividade, mesmo se tratando de sistemas de pequeno porte (MERLIN, 2011). Devido esse fato, o primeiro passo é definir técnicas e critérios de teste para que de fato garanta um teste de boa qualidade.

As técnicas de testes podem ser interpretadas como um conjunto de passos sistematizados que devem ser seguidos para testar sistemas em computador. Dentre as técnicas existentes, a estrutural e a funcional são as mais utilizadas, cada uma com suas particularidades diferenciando-se na origem da informação utilizada na fase de definir os casos de teste (MALDONADO et al., 2004).

A definição dos casos de teste, nada mais é que selecionar os dados que irão ser utilizados no teste do algoritmo. Existem critérios de testes que visam elaborar casos de testes com objetivo de encontrar erros específicos em determinadas regiões do código a ser testado.

2.1.1 Teste Funcional

O teste funcional (também conhecido como caixa-preta), é uma técnica de teste que não foca nos detalhes de como tal sistema foi implementado, portanto, não importando as estruturas de controle internas ao programa, a grande importância está nos requisitos funcionais do sistema (MALDONADO et al. 2004). Em outras palavras, o teste funcional busca verificar se o programa está ou não executando determinada funcionalidade corretamente.

Para a aplicação da técnica de teste estrutural, deve-se definir casos de testes bem específicos para utilizá-los na tarefa de teste, pois cada “entrada” escolhida tem por finalidade localizar erros em uma parte específica do código. Luft (2012) menciona que existem 3 critérios para definição dos casos de teste sendo eles: particionamento de equivalência, análise do valor limite e grafo de causa-efeito.

2.1.2 Teste Estrutural

Segundo o ISTQB (2011) o teste estrutural (também chamado de caixa-branca) é uma técnica de teste que leva em conta a análise criteriosa da estrutura interna do algoritmo para criar os casos de teste para tal sistema.

Esta técnica tem como principal objetivo avaliar o comportamento interno do algoritmo. Pressman (2011) menciona que o teste estrutural possibilita que o engenheiro de software crie casos de testes para garantir que todos os caminhos independentes do software sejam testados pelo menos uma vez, realizando uma análise minuciosa do código para avaliar a forma de como estão implementadas as

estruturas lógica do software bem como as condições de parada dos laços de repetição entre outros aspectos presentes no código.

2.2 ANÁLISE DE COBERTURA

Segundo Tikir e Hollingsworth (2002), análise de cobertura é uma importante técnica de teste para identificar quais partes do código foram executadas durante a última execução.

A técnica de análise de cobertura tem por objetivo facilitar a tarefa de testadores e desenvolvedores de software durante a atividade de teste, pois permite mostrar as regiões do código que foram exercitadas. Segundo ISTQB (2011) a técnica de análise de cobertura fornece uma medida obtida durante a execução do algoritmo, e com essa medida o testador pode verificar se será ou não necessário realizar testes adicionais tomando como base novos casos de teste.

Muitas são as vantagens em utilizar a técnica de análise de cobertura em teste de softwares, pois além desta técnica poder mostrar para os desenvolvedores quais regiões do código foram executadas, a mesma também fornece uma medida em porcentagem mostrando a quantidade de código que foi exercitada. Merlin (2011) exemplifica que uma cobertura de 80% de comandos, significa que 80% dos comandos de um algoritmo foram executados durante o teste realizado.

Atualmente existem ferramentas que realizam a análise de cobertura para algoritmos desenvolvidos em Java (SAKAMOTO; WASHIZAKI; FUKAZAWA, 2010). Dentre as principais estão Emma, Eclemma, CodeCoverage e Rcov. Na Figura 2 é apresentado um algoritmo com operações matemáticas básicas, onde se têm como entrada 2 números e 1 sinal que indica a operação que realizará o cálculo, este será executado utilizando a técnica de análise de cobertura.

Figura 2 - Exemplo de Código

```
1  import java.util.Scanner;
2  public class Calculadora {
3      public static void main(String []args){
4          float valor1 = 0;
5          float valor2 = 0;
6          float resultado = 0;
7          String sinal_operacao = new String();
8          Scanner tec = new Scanner(System.in);
9          System.out.println("Digite o primeiro valor: ");
10         valor1 = Float.parseFloat(tec.nextLine());
11         System.out.println("Digite o segundo valor: ");
12         valor2 = Float.parseFloat(tec.nextLine());
13         System.out.println("Digite o sinal da operação desejada: ");
14         sinal_operacao = tec.next();
15
16         switch (sinal_operacao){
17             case "+" : resultado = valor1 + valor2;
18             break;
19             case "-" : resultado = valor1 - valor2;
20             break;
21             case "*" : resultado = valor1 * valor2;
22             break;
23             case "/" : resultado = valor1 / valor2;
24             break;
25         }
26
27         System.out.println(resultado);
28     }
29 }
```

Fonte: O autor.

Para as variáveis **`valor1`** e **`valor2`** foram atribuídos os valores 4 e 5 respectivamente, para o **`sinal_operação`** foi escolhido o caractere ‘*’ que indica a operação de multiplicação. Neste ponto, o caso de teste já foi definido e o desenvolvedor já possui uma resposta prévia do que o sistema deverá retornar. Na Figura 3 tem-se o mesmo algoritmo, porém executado com as entradas definidas anteriormente. É possível perceber que algumas linhas estão destacadas em verde. As linhas destacadas em verde são as linhas que de fato foram exercitadas durante a execução do algoritmo. Pode ser percebido que foram executadas as linhas de leitura das entradas, e as linhas que representam o cálculo de multiplicação.

Figura 3 - Exemplo de Análise de Cobertura

Fonte: O autor.

Na Figura 3 são mostradas as partes do algoritmo que foram exercitadas, bem como a quantidade em porcentagem de código executado.

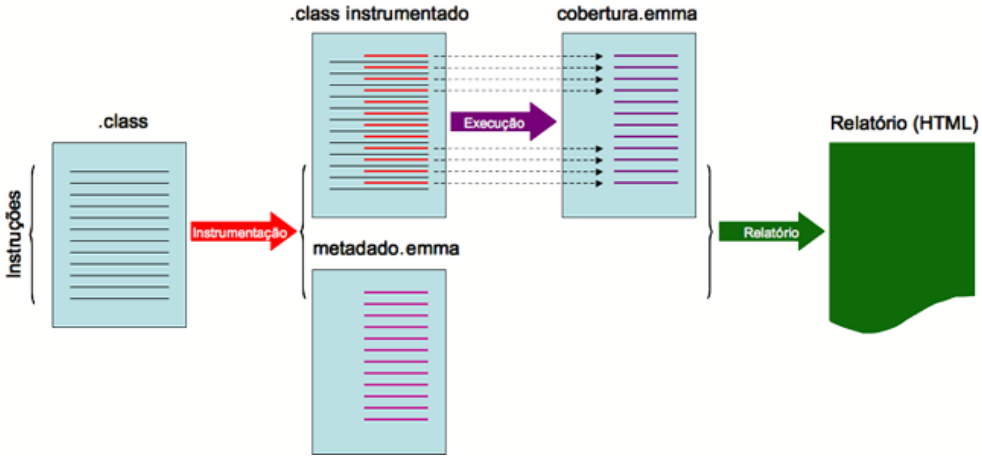
A análise de cobertura durante o teste de software é utilizada para saber quais regiões do código foram exercitadas com determinado conjunto de caso de teste. Supondo que no algoritmo testado anteriormente, o caso que calcula a multiplicação tivesse o operador trocado erroneamente, em vez de ‘*’, seria ‘/’, a

resposta retornada pelo programa seria diferente da resposta prevista pelo testador. Com o destaque das regiões exercitadas no algoritmo, a facilidade de localizar tal erro é maior.

Segundo Tikir e Hollingsworth (2002), o uso de ferramentas para análise de cobertura pode deixar a execução do programa um pouco mais lenta durante a fase de teste, isso se dá pelo fato de as ferramentas de análise de cobertura contarem com técnicas de instrumentação do *bytecode* e gravação dos pontos que o programa foi executado.

Existem uma série de passos a serem seguidos para realizar a instrumentação do *bytecode*, por esse motivo, é possível ver que muitas das ferramentas existentes compartilhem dos mesmos passos para tal finalidade. A Figura 4 tem como objetivo ilustrar de como a ferramenta Emma realiza a instrumentação do *bytecode*.

Figura 4 - Instrumentação do bytecode Emma



Fonte: http://desenvolvimentoagil.com.br/xp/praticas/tdd/emma

Teles (2006) relata que para realizar a instrumentação do *bytecode*, a ferramenta Emma realiza uma cópia do arquivo **“`.class`”**, e logo após realiza a inserção de outros *bytecodes* combinando com os que já existem, são esses *bytecodes* novos que irão apoiar o funcionamento do Emma. O arquivo **“`metadado.emma`”** é um arquivo que tem como finalidade armazenar metadados das classes instrumentadas pois é este arquivo que irá ser usado na elaboração do relatório final em HTML.

Durante a execução, as instrumentações inseridas no **“.class”** irão gravar informações no arquivo **“cobertura.emma”**, são essas informações que indicarão quais foram as instruções exercitadas. O relatório final em HTML é gerado a partir das informações contidas nos arquivos **“metadado.emma”** e **“cobertura.emma”**.

Como citado anteriormente, o processo da técnica de análise de cobertura de um algoritmo torna a execução do mesmo mais lenta devido a sequência de passos para tal objetivo. É de fundamental importância ressaltar que após o algoritmo estar testado e validado, a versão final do código **“.class”** não irá conter nenhuma instrumentação, sendo assim seu tempo de execução passa a ser normal não contendo nenhum atraso de execução.

2.3 ONTOLOGIAS

Como já dito anteriormente, sistemas de conhecimento podem ser definidos como sistemas específicos para a manipulação de conhecimento representado explicitamente. Ontologia é uma das diversas maneiras para se representar o conhecimento explícito.

O termo ontologia segundo Chauí (2003) apud Schiessl e Bräscher (2011) possui suas origens na Filosofia e tornou-se popular na computação devido a necessidade do homem representar o mundo físico (real) no mundo computacional (virtual).

Segundo Studer et al. (1998) “as ontologias tornaram-se comuns no campo da engenharia do conhecimento, principalmente pelo fato de permitir o compartilhamento de conceitos tanto por humanos quanto por computadores. ”

Noy e McGuinness (2001) descrevem que as ontologias podem ser entendidas como uma descrição formal explícita de conceitos de um domínio. A descrição de uma ontologia deve seguir uma estrutura hierárquica a fim de organizar toda a representação do mundo real.

“O papel das ontologias na engenharia do conhecimento é fornecer um vocabulário de termos e relacionamentos com os quais o modelo do domínio é construído” (STUDER et al., 1998). Desta forma, torna-se possível organizar de forma estruturada todo o conhecimento de um domínio específico.

Para que seja possível ter um entendimento pleno sobre ontologias, é necessária uma descrição de alguns conceitos da mesma, sendo eles:

Classe: proporciona um mecanismo para agrupar recursos com características semelhantes (W3C., 2004);

Indivíduo: é considerado um recurso específico, muitas vezes chamado de extensão da classe. Um indivíduo estendido de uma classe é chamado de instância de classe e representa um “objeto” ou uma “coisa” em específico (W3C., 2004);

Propriedade: é uma ligação abstrata entre as classes. As propriedades podem ser divididas em duas categorias, podendo ser, propriedades de objetos que tem por finalidade relacionar indivíduos a indivíduos, e as propriedades de dados que associam indivíduos com valores de dados (W3C., 2004).

Para elucidar esses conceitos Noy e McGuinness (2001) utilizam em seu trabalho uma ontologia para vinhos. A classe é definida como vinho, sendo assim, a classe serve para representar o conjunto de todos os vinhos existentes. Seguindo esse contexto uma ontologia para carros, teria a classe definida carros que englobaria todos os carros existentes.

Uma instância da classe vinho pode ser entendida como um vinho em específico. Segundo Merlin (2011), uma classe pode possuir subclasses, que tem por finalidade representar vinhos mais específicos daquela instância de classe.

Uma instância de vinho deve possuir características que diferem uma das outras, e é por este motivo que existem as propriedades. A instância da classe de vinho chamada de **`Sweet Riesling`** possui o valor **`encorpado`** para a propriedade **`corpo`**.

Segundo Noy e McGuinness (2001), como na programação orientada a objetos, as ontologias também possuem classes, no entanto, enquanto a técnica de orientação a objeto se dá ênfase nas operações da classe, em uma ontologia se dá mais ênfase na estrutura da classe, considerando todos os componentes de marcação para a estruturação da ontologia.

2.4 LINGUAGENS DE REPRESENTAÇÕES

Para que seja possível munir um sistema computacional com o conhecimento de um domínio especifico, se faz necessário fazer a representação do conhecimento do mundo real, para o mundo virtual. Tal representação deve ser descrita seguindo um

padrão de alto nível para que o humano possa entender e, além disso, seguir um padrão para que o computador entenda, é para isso que serve as linguagens de representações.

Existem diversas linguagens de representação para representar o conhecimento na forma computacional, no entanto, este trabalho irá centrar somente nos conhecimentos formalizados na linguagem RDF/OWL.

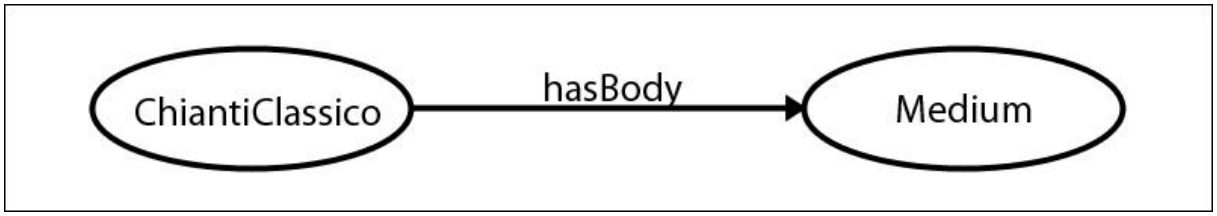
2.4.1 Resource Description Framework

O *Resource Description Framework* (RDF) é um modelo de dados criado pela W3C, e seu objetivo é representar as informações por meio de um conjunto de instâncias chamadas de *Statments*. Segundo W3C (2004), cada *Statment* (também chamado de tripla) é composto de um sujeito *(subject)*, um predicado *(predicate)* e um objeto *(object)*.

Hebeler et al. (2009) apud Merlin (2011) menciona que o sujeito de um *Statment* é a “coisa” que tal tripla descreve; o predicado tem por finalidade descrever a relação entre o sujeito e o objeto. Merlin (2011) cita como o exemplo um *Statment ChiantiClassico hasMaker McGuinesso*, tal tripla foi extraída de uma ontologia de vinhos usada por Noy e McGuinness (2001), e apresenta o significado de que o vinho ChiantiClassico (sujeito) possui como fabricante (predicado) McGuinesso (objeto).

De acordo com W3C (2004), um *Statment* pode ser representado como um grafo orientado onde os nós representam o sujeito e o objeto, e o arco que ligam estes, o predicado. Na Figura 5 é mostrado um exemplo de grafo RDF para representar o *Statment ChiantiClassico hasBody Medium*.

Figura 5 - Exemplo de grafo RDF baseado na notação W3C.



Fonte: Adaptado de Noy e McGuinness (2001).

Como já dito anteriormente, os nós de um grafo RDF representam os sujeitos e os objetos. Os nós de um grafo se distinguem em dois tipos: recursos (*resources*) e literais (*literals*). Os nós do tipo literais representam os valores concretos, como por exemplo números e strings, e podem ser somente nós de objetos. Os nós de tipo recursos equivalem a todos os outros elementos, por este motivo podem ser tanto sujeitos quanto objetos.

Os arcos que ligam os nós são os predicados, também chamados de propriedades, e representam as relações entre os recursos. Na Figura 5, a propriedade *hasBody* conecta o recurso *ChiantiClassico* ao recurso *Medium*.

2.4.2 Linguagem OWL

Web Ontology Language (OWL) é uma linguagem para especificação de ontologias, desenvolvida pela W3C. Segundo Allemang e Hendler (2008), a linguagem OWL oferece melhores formas de estruturar os conceitos semanticamente em relação às primeiras linguagens para tal finalidade. A linguagem OWL foi derivada de linguagens como *Ontology Interface Language* (OIL) e *DARPA Agent Markup Language*.

A linguagem OWL pode ser dividida em três categorias, sendo: *Lite*, *DL* e *Full*, porém, a sublinguagem DL (*Description Logics*) é a mais utilizada (Allemang e Hendler, 2008).

Na W3C (2003) está descrito que a versão *Lite* é usada quando as restrições e hierarquias declaradas são caracterizadas como simples. A versão *DL* é usada quando deseja-se um detalhamento que contempla a completude dos conceitos representados. Por fim a *OWL Full* é usada quando deseja-se um máximo detalhamento dos conceitos, no entanto, é possível estender as sintaxes fornecidas pela linguagem RDF.

A linguagem OWL *Full* pode ser entendida como uma linguagem que incorpora o RDF, pois esta utiliza todo o poder de sintaxe do mesmo. Já as linguagens OWL Lite e OWL Full podem ser vistas como uma versão com menos recursos que a OWL Full, pois estas não estendem a capacidade de sintaxe do RDF.

2.6 ANÁLISE DE COBERTURA EM ARQUIVOS RDF/OWL

O presente trabalho tem como principal objeto de estudo desenvolver uma técnica de análise de cobertura para ser aplicada sobre arquivos RDF, tendo a finalidade de mostrar visualmente para o testador, quais instâncias na ontologia foram consultadas, fornecendo dados para o testador decidir se são necessários testes adicionais ou não.

Na Figura 6 é demonstrado um trecho de código descrito RDF de uma ontologia de vinho usada por Noy e McGuinness (2001). Na Figura 7 é mostrada a hipótese que este trabalho pretende atingir.

Figura 6 - Trecho de código de ontologia (Noy e McGuinness 2001)

```
01 <PetiteSyrah rdf:ID="SeanThackreySiriusPetiteSyrah">
02        <locatedIn rdf:resource="#NapaRegion" />
03        <hasMaker  rdf:resource="#SeanThackrey" />
04        <hasSugar  rdf:resource="#Dry" />
05        <hasFlavor rdf:resource="#Strong" />
06        <hasBody   rdf:resource="#Full" />
07 </PetiteSyrah>
08
09 <Winery rdf:ID="Selaks" />
10
11 <IceWine rdf:ID="SelaksIceWine">
12       <locatedIn rdf:resource="#NewZealandRegion" />
13        <hasMaker  rdf:resource="#Selaks" />
14        <hasFlavor rdf:resource="#Moderate" />
15        <hasBody   rdf:resource="#Medium" />
16        <hasColor rdf:resource="#White" />
17 </IceWine>
```

Fonte: O autor.

Figura 7 - Hipótese de análise de cobertura em ontologia expressa em RDF

```
01 <PetiteSyrah rdf:ID="SeanThackreySiriusPetiteSyrah">
02        <locatedIn rdf:resource="#NapaRegion" />
03        <hasMaker  rdf:resource="#SeanThackrey" />
04        <hasSugar  rdf:resource="#Dry" />
05        <hasFlavor rdf:resource="#Strong" />
06        <hasBody   rdf:resource="#Full" />
07 </PetiteSyrah>
08
09 <Winery rdf:ID="Selaks" />
10
11 <IceWine rdf:ID="SelaksIceWine">
12       <locatedIn rdf:resource="#NewZealandRegion" />
13        <hasMaker  rdf:resource="#Selaks" />
14        <hasFlavor rdf:resource="#Moderate" />
15        <hasBody   rdf:resource="#Medium" />
16        <hasColor rdf:resource="#White" />
17 </IceWine>
```

Fonte: O autor.

O objeto de estudo desta pesquisa centra-se no desenvolvimento de maneiras para realizar a análise de cobertura em arquivos RDF/OWL. Na Figura 7 é demonstrado o que se espera atingir com a presente pesquisa. No exemplo dado, o algoritmo fez uma consulta sobre a ontologia de Vinhos proposta por Noy e McGuinness (2001) seguindo a questão “Qual vinho possui cor branca?”, tal inferência resultou como resposta a instância de vinho **`SelaksIceWIne`**.

3 MÉTODO

Segundo Gil (2008), a pesquisa realizada segundo o ponto de vista da natureza é aplicada, pois tem como objetivo gerar novos conhecimentos práticos para a solução de problemas específicos. Do ponto de vista do problema será seguido uma abordagem qualitativa, por não envolver nenhuma quantificação numérica. Quanto aos fins, a pesquisa a ser realizada é exploratória, pois envolve um levantamento bibliográfico e avaliação das experiências.

Quanto o procedimento técnico trata-se de pesquisa ao mesmo tempo bibliográfica e experimental. Classifica-se como uma pesquisa bibliográfica pois será consultado artefatos literários para a contextualização do tema/problema.

Para o desenvolvimento deste trabalho, as seguintes etapas foram realizadas:

a) Revisão bibliográfica: nesta etapa foi realizado um estudo dos conhecimentos relacionados às áreas relacionadas à este trabalho, que são sistemas de conhecimento, ontologias, análise de cobertura em teste de software;
b) Construção de um protótipo que manipulam ontologias expressas em RDF/OWL;
c) Investigação da possibilidade de análise de cobertura em arquivos RDF/OWL;
d) Elaborar propor uma ferramenta para realizar análise de cobertura em arquivos RDF/OWL;
e) Realizar análise de cobertura utilizando a ferramenta proposta juntamente com o protótipo construído após o levantamento bibliográfico.

4 DESENVOLVIMENTO

Para que fosse possível realizar a investigação de análise de cobertura em arquivos RDF, foi seguido um roteiro no qual cada etapa da fase de desenvolvimento teve escopo e tarefas bem definidas para chegar no resultado final esperado. A fase de desenvolvimento foi desempenhada sob três grandes etapas, sendo elas respectivamente: desenvolvimento de um protótipo para realizar consultas em ontologias, instrumentação do arquivo RDF, e por último, a geração do relatório final em HTML com análise de cobertura.

O início da fase de desenvolvimento é marcado pela construção de um protótipo de aplicação, pois não seria possível investigar o processo de instrumentação do arquivo RDF/OWL sem que o protótipo de aplicação estivesse construído, uma vez que essas etapas possuem dependências umas com as outras.

Nesta seção são descritas as tecnologias que foram utilizadas para o desenvolvimento da presente pesquisa, bem como a construção do protótipo de aplicação, a instrumentação do arquivo RDF/OWL e a geração do relatório final em HTML com análise de cobertura. Ainda é descrito sobre alguns pontos da arquitetura desenvolvida em cada fase, na qual traz diagramas de exemplificação e diagramas da UML 2.0. Também são descritas as abordagens de testes utilizadas para validar cada etapa do desenvolvimento.

4.1 TECNOLOGIAS E FERRAMENTAS UTILIZADAS

Todas as fases do presente trabalho foram desenvolvidas sobre a plataforma J7SE (Java 7 Standart Edition), pois além de ser uma linguagem de programação gratuita, possui grande quantidade de material de referência em fóruns e no próprio site da empresa mantenedora da linguagem. Além disso, encontram-se bibliotecas já desenvolvidas para a construção de sistemas que manipulam o conhecimento expressos em ontologias, como é o caso da biblioteca Jena.

4.1.1 Jena

Para que um algoritmo consiga realizar consultas em uma ontologia é necessário o uso de uma “interface” para possibilitar tal comunicação entre o algoritmo e a ontologia, neste caso, expressa em RDF. Existem diversas bibliotecas escritas em

Java para tal finalidade, no entanto, foi utilizada a biblioteca Jena por ser uma das mais conhecida atualmente.

Segundo Gonçalves e Brito (2004), a biblioteca Jena fornece classes para manipular os conceitos expressos em ontologias na forma de objetos. Uma ontologia em Jena é chamada de modelo, e esta pode ser representado pela classe **`Model.`**

Hebeler et al. (2009) apud Merlin (2011) destacam as principais classes existente na biblioteca Jena, são elas:

Resource: é a classe que tem por finalidade representar um elemento de um recurso de uma ontologia, também chamado de Statement.

Statement: Uma instância da classe Statment pode ser entendida como uma tripla, isto é um objeto contendo **`subject, predicate`** e **`object.`**

Reasoner: Classe usada para realizar consultas na base de conhecimento, pois permite a utilização do motor de inferência.

Merlin (2011) menciona que a biblioteca Jena possui funções para carregar uma ontologia de diferentes formatos, podendo ser de um endereço web, de um arquivo e disco ou até mesmo de um fluxo de dados.

4.2 PROTÓTIPO DE APLICAÇÃO

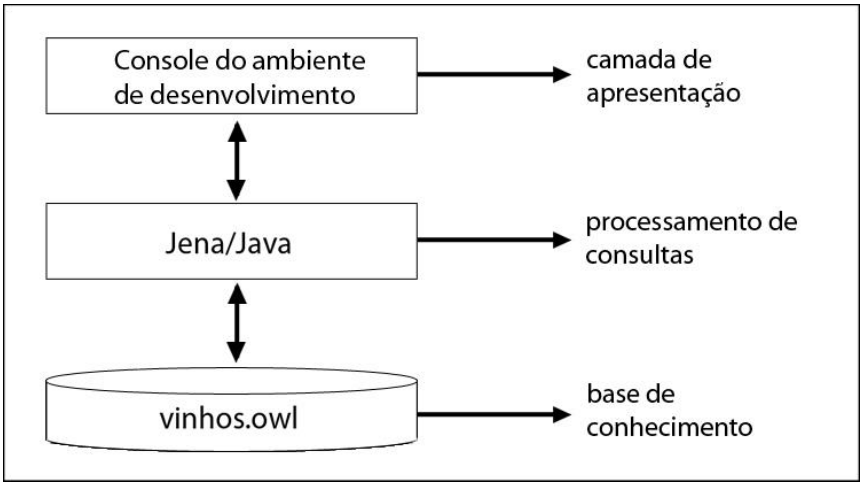
Para investigar a possibilidade de instrumentação do arquivo RDF, inicialmente precisou-se desenvolver uma aplicação com a capacidade de manipular conhecimento representado em ontologia RDF/OWL. Para o desenvolvimento deste protótipo de aplicação, utilizou-se uma ontologia de vinhos proposta por Noy e McGuinness (2001).

Além da ontologia, foram utilizadas bibliotecas e funções descritas em Java para o desenvolvimento do protótipo. A seguir, é abordado sobre a arquitetura do protótipo desenvolvido, suas capacidades e a abordagem de teste utilizada para validar o funcionamento de tal aplicação.

4.2.1 Arquitetura do Protótipo

O protótipo foi desenvolvido para o ambiente *desktop* e foi dividido em 3 camadas principais mostradas na Figura 8.

Figura 8 - Arquitetura do protótipo

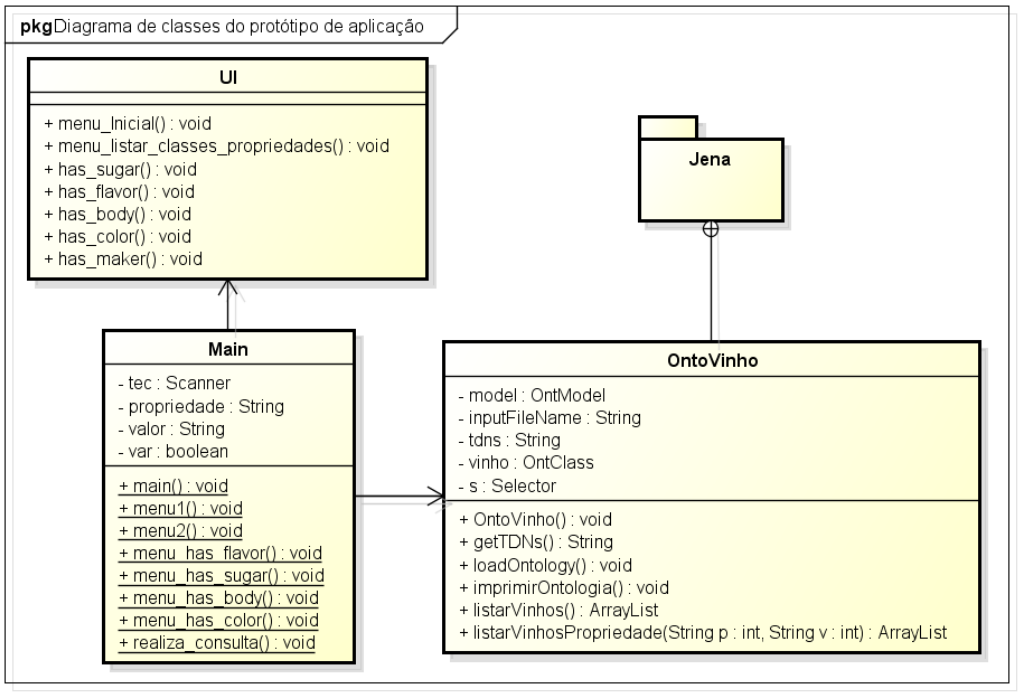


Fonte: Adaptado de Merlin (2011).

A camada de aplicação desenvolvida não contém interface gráfica por esta não ser relevante para o escopo deste trabalho. A interação do usuário com a aplicação ocorre por meio de linha de comando no *console* do próprio ambiente de desenvolvimento, no qual o usuário pode informar ao programa qual o tipo de conhecimento que se quer obter.

Na camada de processamento de consultas (também chamada de camada de domínio), encontram-se as classes Java que fazem uso da biblioteca Jena. Estas são responsáveis por realizar inferências (consultas) na ontologia. Nesta camada também se encontra a classe responsável por carregar a ontologia para o software a partir de um arquivo `.owl`. Na Figura 9 é demonstrado um diagrama da UML 2.0, que contém as classes e as relações entre as mesmas.

Figura 9 - Diagrama de classes do protótipo de aplicação



Fonte: O autor.

Por fim, mas não menos importante, na camada da “base de conhecimento” encontra-se a ontologia de vinho proposta por Noy e McGuinness (2001). A ontologia é representada em um arquivo `.owl`, e este é carregado de um arquivo em disco. Uma vez carregada do disco, o algoritmo passa a manipular a mesma na memória.

4.2.2 Teste do Protótipo

A abordagem de teste para a validação do protótipo desenvolvido foi baseada nas capacidades que tal software possui. Durante a fase de teste do protótipo, o software tinha que estar funcionando de acordo com os comandos do usuário, um exemplo que foi usado para teste do protótipo foi uma questão do tipo “Quais vinhos que possuem o valor branco para a propriedade cor?”, para a aplicação retornar a lista de todos os vinhos existentes na ontologia que possuem tal propriedade e valor.

Após o teste das capacidades do software, a primeira etapa do desenvolvimento foi concluída. Após ter o protótipo de aplicação implementado, testado e funcionando, deu-se início a segunda etapa, na qual o foco era o de realizar a instrumentação do arquivo RDF/OWL. A seguir é descrito sobre o

processo de instrumentação do arquivo RDF/OWL, bem como suas limitações e objetivos.

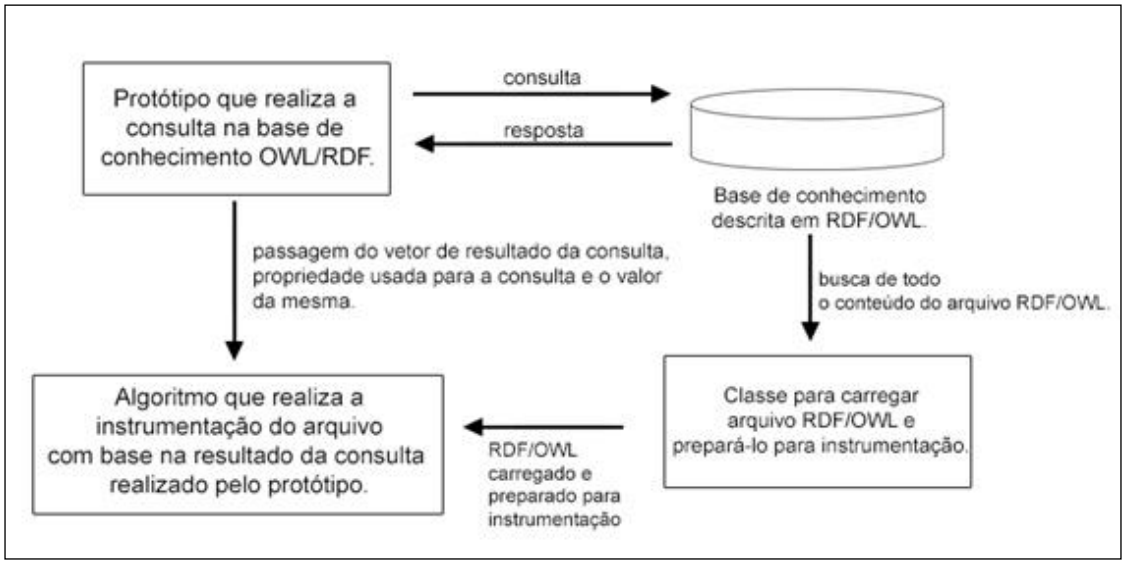
4.3 INSTRUMENTAÇÃO DO ARQUIVO RDF/OWL

Para que seja possível o processo de análise de cobertura em código fonte, primeiro, antes de realizar a tarefa de análise de cobertura se faz necessário a instrumentação de tal código. Na análise de cobertura do arquivo RDF/OWL não é diferente, pois são as instrumentações que irão apoiar o processo de geração do relatório final em HTML e permitirão análise de cobertura.

No entanto, diferentemente da instrumentação de código fonte, que se dá por inserções de código extra no código original do programa, o processo de instrumentação de arquivos RDF/OWL se dá por inserções de *tags* HTML que marcam quais instâncias de classes na ontologia foram consultadas.

A instrumentação do arquivo RDF/OWL é baseada no parâmetro de retorno da consulta realizada pelo protótipo de aplicação descrito no item 4.2 deste trabalho. Uma vez realizada a consulta pelo protótipo, têm-se o nome de todos os vinhos especificados por uma restrição de consulta e são esses nomes que irão apoiar o processo de instrumentação do RDF/OWL. Na Figura 10 é demonstrada como ocorre a interação entre o protótipo de aplicação com o algoritmo que realiza a instrumentação do arquivo RDF.

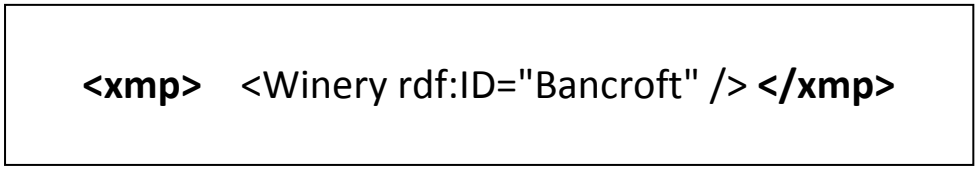
Figura 10 - Processos para instrumentação do RDF/OWL



Fonte: O autor.

O processo de instrumentação é iniciado realizando uma cópia do arquivo RDF/OWL na memória primária do computador, já durante esse processo de cópia para a memória, são adicionadas as *tags* HTML <xmp> e </xmp> em cada linha do arquivo, tendo como finalidade preparar o arquivo para ser instrumentado. Na Figura 11 é demonstrado a concatenação de uma linha original do arquivo RDF/OWL com as devidas *tags* <xmp> e </xmp>.

Figura 11 - Exemplo de concatenação de RDF/OWL com *tags* <xmp></xmps>



Fonte: O autor.

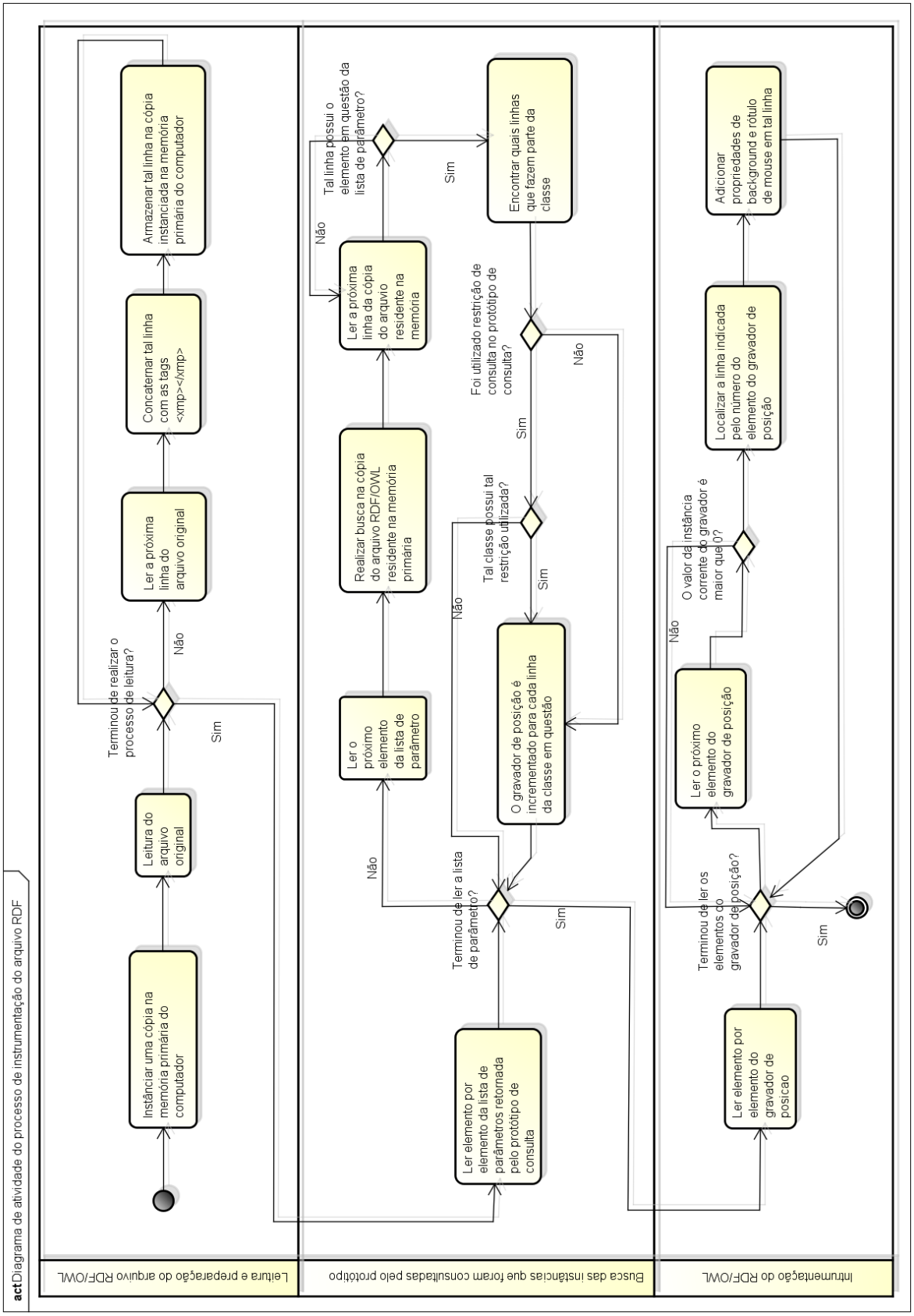
Uma vez que o arquivo está pronto para ser de fato instrumentado, o programa de instrumentação irá buscar as instâncias de classes que possuem nomes iguais aos que foram retornados pela consulta do protótipo de aplicação. Quando encontrado uma instância de classe com mesmo nome, basta verificar se tal instância possui propriedade e valor igual ao que foi utilizado para consulta na ontologia pelo protótipo. Uma vez que tal instância possui propriedade e valor igual, um gravador de posição é incrementado. O gravador de posição é um elemento que irá monitorar quais classes foram consultadas com um determinado caso de teste e irá incrementar todas as vezes que tal classe for encontrada. Esse elemento irá fornecer dados específicos de cada classe para apoiar na instrumentação do arquivo RDF/OWL.

Quando é terminado o processo de busca das instâncias, têm-se no gravador de posição os números das linhas do arquivo RDF/OWL que foram consultadas durante a bateria de testes, possibilitando assim a instrumentação do arquivo RDF/OWL. Após a busca pelas linhas das classes que foram consultadas, é de fato realizado o processo de instrumentação, na qual consiste em realizar troca da cor de fundo das *tags* HTML <xmp></xmp>, de acordo com os dados armazenados pelo gravador. Essa troca de cor de fundo das *tags* tem como finalidade dar o devido destaque das classes que foram consultadas. Para os diferentes destaques das classes do arquivo RDF/OWL foram propostas 5

tonalidades da cor verde para serem usadas no processo de análise de cobertura. Cada vez que uma instância do arquivo RDF/OWL é consultada durante uma bateria de testes, o tom de verde que destaca a instância corrente vai sendo alterada para uma tonalidade de verde cada vez mais escuro tendo como finalidade dar mais ênfase na instância para diferenciá-la das demais. Sendo assim, torna-se possível verificar no relatório final quais foram as instâncias que mais foram consultadas durante a bateria de testes devido o uso da escala de cor. Além da escala de cor do fundo, o gravador também fornece dados para incluir no relatório final uma espécie de rótulo no cursor do mouse, para que o testador possa ter a certeza de quantas vezes tal classe foi consultada. Na Figura 12 é demonstrado um diagrama de atividade, que tem por finalidade elucidar todo o processo de instrumentação do arquivo RDF/OWL. O digrama da Figura 12 é dividido em três raias as quais representam os sub processos da etapa de instrumentação. O processo é iniciado na primeira raia sendo marcado pelo nó inicial. Nesta raia é demonstrado o laço de repetição utilizado para realizar a leitura e a preparação do arquivo RDF/OWL para receber a devida instrumentação.

Na segunda raia são exibidos todos os testes de condições, laços de repetição e os processos que são realizados para encontrar as instâncias que foram consultadas com um determinado caso de teste. Nessa raia também é possível visualizar a existência de um laço de repetição para incrementar o gravador de posição nas linhas que compõe cada classe que foi encontrada. Na terceira raia é possível verificar as etapas que foram seguidas para a instrumentação do arquivo RDF de acordo com os dados do gravador de posição obtidos na raia anterior. O final do processo representado pelo diagrama é demarcado pelo símbolo de atividade final. Na Figura 13 é demonstrado o algoritmo usado para o processo de instrumentação do arquivo RDF/OWL.

Figura 12 - Procedimento realizado para instrumentação de RDF/OWL



Fonte: O autor.

Figura 13 - Algoritmo usado para instrumentação do arquivo RDF/OWL

```
1  procedimento instrumentacaoEcobertura(Lista<Texto> resultados, Texto propriedade, Texto valor) faça
2      inteiro vezes[]; // declara um vetor de inteiro, porém não instancia
3      texto cores[]← novo texto[6]; // instancia um vetor de texto de 6 posições
4      logico condicao  ← falso;
5
6      Texto linha;
7      Texto textoPreparacao;
8      Arquivo arq ← novo Arquivo ('wine.owl'); // carrega a ontologia na memória
9      inteiro tamanho;
10     Texto resultadoCorrente;
11     Arquivo relatorioFinal ← novoArquivo ('saida.html);
12     Texto arquivoTemporario;
13     //gravar os cabeçalhos da página HTML no arquivo de relatório e preparar o arquivo para o
14
15     enquanto (arq) faça // enquanto existir linha sem ler do arquivo arq
16         linha ← arq.linha; // variável linha recebe a linha em questão do arquivo arq
17         arquivoTemporario ← arquivoTemporario + '<xmp>' + linha + '</xmp>~';
18         tamanho ← tamanho + 1;
19     fim para// fim do processo de preparação do arquivo OWL para instrumentação
20
21     se (condicao = falso) faça // entrará uma só vez para instância do contador de posição
22         vezes ← inteiro[tamanho]; //define um tamanho para o vetor de inteiro
23     fim se
24
25     enquanto (lista) faça //enquanto existir elemento na lista recebida como parâmetro faça
26         resultadoCorrente ← lista.linha;
27         //encontrar o bloco da ontologia que possui o mesmo elemento em questão
28
29         se (encontrou = verdadeiro) faça
30             se (valor = vazio e propriedade = vazio) faça
31                 // se é vazio é por que está listando todas as classes da ontologia
32                 // sem uso de restrição
33                 vezes[posição de cada inha]++; // é incrementado para cada linha
34                 // que compõe o bloco da classe em questão
35             senão faça
36                 // verificar se o bloco da classe em questão possui a restrição
37                 se (possuir = verdadeiro) faça
38                     vezes[posição de cada inha]++; // é incrementado para
39                     //cada linha  que compõe o bloco da classe em questão
40                 fim se
41             fim senão
42         fim se
43     fim para // fim de buscas e contagem das classes que foram consultadas
44
45     Texto auxiliar[] = quebrarTexto('~', arquivoTemporario); // é quebrado o conteúdo onde
46     //contém  o caracter '~'. Tal caractere é adicionado na linha 17 deste algoritmo
47
48     //procedimento para instrumentação
49     inteiro indice ← 0;
50     Texto inst;
51     para  indice de 0 até indice = tamanho faça
52         se (vezes[indice] <> 0) faça
53             //adicionar o parâmetro de background e title da tag <xmp>
54             inst ← '<xmp style="background-color:'+cores[vezes[índice]]+'">';
55             auxiliar[índice].substituirCadeia('<xmp>', inst);
56         fim se
57     fim para
58 fim para // fim do método
```

Fonte: O autor.

No processo de instrumentação para análise de cobertura de programas, é realizado a instrumentação do código fonte e após ser gerado o relatório final da análise de cobertura, são removidos os códigos extras de instrumentação, pois tais instrumentações não apoiarão o funcionamento do programa e poderão até mesmo, tornar a execução do programa mais lenta.

Na instrumentação de arquivo RDF/OWL o processo é aplicado de forma diferente, pois após a geração do relatório final não se faz necessário apagar as *tags* de marcação, já que as instrumentações são realizadas em uma cópia do arquivo RDF/OWL. Permanecendo as instrumentações na cópia do arquivo RDF/OWL, torna-se possível visualizar em um editor de texto quais as classes do arquivo RDF/OWL que foram instrumentadas durante a bateria de testes. Na Figura 14 é demonstrado um trecho da ontologia utilizada sem instrumentação. Na Figura 15 é exposto o mesmo trecho da ontologia, porém instrumentado, vale ressaltar que ambos os trechos de ontologia estão descritos em RDF/OWL.

Figura 14 - Trecho de onotlogia RDF/OWL sem instrumentação

```
01 <PetiteSyrah rdf:ID="SeanThackreySiriusPetiteSyrah">
02        <locatedIn rdf:resource="#NapaRegion" />
03        <hasMaker  rdf:resource="#SeanThackrey" />
04        <hasSugar  rdf:resource="#Dry" />
05        <hasFlavor rdf:resource="#Strong" />
06        <hasBody   rdf:resource="#Full" />
07 </PetiteSyrah>
08
09 <Winery rdf:ID="Selaks" />
10
11 <IceWine rdf:ID="SelaksIceWine">
12       <locatedIn rdf:resource="#NewZealandRegion" />
13        <hasMaker  rdf:resource="#Selaks" />
14        <hasFlavor rdf:resource="#Moderate" />
15        <hasBody   rdf:resource="#Medium" />
16        <hasColor rdf:resource="#White" />
17 </IceWine>
```

Fonte: O autor.

Figura 15 - Trecho de ontologia em RDF/OWL com instrumentação

```
01 <xmp> <PetiteSyrah rdf:ID="SeanThackreySiriusPetiteSyrah"> </xmp>
02 <xmp>       <locatedIn rdf:resource="#NapaRegion" /> </xmp>
03 <xmp>       <hasMaker  rdf:resource="#SeanThackrey" /> </xmp>
04 <xmp>       <hasSugar  rdf:resource="#Dry" /> </xmp>
05 <xmp>       <hasFlavor rdf:resource="#Strong" /> </xmp>
06 <xmp>       <hasBody   rdf:resource="#Full" /> </xmp>
07 <xmp> </PetiteSyrah> </xmp>
08 <xmp>
09 <xmp> <Winery rdf:ID="Selaks" /> </xmp>
10 <xmp>
11 <xmp style=”background-color: #a2ffa0;”> <IceWine rdf:ID="SelaksIceWine"> </xmp>
12 <xmp style=”background-color: #a2ffa0;”>       <locatedIn rdf:resource="#NewZealandRegion" /> </xmp>
13 <xmp style=”background-color: #a2ffa0;”>       <hasMaker  rdf:resource="#Selaks" /> </xmp>
14 <xmp style=”background-color: #a2ffa0;”>       <hasFlavor rdf:resource="#Moderate" /> </xmp>
15 <xmp style=”background-color: #a2ffa0;”>       <hasBody   rdf:resource="#Medium" /> </xmp>
16 <xmp style=”background-color: #a2ffa0;”>       <hasColor rdf:resource="#White" /> </xmp>
17 <xmp style=”background-color: #a2ffa0;”> </IceWine> </xmp>
```

Fonte: O autor.

Na Figura 15 é possível ver que a classe do vinho **`SelaksIceWine`** está instrumentada pois a cor de fundo das *tags* <xmp> está com uma cor definida, a instrumentação se deu devido o protótipo de aplicação ter feito uma consulta sobre a ontologia de Vinhos proposta por Noy e McGuinness (2001) seguindo a questão “Qual vinho possui cor branca?”, tal inferência resultou como resposta a instância de vinho **`SelaksIceWIne`**.

Ao final da etapa de instrumentação do arquivo RDF, cabe a aplicação do processo de análise de cobertura e a geração do relatório final em HTML.

4.4 Geração do Relatório Final em HTML e Análise de Cobertura

Como visto anteriormente, o processo de análise de cobertura só é possível após a instrumentação do arquivo RDF/OWL. Uma vez realizada a devida instrumentação em tal arquivo, torna-se possível a geração do relatório final em HTML, bem como a análise de cobertura.

O processo de análise de cobertura em arquivo RDF/OWL consiste em destacar as classes da base de conhecimento que foram consultadas pelo algoritmo de consulta, como descrito no item 2.6 deste trabalho.

Esta etapa marca o fim da presente pesquisa, pois todos os objetivos gerais descritos no início deste trabalho foram alcançados. Abaixo são descritos os principais pontos do processo de geração final do relatório em HTML, bem como a análise de cobertura, mostrando o que foi utilizado para gerar o arquivo e a estratégia de teste utilizada para validar o relatório final em HTML.

O processo de geração do relatório final tem como saída um arquivo descrito em HTML, que traz em seu conteúdo uma cópia instrumentada do arquivo RDF/OWL para permitir a análise de cobertura do mesmo. Para geração do relatório final, utilizou-se um código para criar no disco do computador um arquivo HTML e gravar neste arquivo todas as *tags* da estrutura padrão de uma página HTML juntamente com a cópia instrumentada do arquivo RDF/OWL obtida na fase anterior do presente desenvolvimento.

Ao fim da gravação do relatório no disco do computador, torna-se possível a visualização da análise de cobertura. Como o arquivo é gerado está alocado no próprio computador, basta o testador abrir tal arquivo em um navegador web para

que possa ser exibido. Na Figura 16 é demonstrada parte do relatório final interpretado por um navegador web, no qual é possível visualizar a análise de cobertura.

Figura 16 - Trecho do relatório final em HTML com análise de cobertura

Fonte: O autor.

No momento em que se consegue chegar no relatório final esperado, é necessário realizar verificações em tal relatório, para analisar se de fato contém todas as informações e se tais informações representam a realidade da bateria de teste da última execução.

4.4.1 Análise do relatório final

A abordagem de teste para a análise do relatório final produzido teve como finalidade verificar se de fato as instâncias consultadas pelo protótipo de aplicação, foram destacadas como o esperado.

Para elucidar o processo de validação do relatório, uma vez que foi executado uma bateria de testes, procurou-se cruzar os casos de testes usados durante tal bateria, com o relatório final produzido pelo método de geração proposto, na qual coube verificar se as classes que foram destacadas na ontologia teriam as restrições utilizadas durante a bateria de testes.

Após a validação do relatório final, a última etapa do desenvolvimento foi concluída, na qual foi possível verificar que existe possibilidade de instrumentação do arquivo RDF/OWL, abrindo novos gaps na área de teste em sistemas baseado em conhecimento.

4.5 RESULTADOS E DISCUSSÕES

O desenvolvimento da presente pesquisa centrou-se na busca por maneiras de realizar a instrumentação e a análise de cobertura de arquivos RDF/OWL.

Existem muitas ferramentas para visualizar a análise de cobertura de algoritmos desenvolvidos em Java, dentre as principais ferramentas estão: Emma, Eclemma, CodeCoverage e RCov (Sakamoto; Washizaki; Fukazawa, 2010). A motivação deste trabalho é apoiada por não existir na literatura estudos que tratam sobre a instrumentação de arquivos RDF/OWL para possibilitar a análise de cobertura destes.

No início do desenvolvimento da instrumentação do arquivo RDF/OWL a preocupação era de como seria realizada a análise de cobertura em tais arquivos, uma vez que tratava da necessidade de indexação do arquivo RDF/OWL para poder encontrar as classes consultadas.

Durante buscas por bibliotecas para a indexação de tais arquivos, foram encontradas as bibliotecas Apache Lucene e Apache Solr, ambas incorporavam a função de *highlighter* para implementar a análise de cobertura, porém tal funcionalidade era desenvolvida apenas para busca de cadeia de caracteres na forma de palavra, impossibilitando de realizar a análise de cobertura em classe da ontologia, uma vez que se trata de um bloco de texto. Outra forma encontrada para realizar o processo de análise de cobertura foi a utilização do *highlighter* fornecido pela própria linguagem Java para ser usada em conjunto com o componente JTextFrame. Após ter implementado todo algoritmo de análise de cobertura utilizando o *highlighter* disponibilizado pela linguagem, percebeu-se que acontecia o

processo de análise de cobertura, porém não era de fato realizada a instrumentação do arquivo RDF/OWL. Como não foi encontrado nenhuma solução para auxílio no processo de instrumentação de arquivos RDF/OWL, optou-se por desenvolver um módulo de software que realizasse a cobertura pretendida.

A utilização das *tags* <xmp></xmp> foi uma nobre saída para a instrumentação do arquivo RDF/OWL pois além de oferecer atributos que permitam realizar a análise de cobertura, ainda fornece a uma barreira para o navegador Web não interpretar o conteúdo que se encontra no interior das mesmas respeitando toda endentação original do arquivo RDF/OWL.

Uma vez que foi encontrada a possibilidade de instrumentação de arquivo RDF/OWL com a utilização das *tags* HTML, o trabalho estendeu a funcionalidade de realizar a análise de cobertura seguindo escalas de cores diferentes para identificar a quantidade de vezes que cada classe da ontologia foi consultada, possibilitando até o uso em conjunto com ferramentas de testes automatizados como o JUnit.

5 CONSIDERAÇÕES FINAIS

Teste de software é uma importante atividade para verificar a existência de erros em programas antes de serem liberados para uso. A Engenharia de Software proporciona abordagens de testes para tal finalidade, dentre as mais comuns encontra-se a técnica de teste estrutural e a técnica de teste funcional.

A técnica funcional tem por principal característica não levar em conta a estrutura do código do programa pois irá avaliar somente se o software está mostrando o resultado esperado independente da forma que o software chega em tal resultado. Diferentemente, a técnica estrutural foca nos requisitos estruturais do algoritmo importando de como e quais comandos o programa utilizou para chegar em determinado resultado, vale ressaltar que quando se utiliza a técnica de teste estruturais é necessário elaborar os casos de testes de uma forma mais criteriosa levando em conta a estrutura do programa.

O procedimento de análise de cobertura fornece uma medida em porcentagem para avaliar quantos por cento de um código foi coberto por um determinado caso de teste. Para tornar essa atividade de análise de cobertura são propostas ferramentas automatizadas para realizar essa quantificação de códigos executados bem como destacar os comandos do código que foram exercitados durante a última execução.

Durante testes de Sistemas baseado em Conhecimento podem ser usadas ferramentas para análise de cobertura, porém tais ferramentas irão realizar somente a análise de cobertura no código fonte responsável pelo processamento de consultas na ontologia, impossibilitando verificar se foram realizadas consultas de forma correta na ontologia utilizada, dificultando o testador se são necessários testes adicionais para garantir a qualidade do software produzido em questão.

A investigação realizada por esse trabalho procurou maneiras de realizar o processo de análise de cobertura em ontologias descritas em RDF/OWL. O processo investigatório consistiu em duas partes: instrumentação do arquivo RDF/OWL e geração do relatório final possibilitando a visualização de análise de cobertura.

5.1 TRABALHOS FUTUROS SUGERIDOS

O presente trabalho demonstrou a possibilidade de instrumentação e análise de cobertura em arquivos RDF/OWL. Como trabalhos futuros sugere-se:

- Análise de cobertura em termos de elementos da ontologia, tais como cobertura de sujeitos, cobertura de predicados e cobertura de relacionamentos;

- Implementar meios para fornecer dados quantitativos referentes a cobertura executada em termos de elemento da ontologia; e

- Criação de um algoritmo genérico para instrumentação e análise de cobertura de qualquer outra ontologia que incorpore o modelo RDF/OWL.

REFERÊNCIAS

ALLEMANG, D; HENDLER, J. **Semantic web for the working ontologist.** Burlington: Elsevier Inc., 2008.

COSTA, W. S.; Silva, S. C. M. **Aquisição de conhecimento:** O grande desafio na concepção de sistemas especialistas. Disponível em: <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/viewFile/71/77>. Acesso em: <08/06/2014>. [On-line].

ISTQB. **Glossário standart de termos usados em testes de software.** 2011.

GIL, A. C. **Como elaborar projetos de pesquisa**. 4 ed. São Paulo: Atlas, 2008.

GONÇALVES, P. R.; BRITO, P. F. **Manipulação de uma ontologia desenvolvida em OWL através da utilização da API JENA 2 Ontology.** Palmas: ULBRA, 2004.

LUFT, C. C. **Teste de software:** uma necessidade das empresas**.** Santa Rosa: UNIJUÍ, 2012.

MALDONADO, J. C.; BARBOSA, E. F.; VINCENZI, A. M. R.; DELAMARO, M. E.; SOUZA, S. R. S.; JINO, M. **Introdução ao teste de software.** São Carlos: ICMC/USP, 2004.

MERLIN, José Reinaldo. **Uma abordagem para criação de casos de teste funcionais para sistemas de conhecimento.** Maringá: 2011. 84p. Dissertação de Mestrado (Sistema de Conhecimento) – Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de informática, Programa de Pós-Graduação em Ciência da Computação, 2011.

NICÁCIO, J. M.; **Engenharia de Software.** 2010. Disponível em: <http://jalvesnicacio.files.wordpress.com/2010/03/engenharia-de-software.pdf>. Acesso em: <27/05/2014>. [On-line].

NOY, N. F.; MCGUINNESS, D. L. **Ontology Development 101:** A Guide to Creating Your First Ontology. Stanford: 2001. 25p.

PRESSMAN, R. S. **Engenharia de software:** uma abordagem profissional**.** Porto Alegre: AMGH, 2011.

SAKAMOTO, K.; WASHIZAKI, H,; FUKAZAWA, Y. **Open code coverage framework:** A consistente and flexible framework for measuring test coverage supporting multiple programming languages. Tokyo: IEEE, 2010.

SCHIESSL, M.; BRÄSCHER, M. **Do texto às ontologias:** uma perspectiva para a ciência da informação. Revisões de Literatura. Ci. Inf, Brasília, DF, v. 40 n. 2, p. 301 – 311, 2011.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. **Knowledge Engineering:** Principles and methods**.** Data & Knowledge Engineering, v. 25, p 161-197, 1998.

TELES, V. M. **Análise de cobertura de testes automatizados usando Emma**. 2006. Disponível em: <desenvolvimentoagil.com.br/xp/praticas/tdd/emma>. Acesso em: <29/10/2013>. [On-line].

TIKIR, M. M.; HOLLINGSWORTH, J. K. **Efficient instrumentation for code coverage testing**. College Park: ACM, 2002.

W3C. **OWL Web Ontology Language guide.** 2003 **.** Disponível em: <http://www.w3.org/TR/2003/CR-owl-guide-20030818> Acesso em:<02/11/2013>. [On-line].

W3C. **Resource Description Framework (RDF):** Concepts and Abstract Syntax**.** 2004**.** Disponível em: <http://www.w3.org/TR/rdf-concepts> Acesso em: <14/11/2013>. [On-line].