3 Ontologias e Vocabulários

Neste capítulo vamos discutir as ontologias e vocabulários que foram utilizados na representação dos dados minerados. Primeiramente, vamos enumerar as ontologias atualmente existentes e que foram utilizadas neste trabalho. Logo após, vamos descrever as duas ontologias que foram criadas com o intuito de permitir a representação de certas informações não consideradas em outras abordagens.

3.1. Description of a Project (DOAP)

DOAP[47] é um RDF Schema e um vocabulário XML utilizado para descrever e compartilhar informações sobre projetos de software, em particular, projetos de código aberto. Este projeto tem sido utilizado na criação e manutenção de catálogos de projetos de software de grandes organizações, como a fundação Apache, a fundação Mozilla e o indexador de pacotes Python. A adoção deste esquema facilita a manutenção das informações sobre os projetos, pois os mantenedores destas informações só precisarem disponibilizar o endereço de seu arquivo DOAP atualizado para que os agregadores destes catálogos atualizem as informações dos projetos. Um benefício desta prática é que o projeto pode ser catalogado em diversos web sites sem a necessidade de cadastros manuais ou utilização de formatos específicos, facilitando o acesso de colaboradores em potencial.

Abaixo, demonstramos um exemplo de como as informações de um projeto são representadas utilizando DOAP (**Figura 5**). A seguir, apresentamos o **Tabela 2** com as principais classes e propriedades definidas nesta ontologia.

[47] https://github.com/edumbill/doap/wiki

```
<rdf:RDF       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:doap="http://usefulinc.com/doap/">
  <doap:Project>
   <doap:name>Example project</doap:name>
   <doap:homepage rdf:resource="http://example.com" />
   <doap:programming-language>javascript</doap:programming-
language>
   <doap:license
rdf:resource="http://example.com/doap/licenses/gpl"/>
  </doap:Project>
 </rdf:RDF>
```

Figura 5 Exemplo da representação das informações de um projeto utilizando DOAP

Classe	**Descrição**
Project	Classe que representa um projeto de software.
Repository	Classe que representa um repositório de código-fonte.
Version	Classe que representa as informações de um release de um projeto.
Propriedade	**Descrição**
mailing-list	Propriedade que representa o endereço da lista de discussão do projeto.
Category	Propriedade que representa a categoria de um projeto.
Release	Propriedade que liga um projeto a um release.
License	Propriedade que representa a url da licença sobre a qual o projeto é distribuído.
Developer	Propriedade que representa um desenvolvedor do projeto de software.
programing-language	Propriedade que representa a linguagem de programação utilizada na implementação do projeto.

Tabela 2 Elementos do vocabulário DOAP

3.2. Friend of a Friend (FOAF)[48]

FOAF é uma ontologia criada para descrever pessoas, suas atividades e seus relacionamentos utilizando a estrutura da web. Isto permite que grupos de

indivíduos formem redes sociais descentralizadas, onde cada elemento fornece uma parte da informação. Isto é possível pelo fato de que cada profile definido via FOAF ter um identificador único que é utilizado para ligar uma pessoa com outros indivíduos ou objetos, este identificador pode ser um e-mail pessoal ou a URL de um blog ou página. Além disso, estas descrições normalmente são publicadas como documentos na Web, resultando em uma rede de documentos que descreve uma rede de pessoas. Pelo fato de utilizarem um vocabulário comum, estes arquivos podem ser combinados de diversas formas e possibilitam, por exemplo, que agentes computacionais encontrarem todas as pessoas que moram na Europa e que são amigos de seus amigos. Abaixo, um exemplo de um profile descrito utilizando FOAF e uma tabela com suas principais classes e atributos:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#JW>
    a foaf:Person ;
    foaf:name "Jimmy Wales" ;
    foaf:mbox <mailto:jwales@bomis.com> ;
    foaf:homepage <http://www.jimmywales.com/> ;
    foaf:nick "Jimbo" ;
    foaf:depiction
<http://www.jimmywales.com/aus_img_small.jpg> ;
    foaf:interest <http://www.wikimedia.org> ;
    foaf:knows [
        a foaf:Person ;
        foaf:name "Angela Beesley"
    ] .

<http://www.wikimedia.org>
    rdfs:label "Wikimedia" .
```

Figura 6 Exemplo da representação de informações utilizando a ontologia FOAF

Classe	Descrição
Agent	Classe que representa um agente, isto é, uma entidade que executa ações, como grupos, pessoas ou organizações.
Person	Subclasse de Agent que é utilizada para representar pessoas. Elas

[48] http://www.foaf-project.org/

	podem ser reais ou personagens fictícios.
Group	Classe que representa uma coleção de indivíduos do tipo Agent. Pode representar um grupo de trabalho, um departamento em uma empresa e etc.
Document	Classe que representa um documento físico ou eletrônico.
Propriedade	**Descrição**
Name	Propriedade que representa um nome.
Nick	Propriedade que representa o apelido de uma pessoa.
Knows	Propriedade que representa que uma pessoa conhece outra pessoa.
Member	Propriedade que representa que um agente é membro de um grupo.
Publications	Propriedade que representa as publicações de uma pessoa.

Tabela 3 Elementos do vocabulário FOAF

3.3.Dublin Core [49]

Dublin Core é um conjunto de vocabulários de termos para descrever objetos digitais, tais como, vídeos, sons, imagens, textos e sites na web. Além de descrever recursos deste tipo, este vocabulário pode ser utilizado para combinar diferentes vocabulários de metadados e prover interoperabilidade entre aplicações da Web Semântica. Abaixo, apresentamos seus principais termos:

Propriedade	**Descrição**
Contributor	Propriedade que representa a entidade responsável por fazer contribuições para o recurso alvo, como um documento ou artigo. Exemplos de contribuidores incluem pessoas, organizações ou grupos.
Creator	Propriedade que representa a entidade primária responsável por criar o recurso alvo.
Identifier	Propriedade que identifica de forma desambígua um recurso em um determinado contexto. Um URI ou URL são exemplos de identificadores.

[49] http://dublincore.org/

Publisher	Propriedade que representa a entidade responsável por tornar um recurso disponível ou publicado.
Type	Propriedade que representa o tipo ou o gênero de um recurso.
Format	Propriedade que pode representar o formato de arquivo, a mídia física ou as dimensões de um recurso.

Tabela 4 Principais termos do vocabulário Dublin Core

3.4. Creative Commons Rights Expression Language (CC Rel) [50]

O CC REL é uma especificação da Creative Commons para descrever informações sobre licenças utilizando o metamodelo RDF. Abaixo, um exemplo de representação do licenciamento do conteúdo de um site.

```
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xhtml="http://www.w3.org/1999/xhtml/vocab#">
  <rdf:Description rdf:about="http://www.lessig.org/blog/">
<xhtml:license
rdf:resource="http://creativecommons.org/licenses/by/3.0/" />
  </rdf:Description>
</rdf:RDF>
```

Figura 7 Representação das informações de uma licença utilizando o vocabulário CC REL

As principais classes e propriedades utilizadas neste vocabulário são descritas na tabela abaixo:

Classe	Descrição
Work	Classe que representa algo que é potencialmente licenciável.
License	Classe que representa um conjunto de permissões e pedidos aos utilizadores de um trabalho.
Prohibition	Classe que representa uma proibição em uma determinada licença.

[50] http://wiki.creativecommons.org/CC_REL

Permission	Classe que representa uma ação que pode ser permitida em relação a uma determinada licença.
Requeriment	Classe que representa uma ação que pode ou não ser requerida aos usuários de trabalhos que utilizam uma determinada licença.
Propriedade	**Descrição**
License	Propriedade que representa o relacionamento entre um trabalho e sua licença.
Requires	Propriedade que liga uma licença a um Requisito.
Prohibits	Propriedade que liga uma licença a alguma proibição.
Permits	Propriedade que liga uma licença a uma permissão.

Tabela 5 Principais elementos do vocabulário CC REL

3.5.Open Services for Lifecycle Collaboration (OSLC) [51]

OSLC é uma comunidade aberta que tem como principal objetivo definir um conjunto de especificações que permitam a interoperabilidade de ferramentas que compõem o gerenciamento do ciclo de vida de aplicações. Inicialmente proposta pela IBM em 2008, mas atualmente composta por outras organizações e desenvolvedores independentes, a OSLC permite que diferentes fornecedores de ferramentas integrem seus dados e fluxos de trabalho para suportar desde a concepção de um software até a etapa de implementação. Estas especificações são definidas na forma de vocabulários RDF e utilizam outros vocabulários conhecidos, como o FOAF e o Dublin Core. Além disso, eles obedecem ao conjunto de regras e padrões da Linked Data[52]. Estes padrões recomendam a utilização de URIs no padrão HTTP para denotar recursos e também recomendam que informações relevantes sejam retornadas quando estas URIs são acessadas, como retornar documentos no formato RDF.

As especificações da OSLC são divididas em grupos de trabalhos que individualmente tratam dos diversos domínios de ferramentas, como: gerenciamento de mudança, gerenciamento da qualidade, gerenciamento de requisitos e gerência de configuração. Cada grupo é responsável por tratar os

[51] http://open-services.net/
[52] http://linkeddata.org/

possíveis cenários de integração e definir um vocabulário RDF comum que suporte estes cenários. Devido às características extensíveis e flexíveis do metamodelo RDF e dos princípios da Linked Data, cada ferramenta que adere a uma determinada especificação pode estender este vocabulário com seus próprios termos sem impactar a disponibilização de seus dados. Para garantir a coesão e integração entre estes domínios, cada grupo de trabalho deve seguir os conceitos e regras definidas na especificação OSLC Core. Este é o grupo de trabalho responsável por definir regras e padrões de comunicação entre as ferramentas, como por exemplo, o fato de que cada artefato é um recurso RDF que pode ser manipulado pelos métodos padrão da especificação HTTP (GET, PUT, POST, DELETE).

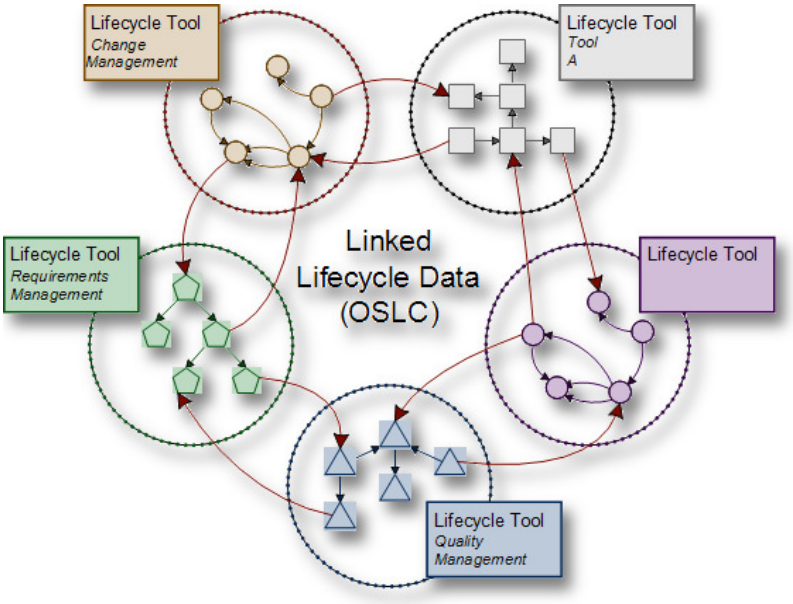

Figura 8 Ilustração da sinergia de ferramentas aderentes aos padrões da OSLC

Abaixo apresentaremos os vocabulários da OSLC que foram utilizados neste trabalho:

3.5.1. OSLC Change Management Vocabulary (OSLC CM) [53]

Este vocabulário define os termos comumente utilizados em ferramentas de Gerenciamento de Mudança, como por exemplo, o pedido de mudança. Abaixo

um exemplo da representação de um pedido de mudança e os principais termos presentes neste vocabulário:

```
<rdf:RDF
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:oslc_cm="http://open-services.net/ns/cm#">
<oslc_cm:ChangeRequest
rdf:about="http://example.com/bugs/4321">
<oslc_cm:relatedChangeRequest
 rdf:ID="link1"
rdf:resource="http://anotherexample.com/defects/123"/>
</oslc_cm:ChangeRequest>
    <rdf:Description rdf:about="#link1">
      <dcterms:title>Defect     123:     Problems     during
      install</dcterms:title>
   </rdf:Description>
</rdf:RDF>
```

Figura 9 Representação das informações de um defeito utilizando o vocabulário OSLC CM

Classe	**Descrição**
ChangeRequest	Classe que representa um recurso do tipo Requisição de Mudança.
Propriedade	**Descrição**
status	Propriedade que indica o status de uma requisição de mudança. Alguns valores possíveis são: Pronta, Em Progresso e Finalizada.
relatedChangeRequest	Propriedade que liga dois pedidos de mudança relacionados.
implementsRequirement	Propriedade que indica o Requisito que é implementado por um pedido de mudança.
testedByTestCase	Propriedade que indica o Caso de Teste que valida as

[53] http://open-services.net/bin/view/Main/CmSpecificationV2

	modificações feitas por um pedido de mudança.

Tabela 6 Elementos vocabulário OSLC Change Management

3.5.2. OSLC Automation Management Vocabulary (OSLC Auto) [54]

Este vocabulário define recursos ligados a ferramentas de automação de software. Estas ferramentas são frequentemente utilizadas para automatizar a execução de tarefas recorrentes durante o processo de desenvolvimento, como por exemplo, a execução de testes unitários, a construção do executável do projeto e a implantação em um determinado ambiente de execução. O principal objetivo desta especificação é permitir que informações sobre planos e requisições de automação possam ser representadas e que também possam relacionadas com recursos fora deste domínio (ex.: com o resultado de um caso de teste). Abaixo um exemplo de um plano de automação representado utilizando este vocabulário:

```
<rdf:RDF
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:oslc_auto="http://open-services.net/ns/auto#">
  <oslc_auto:AutomationResult rdf:about="http://example.com/results/4321">
     <oslc_auto:reportsOnAutomationPlan rdf:ID="link1"
        rdf:resource="http://example.com/plans/123" />
  </oslc_auto:AutomationResult>
  <rdf:Description rdf:about="#link1">
    <dcterms:title>Build Definition 123: Pet Shop App production
build</dcterms:title>
  </rdf:Description>
</rdf:RDF>
```

Figura 10 Representação do resultado de uma automação utilizando o vocabulário OSLC Automation

[54] http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.0/

Na tabela abaixo podemos identificar as principais classes e propriedades deste vocabulário:

Classe	Descrição
AutomationPlan	Classe que representa um plano de automação que pode ser executado pela ferramenta em questão.
AutomationRequest	Classe que define a submissão das informações necessárias para executar um plano de automação.
AutomationResult	Classe que representa o resultado da execução de um plano de automação, em conjunto com as contribuições para o resultado.
Propriedade	**Descrição**
State	Propriedade que indica o estado de um pedido ou de um resultado de automação.
executesAutomationPlan	Propriedade que indica qual plano de automação foi executado por um pedido de automação.
producedByAutomationRequest	Propriedade que indica o pedido de automação responsável por gerar um resultado de automação.

Tabela 7 Elementos do vocabulário OSLC Auto

3.5.3.OSLC Asset Management Vocabulary (OSLC Asset) [55]

Este vocabulário define termos e conceitos utilizados na prática de gerenciamento de ativos. Sistemas de gerenciamento de ativos permitem que empresas cataloguem, governem, gerenciem, pesquisem e mantenham ativos. Um ativo é qualquer coisa tangível ou intangível que agregue valor através de referência ou reuso durante um período de tempo considerável. O termo ativo

[55] http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/

neste vocabulário pode se referir tanto a um software, como sua documentação ou a algum equipamento de hardware que seja utilizado como apoio. Um ativo pode ser descrito por um conjunto de propriedades e um ou mais artefatos, os quais são representados por um ou mais arquivos físicos. Abaixo, um exemplo de utilização deste vocabulário. Em seguida, apresentamos uma tabela com seus principais termos e conceitos:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"xmlns:dcterms=http://purl.org/dc/terms/
 xmlns:oslc_asset="http://open-services.net/ns/artifact#">
   <oslc_asset:Artifact>
      <rdf:type                    rdf:resource="http://open-
services.net/ns/asset#Artifact"/>
      <dcterms:title>Build                                help
documentation</dcterms:title>
      <dcterms:label>documentation</dcterms:label>
      <oslc_asset:content
rdf:resource="http://help.example.com"/>
      <dcterms:format>text/html</dcterms:format>
   </oslc_asset:Artifact>
</rdf:RDF>
```

Figura 11 Exemplo de Representação das informações de um ativo utilizando o vocabulário OSLC Asset

Classe	**Descrição**
Asset	Classe que representa um ativo neste vocabulário
Artifact	Classe que representa um artefato.
Propriedade	**Descrição**
Gui	Propriedade que indica o identificador de um ativo.
version	Propriedade que indica a versão de um ativo.
State	Propriedade que indica o estado de um ativo.
artifact	Propriedade que ligar um Ativo a um Artefato que o compõe.
content	Propriedade que contem a URI do arquivo representado por um artefato

Tabela 8 Elementos do Vocabulário OSLC Asset Management

3.5.4.OSLC Quality Management Vocabulary (OSLC QM) [56]

Este vocabulário define termos e conceitos utilizados na disciplina de Gerenciamento de Qualidade. Isto inclui planos de teste, casos de teste e resultados de teste. Além disso, permite representar relacionamentos entre estes recursos e outros conceitos, como pedidos de mudança e requisitos. Abaixo um exemplo da representação de um caso de teste e de seu relacionamento com o requisito que o valida.

```
<rdf:RDF
xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:oslc_qm="http://open-services.net/ns/qm#">
<oslc_qm:TestCase
rdf:about="http://example.com/testcases/4321">
<oslc_qm:validatesRequirement
rdf:ID="link1"
rdf:resource="http://anotherexample.com/requirements/12
3"/>
</oslc_qm:TestCase>
    <rdf:Description rdf:about="#link1">
<dcterms:title>Requirement 123: Provide keyboard
accessibility</dcterms:title>
   </rdf:Description>
</rdf:RDF>
```

Figura 12 Exemplo da representação de um Caso de Teste utilizando o vocabulário OSLC Quality Management

Classe	Descrição
TestPlan	Classe que representa planos de teste.
TestCase	Classe que representa um caso de teste.
TestExecutionRecord	Classe que representa a execução de um teste.

[56] http://open-services.net/bin/view/Main/QmSpecificationV2

Propriedade	Descrição
relatedChangeRequest	Propriedade que relaciona um artefato de teste com um pedido de mudança
testsChangeRequest	Propriedade que indica qual pedido de mudança está sendo testado por um caso de teste.
validatesRequirement	Propriedade que indica qual requisito está sendo validado por um caso de teste.
usesTestCase	Propriedade que ligar um Plano de Teste a um Caso de Teste.
usesTestScript	Propriedade que liga um caso de teste ao script de teste que ele utiliza.

Tabela 9 Elementos do vocabulário OSLC Quality Management

3.5.5. OSLC Software Configuration Management Vocabulary (OSLC SCM) [57]

Define termos e conceitos utilizados no gerenciamento das versões dos itens que compõe um projeto de software. Estes itens podem englobar desde código-fonte até manuais de utilização do sistema. Abaixo os principais conceitos e propriedades definidos neste vocabulário:

Classe	Descrição
ObjectVersion	Classe que representa uma versão especifica de um objeto que está sendo controlado por um Gerenciador de Configuração de Software. Normalmente estes objetos representam arquivos ou diretórios.
Configuration	Classe que representa um conjunto de versões de objetos organizados em uma hierarquia. Uma configuração pode ter ligações com versões especificas de qualquer número de objetos.
Baseline	Classe que representa uma configuração imutável.
ChangeSet	Classe que representa um conjunto de mudanças feitas em uma

[57] http://open-services.net/bin/view/Main/ScmSpecV1

	configuração, sendo que cada mudança descreve o que deve ser alterado na configuração. Uma mudança pode ser a inclusão, remoção ou substituição de um objeto dentro de uma configuração.
Propriedade	**Descrição**
change	Propriedade que relaciona um Change Set com uma mudança em um objeto
changeType	Propriedade que indica o tipo de uma mudança
changedObject	Propriedade que indica o objeto modificado por uma mudança

Tabela 10 Elementos do vocabulário OSLC SCM

3.5.6. As Novas Ontologias

Nesta seção descrevemos as duas ontologias que foram criadas para suportar a nova abordagem de mineração de repositórios de software. Elas foram desenvolvidas para permitir a representação do relacionamento entre as versões de arquivos de código-fonte e as modificações na estrutura da *Abstract Syntax Tree* em cada versão. Conforme mencionamos anteriormente, as abordagens de mineração que extraem informações de repositórios de controle de versão não representam de forma explícita esta informação. Sendo que ela pode ser insumo para análises mais detalhadas de quem é o dono de um determinando código-fonte e para medir o conhecimento dos desenvolvedores sobre um determinado projeto.

A primeira ontologia criada descreve as informações sobre as propriedades e relacionamentos de um elemento do código-fonte. Já a segunda fornece conceitos e propriedades que podem ser utilizados na representação do relacionamento entre as versões de um arquivo e as mudanças nos elementos de código-fonte de uma AST. Nas próximas seções vamos descrever em detalhes as duas ontologias.

3.5.6.1. A Ontologia de Entidades, Propriedades e Relacionamentos do Código-Fonte (EPR)

A ontologia de entidades, propriedades e relacionamentos foi elaborada para descrever as características normalmente encontradas em uma AST de um arquivo de código-fonte. Alguns exemplos são Classes, Métodos, Atributos e seus

relacionamentos como, quais interfaces que uma classe implementa ou a relação de parâmetros de um método. Além disso, características como o nome, s assinatura de um método e seus modificadores também são representáveis. Entretanto, diferente de outras abordagens que utilizam predicados RDF para representar relacionamentos e propriedades (HYLAND-WOOD, CARRINGTON, KAPLAN, 2006) (TAPPOLET, 2010), em nossa abordagem estas informações são representadas como classes OWL. Como exemplo, pode-se observar a *data property name* na ontologia Evoont SOM (TAPPOLET, 2010), esta *data property* funcional é utilizada para identificar o nome de uma entidade, como o nome de uma classe ou de um método. Na **Figura 13** utilizamos esta propriedade para representar o nome da classe identificada pelo URI http://ontology/com/example/Class1 e apresentamos a mesma informação representa utilizando a Ontologia EPR.

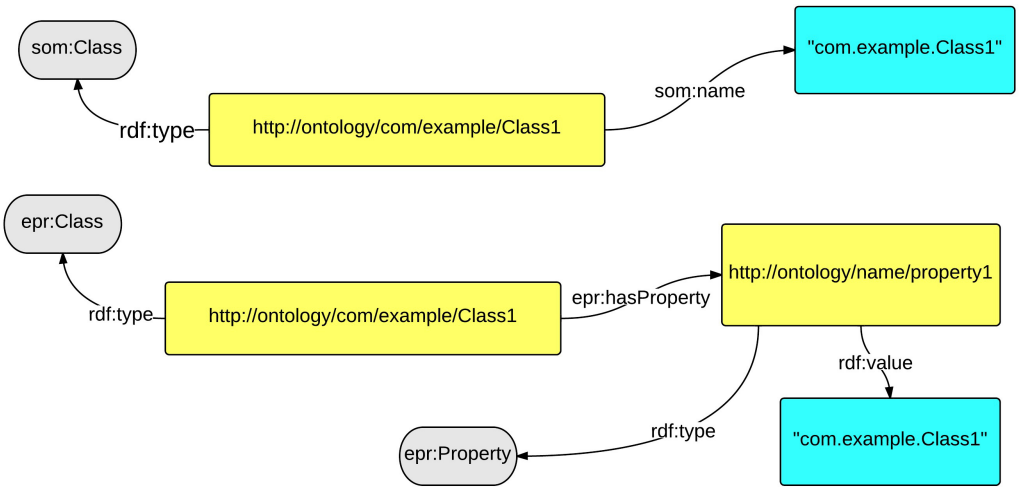

Figura 13 Exemplo de representação de uma classe chamada “org.example.Class1” na ontologia Evoont SOM e na ontologia EPR

Além deste tipo de propriedade, a ontologia também representa, como classes OWL, as *object properties* que relacionam duas entidades, como por exemplo, um método e a classe do tipo de seu retorno. Na **Figura 14** podemos observar esta representação em detalhes.

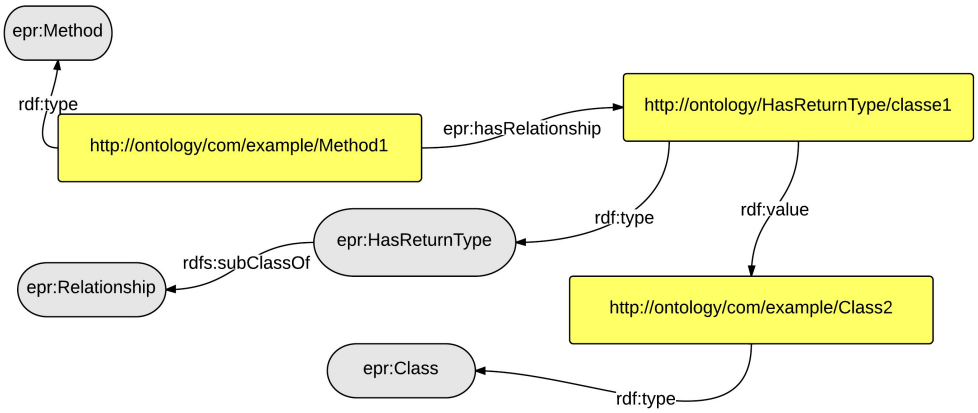

Figura 14 Exemplo de representação do relacionamento entre um método e o tipo de seu retorno

Na tabela abaixo apresentamos os principais conceitos e propriedades que são definidas nesta ontologia. Já na Figura 15, demonstramos visualmente como estes principais conceitos se relacionam.

Classe	**Descrição**
Entity	Classe que representa uma entidade de código-fonte. Suas principais especializações são *Class*, *Method*, *Parameter* e *Attribute*.
Property	Classe que representa os tipos de propriedade que uma entidade pode ter. Suas principais especializações são *Name*, *Signature*, *SimpleName* e *Modifier*.
Relationship	Classe que representa um relacionamento entre duas entidades. Algumas especializações são *HasMethod*, *HasReturnType* e *Extends*
Propriedade	**Descrição**
hasProperty	Propriedade que relaciona uma Entidade com uma instância da classe *Property*.
hasRelationship	Propriedade que relaciona uma Entidade com uma instância da classe *Relationship*.
isPropertyOf	Propriedade inversa de *hasProperty*.
isRelationshipOf	Propriedade inversa de *hasRelationship*

inverseRelationship	Propriedade simétrica que liga duas instâncias de Relacionamento para indicar que são relacionamentos inversos. Exemplo: uma instância do relacionamento *Extends* tem como inversa uma instância do relacionamento *IsExtendedBy*.

Tabela 11 Principais elementos da ontologia de Entidades, Propriedades e Relacionamentos

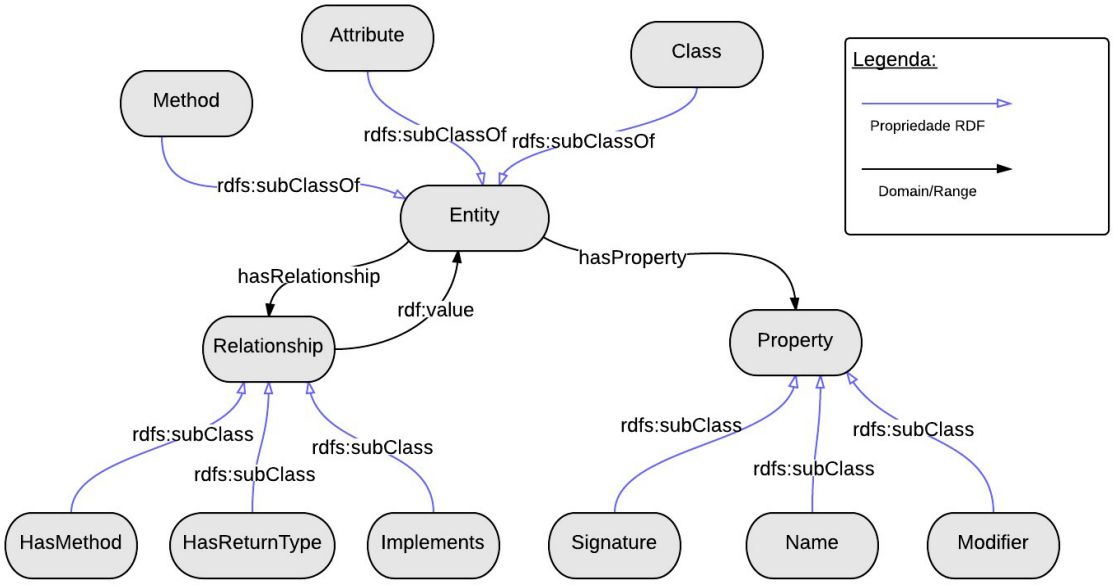

Figura 15 Representação visual dos elementos da ontologia de Entidades, Propriedades e Relacionamentos

3.5.6.1.1. Os metadados das propriedades e relacionamentos

Outros aspectos que são representados nesta ontologia são metadados referentes às subclasses de *Relationship* e *Property*. O primeiro aspecto indica a multiplicidade de um tipo de relacionamento ou de um tipo de propriedade, isto é, quantas vezes instâncias ativas destes tipos podem estar associadas a uma entidade. Um exemplo é a regra que determina que um método tenha apenas uma única assinatura de chamada. O mesmo se aplica para o relacionamento de herança entre duas classes na linguagem Java, pois em um dado instante do tempo, uma classe somente pode ter herança de uma única classe.

O segundo metadado representado indica se um tipo de relacionamento é um container de entidades. Isto significa que as entidades associadas com este

relacionamento dependem do mesmo para existir, da mesma forma que o tipo de associação Composição na UML[58]. Neste caso, se um relacionamento com esta característica sofrer uma remoção, os elementos associados também devem ser removidos. Um exemplo típico é a exclusão de um método em uma classe. Como ele pode ter múltiplos relacionamentos container do tipo *HasParameter* para representar o seu relacionamento com seus parâmetros, sua remoção faz com que a entidade do tipo *Parameter* também seja removida por estar associada com o relacionamento. Abaixo uma representação desta situação:

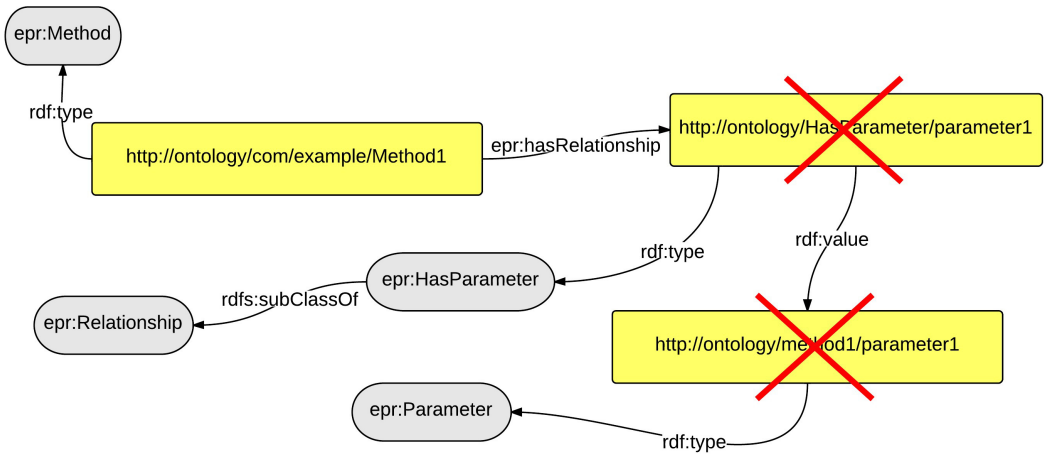

Figura 16 Representação da exclusão de um parâmetro de um método na ontologia EPR

3.5.6.1.2. Discussão sobre a Modelagem da Ontologia de Entidades, Propriedades e Relacionamentos

Apesar do alto nível de indireção causado pela forma granular de representação da Ontologia EPR, este mecanismo provê diversas vantagens em relação ao formato tradicional de representação de outras ontologias. Uma destas vantagens é a possibilidade de suportar diferentes linguagens e paradigmas de programação sem a necessidade de modificar sua estrutura básica – Entidade, Propriedade e Relacionamento. É apenas necessário criar novas classes OWL para representar novos tipos de entidades, relacionamentos e propriedades. A segunda vantagem desta abordagem diz respeito à possibilidade de representação de informações adicionais sobre as mudanças ocorridas em cada versão de um

[58] http://en.wikipedia.org/wiki/Object_composition

arquivo de código-fonte. Um exemplo deste tipo de informação é a data que um determinado relacionamento entre duas classes surgiu e qual foi a revisão no controle de versão que introduziu esta informação.

Para permitir este tipo de modelagem, utilizamos como base um mecanismo existente na especificação RDF para expressar informação sobre triplas ou fatos. Este mecanismo, conhecido como reificação[59], consiste na designação de um URI para cada tripla que se deseja agregar informações adicionais. A partir desse identificador é possível realizar novas asserções, como por exemplo, qual é a fonte de origem da informação ou quem fez a afirmação contida na tripla. A reificação surgiu como forma de superar uma das limitações das linguagens da Web Semântica, RDF e OWL, que é o fato de ambas lidarem somente com relacionamentos binários do tipo sujeito-predicado-objeto. O vocabulário de reificação do RDF consiste no tipo *rdf:Statement* e nas propriedades *rdf:subject*, *rdf:predicate*, e *rdf:object* . O primeiro representa o URI que foi definido para a tripla que se deseja reificar. Já os três últimos representam os três componentes de uma tripla, ou seja, o sujeito, o predicado e o objeto. Abaixo, demonstraremos um exemplo de reificação de uma tripla que afirma que uma pessoa **p1** faz parte de um comitê **c1**.

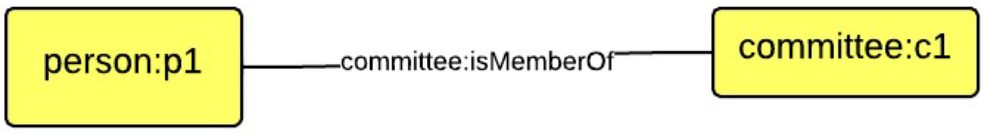

Figura 17 Representação da relação de membro entre a pessoa “p1” e o comitê “c1”

Através da utilização do vocabulário de reificação é possível afirmar novos fatos sobre este relacionamento, como por exemplo, quem é a pessoa responsável por nomear a pessoa **p1** no comitê **c1**. Para isso, foi criado o recurso **committee:membership12345** para que novas informações possam ser agregadas a tripla acima.

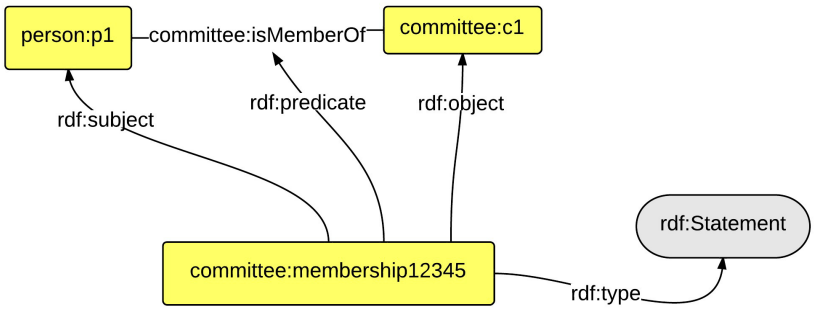

Figura 18 Representação da nova entidade “membership” que foi criada para representar informações sobre o relacionamento da pessoa “p1” e o comitê “c1”

A figura acima define que a URI **committee:membership12345** é um recurso RDF do tipo *rdf:Statement*. Além disso, define seu relacionamento com os elementos da tripla inicial, como o relacionamento do tipo *rdf:subject* entre o URI e o recurso **person:p1**, o relacionamento *rdf:predicate* com o predicado que une **p1** à **c1** e, por último, define um relacionamento *rdf:object* com o objeto **c1** da tripla inicial. Logo, através desse URI, é possível identificar a pessoa **p2** responsável pela criação da tripla inicial, ou seja, pela nomeação de **p1** no comitê **c1**.

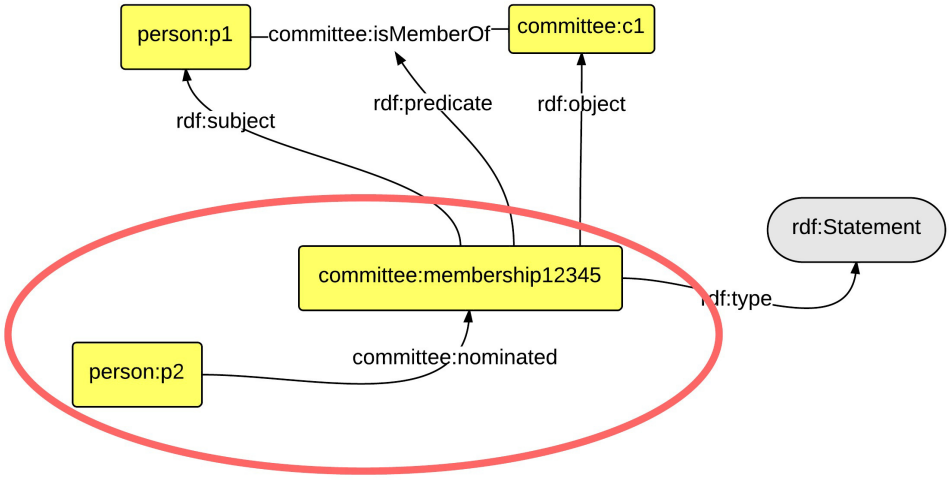

Figura 19 Representação do responsável pela nomeação da pessoa “p1” no comitê “c1”

[59] http://en.wikipedia.org/wiki/Reification_(computer_science)#RDF_and_OWL

Na ontologia em questão, os tipos Propriedade e Relacionamento poderiam ser vistos como subclasses de *rdf:Statement*, por representarem as informações que têm necessidade de reificação, por exemplo, as características de uma classe (ex.: seu nome) ou o relacionamento desta classe com outra classe (ex.: uma classes implementado uma interface). Já as propriedades inversas *isPropertyOf* e *isRelationshipOf* podem ser interpretadas como subpropriedades de *rdf:subject* pelo fato de ligarem uma propriedade ou relacionamento com o sujeito a qual pertencem, neste caso, uma entidade da AST de um código. Já o predicado *rdf:object* seria representado pelo predicado *rdf:value*, por ser o predicado escolhido para ligar uma propriedade ou relacionamento com o seu valor. Entretanto, não existe um correspondente para *rdf:predicate*, pelo fato do próprio recurso que representa um *rdf:Statement*, no caso uma propriedade ou um relacionamento, ter um tipo definido, por exemplo, *Name*, *HasMethod* e etc.

3.5.6.2.A Ontologia de Impactos

A Ontologia de Impactos descreve os termos necessários para a representação das diferenças entre duas versões de uma AST de um código-fonte. Como discutimos anteriormente, as abordagens atuais de extração não consideram explicitamente o relacionamento inerente entre as revisões de um arquivo de código-fonte nas ferramentas de controle de versão e o conjunto de modificações estruturais que cada uma destas revisões causou. Normalmente, as abordagens tratam os arquivos como simples documentos textuais e tratam as diferenças entre suas versões apenas como linhas incluídas, excluídas e modificadas. Similarmente à Ontologia EPR, esta ontologia foi concebida para representar as informações sobre as diferenças de forma granular, em termos de adições e remoções de Entidades, Propriedades e Relacionamentos. Desta maneira é possível derivar novas informações a partir de impactos básicos de adição e remoção. Por exemplo, na **Figura 21** representamos um impacto de Modificação sofrido por um método devido ao fato de sua visibilidade "**package**" ter sido removida e a visibilidade "**public**" ter sido adicionada. Acreditamos que armazenar as diferenças estruturais do código fornece informações e novas possibilidades de consultas que em outros trabalhos não seria possível. Na seção de Estudo de Caso

exemplificaremos alguns cenários onde a Ontologia de Impactos é imprescindível para isso.

O principal elemento desta ontologia, o Impacto, representa um fato sobre uma propriedade ou um relacionamento, ou mesmo, um fato sobre a própria entidade dona de ambos, quanto esta é criada ou excluída, por exemplo. Outros dois importantes conceitos representados são as classes Contexto e Agente. O primeiro tem um duplo relacionamento com um impacto, ora como a origem do estado anterior da AST, ora como origem do estado atual dela. Isto permite identificar exatamente quais as versões da AST foram utilizadas no cálculo de suas diferenças ou impactos. Nesta versão da ontologia, apenas o tipo ChangeSet definido no vocabulário OSLC SCM pode ser utilizado como um contexto, ou seja, o impacto será dado pela diferença de dois conjuntos de mudanças feitos pelos desenvolvedores. Já o segundo termo, o Agente, especifica o criador de um contexto. Nesta versão, um agente é representado pela entidade Person na ontologia FOAF, ou seja, o desenvolvedor responsável por um conjunto de mudanças.

Outra informação possível de ser representada é a hierarquia de impactos (Figura 20). Do mesmo modo que existe uma hierarquia entre os elementos em uma AST, onde, por exemplo, um método está em um nível hierárquico maior do que as suas variáveis internas, também existe uma hierarquia análoga entre os diversos impactos que afetam estes elementos. A captura desta informação permite, por exemplo, responder de forma mais simples quais são os elementos que podem ter sofrido alteração de comportamento devido a uma modificação em um elemento de mais baixo nível de hierarquia. Um exemplo é demonstrar como a mudança em um método pode afetar uma parte sensível do sistema, pois é possível navegar pela hierarquia de impactos e montar o grafo de elementos que tem relacionamentos com o elemento modificado até chegar aos elementos que compõe a parte crítica do software.

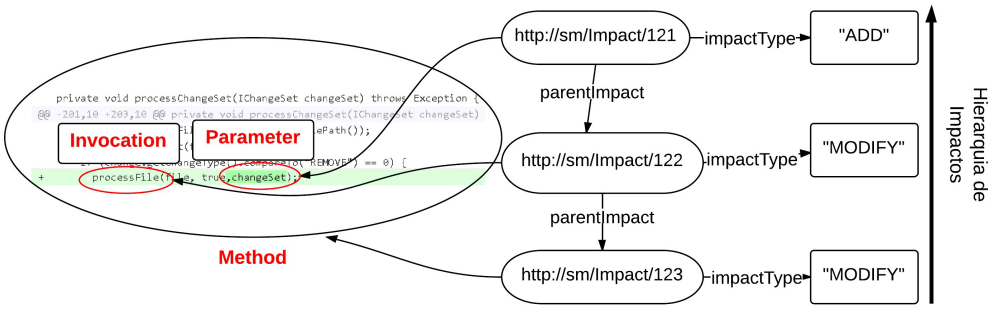

Figura 20 Exemplo da Hierarquia de impactos

Abaixo elencamos as principais classes e propriedades definidas nesta ontologia. Em seguida, mostramos um exemplo de como seria a representação de uma modificação em um método hipotético chamado "**facaAlgo()**". A modificação consiste na alteração de sua visibilidade de "**package**" para "**public**".

Classe	**Descrição**
Impact	Classe que representa qualquer alteração sofrida por uma entidade, propriedade ou relacionamento.
Context	Classe que representa uma referência para um estado de uma AST. Isto permite identificar exatamente quais as versões da AST foram utilizadas no cálculo do impacto.
Agent	Classe que representa a entidade responsável pela criação de um contexto.
Propriedade	**Descrição**
impactType	Propriedade que relaciona um impacto com seu tipo. Alguns de seus valores válidos são: Inclusão, Remoção, e Modificação.
impactDate	Propriedade que representa a data que um impacto ocorreu.
contextFrom	Propriedade que associa um Impacto com o contexto origem utilizado para calculá-lo.
Context	Propriedade que associa um Impacto com o contexto destino utilizado para calculá-lo.
childImpact	Propriedade que associa um impacto de mais alto nível com seus impactos filhos.

Tabela 12 Principais elementos da ontologia de Impactos

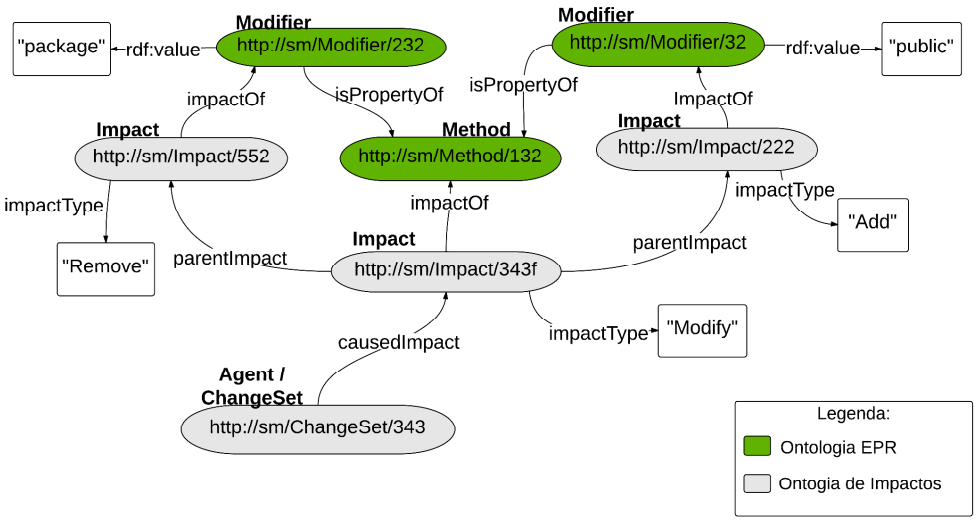

Figura 21 Representação da modificação da visibilidade de um método de “package” para “public”