UNIVERSIDADE CANDIDO MENDES - UCAM
PROGRAMA DE PÓS-GRADUAÇÃO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

Matheus Dimas de Morais

ISSUE PROCEDURE ONTOLOGY (IPO): UMA ONTOLOGIA EXTENSÍVEL PARA O DOMÍNIO DE SINTOMAS, PROBLEMAS E SOLUÇÕES

CAMPOS DOS GOYTACAZES, RJ
SETEMBRO DE 2015

UNIVERSIDADE CANDIDO MENDES - UCAM
PROGRAMA DE PÓS-GRADUAÇÃO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

Matheus Dimas de Morais

ISSUE PROCEDURE ONTOLOGY (IPO): UMA ONTOLOGIA EXTENSÍVEL PARA O DOMÍNIO DE SINTOMAS, PROBLEMAS E SOLUÇÕES

Dissertação apresentada ao Programa de Pós-Graduação em Pesquisa Operacional e Inteligência Computacional da Universidade Candido Mendes – Campos dos Goytacazes / RJ, para obtenção do grau de MESTRE EM PESQUISA OPERACIONAL E INTELIÊNCIA COMPUTACIONAL.

Orientador: Prof. Mark Douglas de Azevedo Jacyntho, D.Sc.

CAMPOS DOS GOYTACAZES, RJ
SETEMBRO DE 2015

MATHEUS DIMAS DE MORAIS

ISSUE PROCEDURE ONTOLOGY (IPO): UMA ONTOLOGIA EXTENSÍVEL PARA O DOMÍNIO DE SINTOMAS, PROBLEMAS E SOLUÇÕES.

Dissertação apresentada ao Programa de Pós-Graduação em Pesquisa Operacional e Inteligência Computacional da Universidade Candido Mendes – Campos dos Goytacazes / RJ, para obtenção do grau de MESTRE EM PESQUISA OPERACIONAL E INTELIÊNCIA COMPUTACIONAL.

Aprovada em 11 de setembro de 2015

BANCA EXAMINADORA

Prof. Mark Douglas de Azevedo Jacyntho, D.Sc.

Universidade Candido Mendes

Prof. Ítalo de Oliveira Matias, D.Sc.

Universidade Candido Mendes

Prof.ª Aline Pires Vieira de Vasconcelos, D.Sc.

Instituto Federal Fluminense

CAMPOS DOS GOYTACAZES, RJ
2015

Dedico este trabalho aos meus pais que, com todo esforço e dedicação, tornaram possível minha caminhada até aqui.

AGRADECIMENTOS

Primeiramente a Deus, por permitir que tudo que conhecemos exista.

A minha esposa Carina, pelo apoio em todos os momentos.

Aos meus familiares e amigos pela compreensão nos momentos em que estive ausente por causa do tempo dedicado a este trabalho.

Ao meu orientador, pelas contribuições valiosas que permitiram a realização dessa dissertação.

Ao Instituto Federal Fluminense, que viabilizou financeiramente a realização do meu mestrado.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“E se as cabeças fossem quadradas?
E se bastassem três acordes?
E se ciência e religião fizessem as pazes?
E se fosse possível prever o futuro?
E se?
Questione.
Descubra.
Mude.
O conhecimento é irresistível. ”

Canal Futura

RESUMO

ISSUE PROCEDURE ONTOLOGY (IPO): UMA ONTOLOGIA EXTENSÍVEL PARA O DOMÍNIO DE SINTOMAS, PROBLEMAS E SOLUÇÕES.

No cenário atual de intensa competitividade entre as empresas, cada vez mais o uso do tempo se torna algo a ser maximizado, visando maior produtividade. Assim, para as organizações se manterem em plena operação, problemas devem ser solucionados o mais breve possível. A informática pode ajudar nesse processo, oferecendo sistemas computacionais que auxiliem na resolução de tais problemas. Com o advento da *Web* Semântica, surge a possibilidade de representar as informações de tal forma que o computador consiga compreendê-las, possibilitando a criação de sistemas inteligentes com o uso de ontologias. A principal contribuição deste trabalho é uma ontologia extensível (*core ontology*) para o domínio de sintomas, problemas e soluções, denominada *Issue Procedure Ontology* (IPO). Esta ontologia pretende prover às máquinas a semântica necessária para que estas forneçam não apenas informações, mas, sobretudo, identifiquem problemas a partir de um conjunto de sintomas e, em seguida, sugiram possíveis soluções, de forma autônoma, para os problemas em questão. Ao final, um estudo de caso realista demonstra uma forma de se especializar a ontologia para se obter maior poder de expressividade e autonomia por parte das máquinas.

PALAVRAS-CHAVE: Ontologia extensível; Sintomas, problemas e soluções; *Web Semântica*, OWL, Dados ligados, RDF.

ABSTRACT

***ISSUE PROCEDURE ONTOLOGY* (IPO): A CORE ONTOLOGY FOR THE DOMAIN OF SYMPTOMS, PROBLEMS AND SOLUTIONS.**

In the current scenario of intense competitiveness among companies, use of time increasingly becomes something to be maximized, aiming at greater productivity. Thus, for organizations remain in full operation, problems should be solved as soon as possible. Information technology can help this process by providing computer systems to assist in troubleshooting. With the advent of the Semantic Web, there is the possibility to represent information in such a way that the computer can understand them, enabling the creation of intelligent systems with the use of ontologies. The main contribution of this work is an extensible ontology (core ontology) for the domain of symptoms, problems and solutions, named Issue Procedure Ontology (IPO). This ontology aims to provide the necessary semantics for machines so that they provide not only information, but, above all, identify problems from a set of symptoms and then autonomously suggest possible solutions for the problems in question. At the end, a realistic case study demonstrates a way to specialize the ontology to obtain greater power of expressiveness and autonomy of the machines.

KEYWORDS: Core ontology; Symptoms, problems and solutions; Semantic Web, OWL, Linked data, RDF.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ASCII – American Standard Code for Information Interchange
CPF – Cadastro Pessoa Física
DC – Dublin Core
DOLCE – Descriptive Ontology for Linguistic and Cognitive Engineering
DnS – Descriptions and Situations
DUL – DOLCE+DnS Ultralite
FOAF – Friend of a Friend
GRDDL – Gleaning Resource Descriptions from Dialects of Languages
HD – Hard Disk
HDO – HelpDesk Support Ontology
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
IPO – Issue Procedure Ontology
IPO-M – Issue Procedure Ontology Medicine
IRI – Internationalized Resource Identifier
ISBN – International Standard Book Number
ITIL v3 – Information Technology Infrastructure Library version 3
JSON - JavaScript Object Notation
JSON-LD – JSON for Linking Data
KOSs – knowledge organization systems
LOD – Linked Open Data
LOV – Linked Open Vocabularies
MARC – Machine Readable Cataloging
N3 – Notation 3
OWL – Ontology Web Language
OWL DL – Ontology Web Language Description Logic
ORG – The Organization Ontology
P-PLAN – Provenance and Plans
PWO – Publishing Workflow Ontology
RDF – Resource Description Framework
RDFa – Resource Description Framework in Attributes

RDFS – Resource Description Framework Schema

REGORG – Registered Organization Vocabulary

RH – Recursos Humanos

RIF – Rule Interchange Format

SKOS – Simple Knowledge Organization System

SPARQL – SPARQL Protocol and RDF Query Language

SPARUL – SPARQL Update Language

SWRL – Semantic Web Rule Language

TI – Tecnologia da Informação

UML – Unified Modeling Language

URI – Uniform Resource Identifier

URL – Uniform Resource Location

W3C – Word Wide Web Consortium

XML – eXtensible Markup Language

CONVENÇÕES

- Nos trechos do texto que são transcritos os códigos, convencionou-se o uso da fonte *Courier New*, visando um destacamento do texto em si:

```
xmlns:ipo=http://purl.org/ipo/core#
```

- Com o mesmo objetivo, as classes e propriedades das ontologias, também foram descritas utilizando a fonte *Courier New*. Classes sempre iniciam com a letra maiúscula e propriedades com a letra minúscula:

```
Issue
solvedBy
```

- Para identificar a ontologia da qual procede cada classe e propriedade expostas neste trabalho, são acrescentados prefixos, convencionados pelas próprias ontologias para referenciá-las:

```
foaf:Person

ipo:indicatedBy

dc:title
```

SUMÁRIO

1 INTRODUÇÃO

No cenário atual de intensa competitividade entre as empresas, cada vez mais o uso do tempo se torna algo a ser maximizado, visando maior produtividade. Assim, as organizações devem alocar seus recursos e gerenciá-los de forma adequada, para que a força de trabalho tenha plena condição de realizar suas tarefas com maior eficiência (MAGALHÃES & PINHEIRO, 2007; REZENDE, 2002).

A área de Tecnologia da Informação (TI) tem sido de suma importância para as organizações, uma vez que os sistemas computacionais podem agilizar o trabalho feito pelos funcionários, servindo como meio para as organizações alcançarem seus objetivos. Hoje, para muitas empresas, a TI se tornou parceira estratégica, fazendo parte do negócio da empresa (PEREIRA; DE SOUZA; DA COSTA, 2012).

Assim, os serviços de TI podem ajudar diversos setores das empresas a solucionar seus problemas. Um problema geralmente possui sintomas que indicam sua existência e uma ou mais possíveis soluções, meios que se consiga sanar (ou amenizar) o problema, acabando (ou reduzindo), assim, os sintomas.

O processo de (i) analisar os sintomas, (ii) identificar o problema e (iii) localizar a solução, depende da ação humana do profissional, que pode errar em seu diagnóstico, devido sua falta de experiência ou atenção. Assim, sistemas computacionais podem ser utilizados para facilitar e auxiliar o profissional nessa importante tarefa.

Com o advento da *Web* Semântica ou *Web* de Dados Ligados (*Web of Linked Data*), surge a possibilidade de representar as informações de maneira que o computador consiga compreendê-las. Até então, os dados publicados na *Web* tinham como foco o entendimento pelo homem, sem a preocupação de estruturá-

los de maneira que a máquina conseguisse compreendê-los. Assim, a *Web* Semântica não é uma *Web* separada, mas uma extensão da atual, onde a informação é publicada na *Web* com um significado explícito e estruturado, permitindo melhor interação entre máquinas e pessoas (BERNERS-LEE; HENDLER; LASSILA, 2001).

Para estruturar os dados na *Web*, o consórcio W3C[1] criou um modelo de dados padrão denominado *Resource Description Framework* (RDF[2]) que permite a publicação de dados na *Web* estruturados no formato de triplas: sujeito – predicado – objeto (MANOLA et al., 2004). De acordo com Heath e Bizer (2011), qualquer dado pode ser publicado em formato de triplas, permitindo que, independente de sistemas e domínios, esses dados sejam compartilhados e reusados por toda a *Web*, uma vez que todos os dados estão descritos sob um mesmo modelo padrão.

Para que os dados sejam compreendidos pelos computadores, se faz necessário que os mesmos sejam estruturados de forma padronizada e que seja dado sentido semântico a eles. Para associar semântica aos dados, o W3C propõe a utilização de ontologias ou vocabulários. Ontologia é uma representação formal de um determinado domínio de conhecimento, onde conceitos (classes) deste domínio são explicitamente definidos, bem como suas propriedades e relacionamentos (JACYNTHO, 2012). São as ontologias que dão poder de inferência às máquinas possibilitando, através de raciocinadores (*softwares* que fazem uso dos axiomas ontológicos para inferir novos dados), a descoberta de novos conhecimentos (LICHTNOW & OLIVEIRA, 2009; PICKLER, 2007).

Assim, com o auxílio das tecnologias da *Web* Semântica, é possível desenvolver sistemas semânticos que auxiliem no processo de (i) analisar os sintomas, (ii) identificar o problema e (iii) localizar a solução. Diante disso, a criação de uma ontologia que represente o domínio de sintomas, problemas e soluções seria de grande valia para construção de sistemas computacionais que auxiliem profissionais na tomada de decisão.

1.1 PRESSUPOSTOS E QUESTÕES DE PESQUISA

[1] *World Wide Web Consortium* - http://www.w3.org/
[2] RDF - http://www.w3.org/RDF/

Em diversas profissões, as pessoas se veem em situações em que devem tomar decisões. Essas decisões são tomadas com base em um conjunto de dados que descrevem a situação atual e, a partir da decisão tomada, se pretende alcançar um objetivo ou solucionar um problema.

Um setor de TI de uma empresa, por exemplo, é responsável por manter os serviços de TI em pleno funcionamento e qualquer problema pode acarretar em grandes prejuízos. Assim, para estes profissionais, o tempo de resolução de problemas se torna algo a ser minimizado para que o tempo de operabilidade dos equipamentos tecnológicos seja maximizado e os funcionários não fiquem inoperantes.

Diante desse cenário de intensa responsabilidade, ter um sistema que auxilie na identificação rápida dos problemas, a partir dos relatos dos funcionários, permite que o problema seja rapidamente solucionado. Assim, um sistema que auxilie nessa tarefa se torna algo bastante desejável ou, em alguns casos, essencial.

Com a *Web* Semântica, a máquina ganha uma nova área de ação, pois, com o uso de ontologias, a mesma passa a entender os dados que estão disponíveis nas bases digitais e com as afirmações semânticas (axiomas) contidas nas ontologias, é possível que a máquina raciocine em cima desses dados, possibilitando criar sistemas inteligentes capazes de auxiliar os profissionais em diversas tarefas.

Sendo assim, questiona-se como se projetar uma ontologia que permita a máquina relacionar sintomas com problemas e estes com possíveis soluções de modo a identificar e classificar automaticamente tais problemas. Qual seria a melhor forma de representar as estruturas dos sintomas, problemas e soluções? Quais principais conceitos deveriam estar presentes nessa ontologia? Como criar uma ontologia suficientemente genérica para diversos tipos de problema, mas que, ao mesmo tempo, possa ser especializada para um domínio de problemas particular?

Este trabalho visa, sobretudo, resolver estas questões de pesquisa.

1.2 OBJETIVOS

1.2.1 Objetivo geral

- O objetivo geral deste trabalho é projetar uma ontologia extensível (*core ontology*) que represente o domínio de sintomas, problemas e soluções, provendo meios para que a máquina consiga relacionar os sintomas com seus problemas e estes com possíveis soluções.

1.2.2 Objetivos específicos

- Projetar uma ontologia genérica extensível (*core ontology*), de acordo com as tecnologias e padrões da *Web* Semântica propostos pelo W3C, que represente o domínio de sintomas, problemas e soluções, provendo meios para que a máquina consiga relacionar os sintomas com seus problemas e estes com suas possíveis soluções.
- Desenvolver um estudo de caso realista, especializando a ontologia aqui criada para demonstrar como esta pode ser customizada para melhor adaptação às especificidades dos problemas de um domínio de conhecimento particular.

1.3 JUSTIFICATIVA

Com o surgimento da *Web* Semântica, que propôs uma extensão da *Web* atual para a *Web* de dados, surge também a necessidade de representar esses dados para que máquina consiga interpretá-los para então nos ajudar na gestão e aproveitamento dessa valiosa base de dados que a *Web* se tornou.

Assim, as tecnologias da *Web Semântica* vêm prover meios para que a estruturação e representação desses dados sejam possíveis.

Sob o ponto de vista das ontologias, o objetivo é representar formalmente um determinado domínio de conhecimento, no qual conceitos (classes) deste domínio são explicitamente definidos, bem como suas propriedades e relacionamentos, de modo que a máquina os compreenda e racione sobre eles. Isso confere a máquina o poder de inferência, possibilitando, com o emprego de

softwares raciocinadores, a descoberta de novos conhecimentos (JACYNTHO, 2012; LICHTNOW; DE OLIVEIRA, 2009).

O papel das ontologias na *Web* de dados compreende, dentre outras atribuições: promover compartilhamento de dados; minimizar interpretações ambíguas, explicitar relacionamentos entre conceitos; integrar diferentes fontes de dados; prover axiomas lógicos que possibilitem inferência de novos dados.

Dessa forma, emerge uma demanda latente de ontologias para representar os diversos domínios do conhecimento. Este trabalho almeja ajudar nessa tarefa propondo uma ontologia para o domínio de sintomas, problemas e soluções contribuindo, portanto, para o desenvolvimento da *Web* de dados.

1.4 CONTRIBUIÇÕES

A principal contribuição deste trabalho é a ontologia proposta que pretende representar o domínio de sintomas, problemas e soluções.

Espera-se que a ontologia proveja os mecanismos necessários para que a máquina seja capaz de identificar os possíveis problemas a partir de sintomas, bem como sugerir possíveis soluções a partir dos problemas identificados. Diferentemente de um sistema de informação, onde a máquina nos fornece informações para que resolvamos o problema, espera-se que esta ontologia seja um passo fundamental em direção a construção de sistemas de conhecimento, onde a máquina não forneça apenas informações, mas também forneça possíveis soluções, de forma autônoma, para o problema em questão.

Pretende-se com este auxílio precioso das máquinas, reduzir o tempo desde a identificação do problema até sua solução, bem como reduzir o risco de tomar uma decisão equivocada por parte de profissionais menos experientes.

Outra contribuição a ser considerada é o estudo de caso desenvolvido neste trabalho, pois apresenta as diretrizes de uso da ontologia proposta, servindo com uma fonte de conhecimento auxiliar para futuras aplicações.

1.5 ORGANIZAÇÃO DO TEXTO

Nesta Introdução foram apresentados o contexto, pressupostos e questões que norteiam essa pesquisa, além dos objetivos gerais e específicos.

No capítulo dois, *Fundamentos Teóricos*, aprofunda-se nos principais conceitos: *Web* Semântica, Ontologias e Resolução de Problemas.

O capítulo três, *Design* da *Issue Procedure Ontology*, é descrita a ontologia desenvolvida nesse trabalho, expondo suas classes, propriedades e características semânticas. Além disso, é demonstrado um exemplo de uso da ontologia criada, especializando-a para o domínio da medicina.

No capítulo quatro, *Trabalhos Relacionados*, são apresentadas algumas ontologias existentes e, por fim, o quinto capítulo, *Conclusões*, discorre sobre a considerações finais e trabalhos futuros.

2 FUNDAMENTOS TEÓRICOS

O objetivo deste capítulo é apresentar os principais conceitos abordados neste trabalho, a saber: *Web* Semântica, *Ontologias* e *Resolução de Problemas*. O intuito é defini-los e fixar o entendimento para melhor compressão do trabalho realizado.

Na próxima seção será apresentado uma visão geral sobre a *Web* Semântica abordando suas tecnologias e padrões. Na seção seguinte, um estudo sobre ontologias, seus tipos e metodologias e, por fim, é apresentado o processo de resolução de problemas, em diversos contextos e como eles são percebidos.

2.1 *WEB* SEMÂNTICA

A *Web* Semântica pretende evoluir a *Web* atual, conhecida como “*Web* de documentos”, que tem como principal objetivo disponibilizar informações para as pessoas, para a “*Web* de dados” que visa disponibilizar os dados para que as máquinas possam manipulá-los de modo mais eficiente, transformando a *Web* em um grande banco de dados. Assim, a *Web* Semântica não é uma *Web* separada, mas uma extensão da atual, onde a informação é publicada na *Web* com um significado explícito e estruturado, permitindo melhor interação entre máquinas e pessoas (BERNERS-LEE; HENDLER; LASSILA, 2001).

Breslin; Davis; Nova (2008) apontam quatro fases no crescimento da *Web*, tendo como parâmetro sua potencialidade de fazer conexões. A primeira fase (*Web* 1.0) abrange somente a conexão de informações. Já a segunda fase (*Web* 2.0) trata de conectar pessoas, colocando o “eu” na interface de usuário e o “nós” nas tecnologias de participação social. A terceira fase (*Web* 3.0) visa adicionar

significado aos dados para permitir um melhor uso dos dados disponíveis, colocando as máquinas para trabalhar para os humanos. E a última fase (*Web* 4.0), a ser vislumbrada no futuro, seria um ambiente onde humanos e máquinas trabalhariam conectados. A Figura 1 ilustra essas quatro fases da evolução da *Web*.

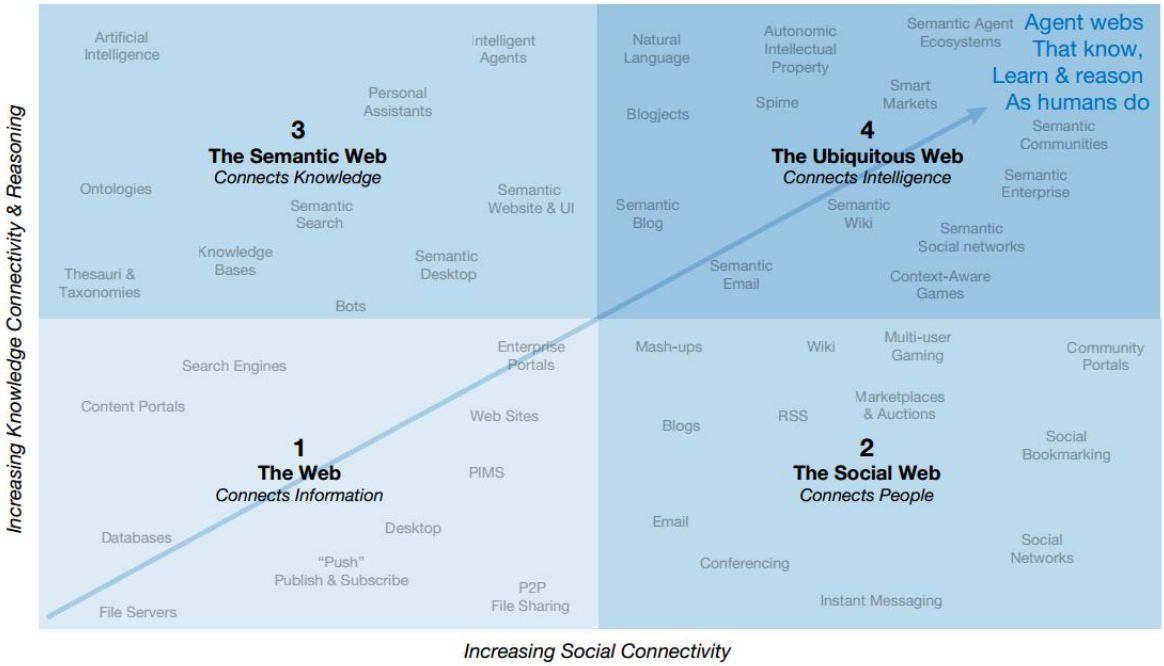

Figura 1 - A evolução da Internet.
Fonte: Breslin; Davis; Nova (2008).

Pode-se entender a *Web* Semântica como um conjunto de tecnologias e padrões que visam tornar possível que as máquinas entendam o significado, ou semântica, das informações publicadas na *Web* (YU, 2011).

Para que a *Web* Semântica se torne real, faz-se necessário atender a alguns requisitos. Jacyntho (2012) estabelece uma correlação entre estes requisitos com as tecnologias e padrões propostos pelo W3C e pode ser vista no Quadro 1.

Requisitos	**Tecnologias e Padrões**
Tem que haver um modelo para representar o conhecimento na *Web* e este modelo tem que ser facilmente processado (entendido) por máquinas.	*Resource Description Framework (RDF)*

Requisitos	Tecnologias e Padrões
Este modelo tem que ser aceito como um padrão por todos os *Web* sites; assim *statements*[3] contidos em diferentes sites, serão todos similares.	*Resource Description Framework (RDF)*
Tem que haver uma forma de adicionar estes *statements* em todos os *Web* sites, manualmente ou automaticamente.	*Semantic markup, RDFa*[4]*, Microformats*[5]*, GRDDL*[6]*, Microdata*[7]
Os *statements* contidos em diferentes *Web* sites não podem ser totalmente arbitrários. Eles devem ser criados usando termos e relacionamentos comuns, isto é, um vocabulário comum para um dado domínio.	*Ontologias Específicas de Domínio / Vocabulários*
Tem que haver uma linguagem comum para criação de vocabulários e ontologias.	*RDF Schema*[8] *(RDFS), Web Ontology Language*[9] *(OWL)*
Os agentes de software devem ser capazes de processar (entender) cada *statement* que eles coletem, com base nos termos e relacionamentos comuns usados para criar os *statements*.	*Ferramentas e Frameworks para processamento de ontologias*
Os agentes de software devem ser capazes de "raciocinar", gerando novos *statements*, com base no seu entendimento dos termos e relacionamentos comuns.	*Inferência* (em inglês, *Reasoning*) com base em ontologias

[3] *Statement* – Unidade de informação publicada em RDF, ou seja, uma tripla RDF composta por sujeito – predicado – objeto.
[4] RDFa - http://www.w3.org/TR/xhtml-rdfa-primer/
[5] Microformats - http://microformats.org/
[6] GRDDL - http://www.w3.org/TR/grddl/
[7] Microdata - https://html.spec.whatwg.org/multipage/microdata.html
[8] RDFS - http://www.w3.org/TR/rdf-schema/
[9] OWL - http://www.w3.org/TR/owl-ref/

Requisitos	Tecnologias e Padrões
Os agentes de software devem ser capazes de submeter consultas sobre os *statements* coletados.	*SPARQL*[10]
Os agentes de software devem ser capazes de inserir e remover *statements* nas bases de conhecimento.	*SPARUL*[11] *(SPARQL/Update)*

Quadro 1 - Mapeamento dos requisitos da *Web Semântica* em tecnologias e padrões.
Fonte: Jacyntho (2012).

2.1.1 Agentes inteligentes

Segundo Wooldridge (2009), agentes inteligentes são softwares projetados para realizar, autonomamente, determinadas tarefas visando um objetivo final. Um agente tem basicamente duas capacidades importantes. Em primeiro lugar, os agentes são capazes de realizar ações de maneira autônoma e, portanto, podem tomar decisões por si mesmos para atingir o objetivo para o qual foram projetados. Em segundo lugar, os agentes podem interagir com outros agentes através do compartilhamento de informações e ainda cooperar com outros agentes para realizar atividades conjuntas.

Tais agentes podem, acessando bases de conhecimento contendo fatos sobre certo domínio, conseguir analisar semanticamente o conteúdo na *Web* e realizar ações com base no conhecimento adquirido com o próprio conteúdo analisado (HENDLER, 2001).

Segundo (BERNERS-LEE; HENDLER; LASSILA, 2001), o verdadeiro poder da *Web* Semântica dar-se-á quando as pessoas começarem a criar agentes para coletar dados na *Web*, analisá-los e compartilhá-los uns com os outros. A efetividade desse cenário aumentará exponencialmente ao passo que os conteúdos da *Web* forem publicados com semântica, permitindo assim, que mais dados sejam inteligíveis pelas máquinas.

[10] SPARQL - http://www.w3.org/TR/rdf-sparql-query/
[11] SPARUL - http://www.w3.org/TR/sparql11-update/

2.1.2 Anotações semânticas

A *Web* Semântica permite que as máquinas interpretem, combinem e usem os dados publicados na *Web*. Considerando que a *Web* de documentos, anterior a *Web* de dados, possibilita que somente os humanos pudessem compreender os dados publicados, com a *Web* Semântica surge a necessidade de se incorporar semântica nesses dados de modo que os mesmos sejam compreendidos por humanos e máquinas. Para solucionar essa questão, é possível criar descrições anotando recursos ou dados com metadados, resultando em "anotações" sobre esses recursos (OREN et al., 2006).

As anotações semânticas baseadas em ontologias agregam informação aos dados ”anotados”, permitindo que a máquina incorpore novas informações às já existentes. Weinstein (1998) realizou a conversão de um catálogo bibliográfico codificado no formato *MAchine Readable Cataloging* (MARC[12]) para um modelo de ontologia, obtendo uma base de conhecimento com maior poder de expressividade, pois os dados passam a ser mapeados pelas classes e propriedades da ontologia, arraigando muito mais informações oriundas da taxonomia, relacionamentos e assertivas.

A Figura 2 apresenta um exemplo de anotação semântica incorporando uma ontologia aos dados de um texto. Nesse exemplo, um texto simples é anotado, ou seja, termos principais do texto são associados a elementos semânticos que permitem a máquina compreender que:

- a gripe e pneumonia são doenças;
- a gripe é causada pelo Influenza;
- a febre, dor de cabeça e dor muscular são sintomas e que a gripe é indicada por eles.

Com esse exemplo simples, pode-se notar que a máquina conseguiu adicionar, automaticamente, novos dados aos dados originais do texto.

[12] MARC - http://www.loc.gov/marc/umb/

A gripe é uma infecção do sistema respiratório cuja principal complicação é a pneumonia, seus principais sintomas são febre, dor muscular, dor de garganta, entre outros.

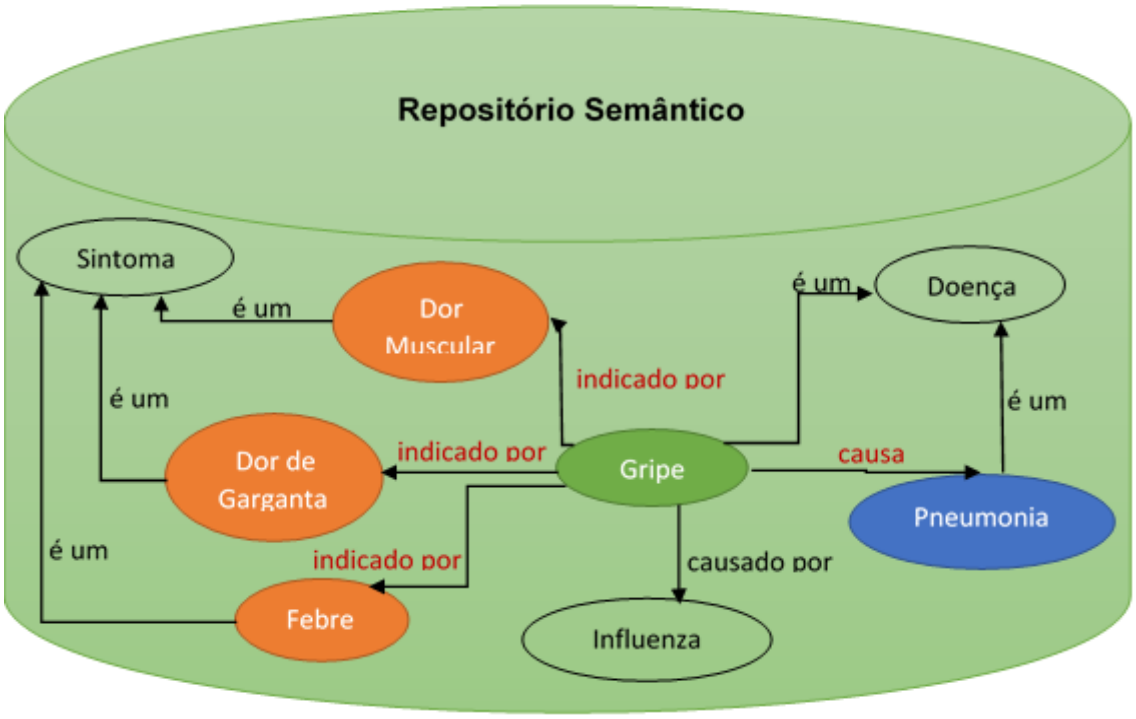

Figura 2 - Exemplo de anotação semântica aplicado a um texto simples.
Fonte: Adaptado de Fontes; De Carvalho Moura; Cavalcanti (2010).

2.1.3 Arquitetura da *Web* Semântica

Desde que (BERNERS-LEE; HENDLER; LASSILA, 2001) propôs a reformulação da *Web* de documentos para a *Web* Semântica, Berners-Lee apresentou quatro versões de arquitetura que demonstram o papel de cada tecnologia dentro da *Web* Semântica. Todas essas versões foram apresentadas por Berners-Lee em palestras, mas nunca foram publicadas na literatura ou como uma recomendação do W3C (GERBER; VAN DER MERWE; BARNARD, 2008). A Figura 3 apresenta a última versão proposta por Berners-Lee.

Para um melhor entendimento da arquitetura proposta, a seguir uma breve descrição de cada camada da Figura 3.

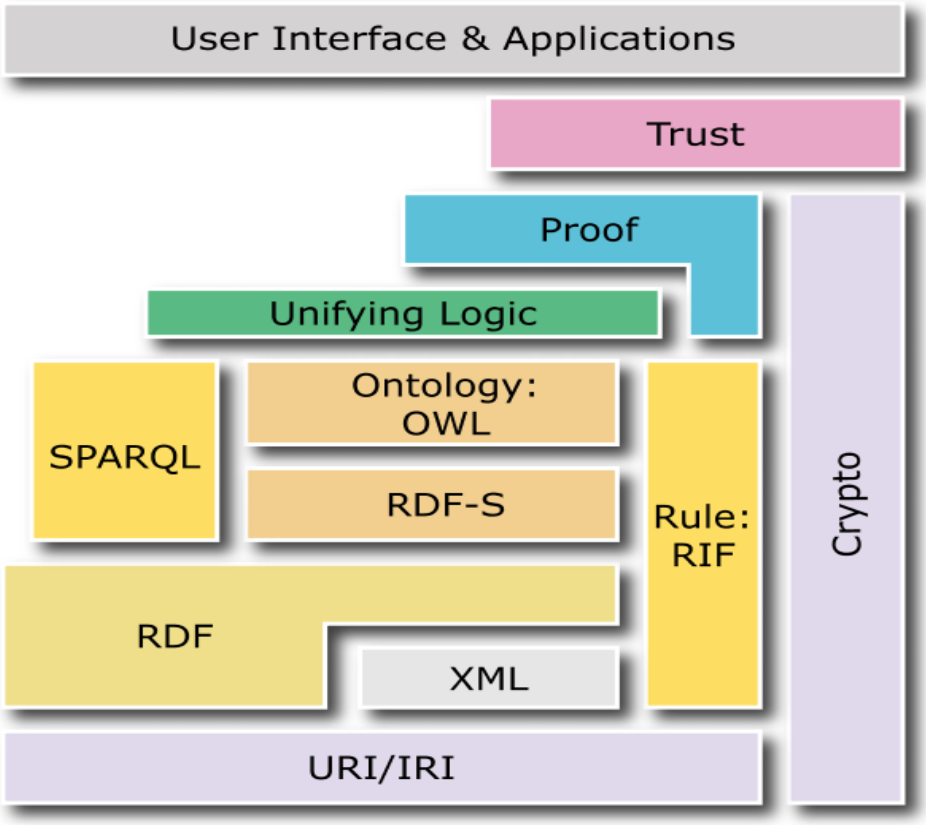

Figura 3 - Arquitetura da *Web* Semântica.
Fonte: BRATT (2007).

URI/IRI:

Uniform Resource Identifier (URI) consiste em uma cadeia de caracteres, limitados ao conjunto de caracteres ASCII, usada para identificar, unicamente, um recurso na *Web*. Essa identificação única permite a localização e referenciamento de qualquer recurso na Internet. URI exerce uma função parecida ao do ISBN para livros (GREENBERG; SUTTON; CAMPBELL, 2003). Já *Internationalized Resource Identifier* (IRI) é uma extensão de URI que abrange o conjunto de caracteres UNICODE[13] que permite ao computador trabalhar com caracteres da maioria dos sistemas de escrita existentes.

XML:

eXtensible Markup Language (XML) é uma recomendação formal do W3C e, em determinados aspectos, assemelha-se a HTML. Ambas contêm *tags* ou marcações para descrever o conteúdo de um documento *Web*. Porém, enquanto

[13] UNICODE - http://unicode.org/

HTML tem como objetivo controlar a forma com que os dados serão exibidos pelo navegador, XML tem como objetivo a descrição dos dados contidos nos documentos, definindo elementos de metadados. Além disso, com XML é possível acrescentar novas *tags*, isso a torna uma linguagem mais flexível do que HTML, permitindo criar descrições mais significativas para a diversidade de dados que são publicados na *Web*, dando a possibilidade de inserir descrições semânticas (SOUZA; ALVARENGA, 2004).

RDF:

Para possibilitar a navegabilidade e interoperabilidades entre sistemas por parte das máquinas, se faz necessário que os dados publicados na *Web* tenham uma estrutura padrão. Para suprir essa necessidade o W3C recomenda o uso do *Resource Description Framework* (RDF), que provê um modelo de representação de dados em grafo, extremamente simples, porém estritamente adaptados para a arquitetura da *Web* Semântica (HEATH; BIZER, 2011).

Em se tratando da *Web Semântica*, RDF é utilizado, não somente, para descrever metadados sobre páginas disponíveis na *Web*, mas para descrever quaisquer recursos (pessoas, documentos, empresas, eventos, produtos, paradigmas, ideologias, etc.) do mundo real, com suas propriedades e relacionamentos (JACYNTHO, 2012).

RDF tem como princípio descrever os dados e os metadados por meio de um esquema de “triplas” (*Sujeito – Predicado – Objeto*), o que permite dividir os dados em pequenas afirmações (*statements*). *Sujeito* e *Objeto* são nomes únicos (URIs) que identificam recursos, que possuem um relacionamento entre si por meio de um *Predicado*, que também é identificado por um URI. *Predicados* geralmente são definidos por uma ontologia ou *schema* (JACYNTHO, 2012; MANOLA et al., 2004).

Na Figura 4 pode ser visto um exemplo de uma tripla ou *statement* RDF.

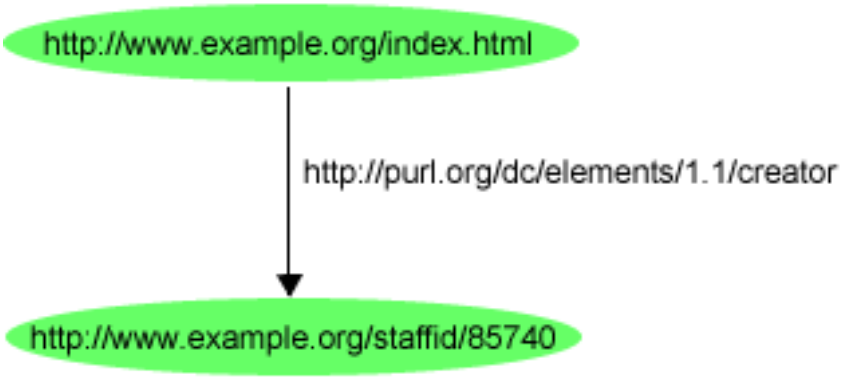

Figura 4 - Um *statement* RDF realista.
Fonte: Manola *et al.* (2004).

No exemplo exposto na Figura 4, podemos visualizar um relacionamento entre dois recursos, onde o recurso *http://www.exemple.org.index.html* tem como criador (*http://purl.org/dc/elements/1.1/creator*) o recurso *http://www.exemple.org/staffid/85740*. Uma outra denominação para os elementos da tripla RDF poderia ser *Recurso – Propriedade - Valor*, onde a propriedade pode ter como valor um outro recurso, como na Figura 4, ou a pode ter como valor um dado literal (ou primitivo), descrevendo assim um atributo simples. Na Figura 5 é possível visualizar um grafo RDF com três triplas sobre um único recurso (*http://www.exemple.org/index.html*) onde esse recurso se relaciona com outro recurso e com dados literais.

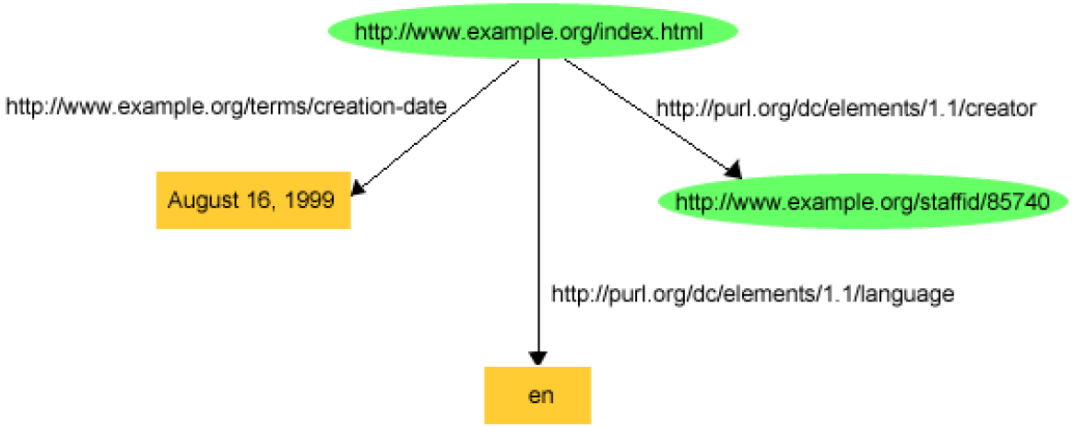

Figura 5 - *Statements* representando relacionamentos e atributos.
Fonte: Manola *et al*. (2004).

A Figura 6 mostra o mesmo grafo exposto na Figura 5, porém fazendo uso da notação em triplas.

```
<http://example.org/index.html> <http://purl.org/dc/elements/1.1/> <http://example.org/staffid/85740> .

<http://example.org/index.html> <http://www.exemple.org/terms/creation-date> "August 16, 1999" .

<http://example.org/index.html> < http://purl.org/dc/elements/1.1/language> "en" .
```

Figura 6 - Notação em triplas.
Fonte: Adaptado de Manola *et al*. (2004).

As descrições RDF visualizadas até então foram apresentadas em um modelo abstrato de grafo que descrevem os recursos. Porém, dados em RDF, para serem efetivamente publicados na *Web*, precisam ser codificados em sintaxes concretas. A sintaxe mais utilizada e recomendada pelo W3C é RDF/XML[14], que possibilita a serialização de um grafo RDF em um documento no formato XML. Mas existem outras sintaxes mais amigáveis para seres humanos. São elas, ordenadas da mais complexa para a mais simples: *Notation* 3[15] (N3), *Turtle*[16] e *N-Triples*[17]. Dentre estas, *Turtle* é bastante popular entre desenvolvedores e foi escolhida como base da sintaxe da linguagem de consulta SPARQL (JACYNTHO, 2012; PRUD'HOMMEAUX; SEABORNE, 2008). Por fim, mais recentemente surgiu uma sintaxe baseada em JSON, que vem ganhando espaço, denominada JSON-LD[18].

A Figura 7 mostra o mesmo grafo RDF exibido na Figura 5, porém na sintaxe *Turtle*, note que em *Turtle* é possível criar prefixos, isso permite a redução do tamanho dos URIs ao se descrever as triplas, facilitando a leitura e interpretação por parte dos seres humanos.

```
@prefix dc:     <http://purl.org/dc/elements/1.1/> .
@prefix ex:     <http://example.org/terms/> .
@prefix :       <http://example.org/> .

:index.html    dc:creator      :staffed/85740 ;
               dc:language     "en" ;
               ex:creation-date  "August 16, 1999 .
```

Figura 7 - Notação em Turtle.
Fonte: O autor

Heath e Bizer (2011) enumeram os principais benefícios ao se fazer uso do modelo de representação de dados RDF. São eles:

[14] RDF/XML - http://www.w3.org/TR/rdf-syntax-grammar/
[15] N3 - http://www.w3.org/DesignIssues/Notation3
[16] Turtle - http://www.w3.org/TeamSubmission/turtle/
[17] N- Triples - http://www.w3.org/TR/n-triples/
[18] JSON-LD - http://json-ld.org/

a) Por usar HTTP URIs, para representar recursos e propriedades, o modelo RDF é inerentemente projeto para ser altamente escalável e possibilita qualquer pessoa, de qualquer lugar do mundo, referenciar qualquer “coisa” na Internet.
b) A máquina pode obter dados de diversas fontes através de URIs, isso permite agregar dados automaticamente de diversas fontes disponíveis na *Web*.
c) O modelo de dados RDF habilita configurar *links* (RDF *Links*) com outras fontes de dados.
d) Informações de diferentes fontes de dados podem facilmente serem fundidas em um único grande grafo RDF.
e) RDF é um modelo de dados semiestruturado, ou seja, permite representar os dados com diferentes esquemas ou ontologias, viabilizando agregação de dados descritos com ontologias diferentes.
f) RDF combinado com linguagens de ontologias, como RDFSchema e OWL, permite a estruturação dos dados incorporando-lhes semântica.

RDFS:

RDFSchema (RDFS) (BRICKLEY; GUHA, 2004) é um vocabulário que provê termos que permitem a definição de outros vocabulários (ontologias). Com o RDFS é possível criar classes e propriedades para representar um determinado domínio do conhecimento. Os termos definidos pelo RDFS possuem URIs que são precedidas de: http://www.w3.org/2000/01/rdf-schema#. Usualmente, este prefixo é associado com o prefixo de *namespace* rdfs.

***Ontology* – OWL:**

É através da ontologia que se torna possível adicionar semântica aos dados para que a máquina consiga raciocinar e inferir novos conhecimentos. Por ser o principal tema trabalhado nesta dissertação, será melhor apresentado na seção 2.2.

SPARQL:

Com a massiva publicação de dados em RDF, faz-se necessário ter uma forma que possibilite buscas/pesquisas em cima desses dados para que seja possível extrair informação de uma maneira mais precisa e eficiente. Para isso o W3C recomenda a linguagem SPARQL (HARRIS; SEABORNE; PRUD'HOMMEAUX, 2013).

SPARQL é um acrônimo recursivo para *SPARQL Protocol and RDF Query Language*. Portanto, SPARQL é uma linguagem de consulta e um protocolo de acesso para dados RDF. Assim, como o SQL é uma linguagem para consultar dados relacionais, o SPARQL é uma linguagem para consultar dados RDF.

As consultas SPARQL são baseadas nos conceitos de *triple pattern* e *graph pattern*, ou seja, para que os dados sejam selecionados, faz necessário que estes correspondam ao modelo fornecido na consulta SPARQL.

Um *triple pattern* é semelhante à uma tripla RDF, porém cada elemento da tripla pode ser substituído por uma variável (prefixada pelo caractere "?" ou "$") e esta pode assumir valores diversos, desde que se enquadrem no padrão definido pelo *triple pattern*. Já um conjunto de *triple patterns* formam um *graph pattern*. Um *graph pattern* permite a criação de consultas mais complexas, onde variáveis podem figurar em vários *triples patterns* e, ao se executar a consulta, esta variável receberá o mesmo valor em todos os triples *patterns* em que aparece (JACYNTHO, 2012).

A Figura 8 mostra um exemplo de consulta SPARQL SELECT onde se deseja obter o título de um livro. Pode ser notado, nesse exemplo, o uso da variável `?title`, que pode assumir qualquer valor. Caso haja dentro dos dados pesquisados uma tripla que atenda ao modelo `<http://exemple.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title`, onde `?title` pode ser qualquer valor, essa tripla será retornada como resultado. Na cláusula SELECT está declarado que somente será exibido o valor da variável `?title`, assim, o resultado da consulta será somente o título do livro.

Data:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Query:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

This query, on the data above, has one solution:

Query Result:

title
"SPARQL Tutorial"

Figura 8 - Consulta SPARQL com *triple pattern.*
Fonte: *Harris; Seaborne; Prud'hommeaux (2013)*.

Uma consulta mais complexa é ilustrada na Figura 9. Nesse exemplo deseja-se retornar os valores das propriedades `foaf:name` e `foaf:mbox` de todos os recursos que possuam, ao mesmo tempo, estas duas propriedades. Ao se observar os dados, pode-se verificar que dois recursos satisfazem o critério de seleção (*graph pattern*): o recurso `_:a` e `_:b`, pois esses dois recursos possuem nome (`foaf:name`) e e-mail (`foaf:mbox`). Já o recurso `_:c` não possui nome (`foaf:name`) e por isso não é retornado como resultado.

Data:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .
```

Query:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
```

Query Result:

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Figura 9 - Consulta SPARQL com *graph pattern*.
Fonte: *Harris; Seaborne; Prud'hommeaux (2013)*.

***Rule* – RIF:**

Regra (Rule) é a prescrição de algo que deve ser seguido. Segundo Morgenstern; Welty; Boley (2013), na computação existem dois tipos de regras: Regras de Produção e Regras Declarativas.

As regras de produção estão intimamente relacionadas com instruções de um programa de computador, ou seja, se alguma condição é válida, implica que alguma coisa deve ser feita.

As regras declarativas representam fatos sobre o mundo, não descrevendo regras que dizem uma ação a ser feita em certas condições, mas descrevem como as "coisas" são. Em outras palavras, uma regra declarativa descreve como o mundo é, ao invés de prescrever como as coisas deveriam ser.

Existem diversas linguagens para descrever regras como SILK[19], OntoBroker[20], Eye[21], VampirePrimer[22], N3-Logic[23] e SWRL[24] destinadas a descrever regras declarativas e Jess[25], Drools[26], IBM ILog[27] e Oracle Business Rules[28], estas destinadas a descrever regras de produção.

Devido a essa diversidade de linguagens, o W3C propôs o padrão *Rule Interchange Format* (RIF), um conjunto de linguagens ou dialetos, que visa prover meios para interoperabilidade entre diversos conjuntos de regras. A família de dialetos RIF destina-se a ser uniforme e extensível. A uniformidade de RIF significa que se espera que dialetos partilhem, tanto quanto possível, dos aparatos semânticos e sintáticos existentes. Já a extensibilidade de RIF significa que deve ser possível para especialistas definir um novo dialeto RIF como uma extensão sintática de um dialeto RIF existente (KIFER; BOLEY, 2010).

***Unifying Logic*:**

[19] SILK - http://silk.semwebcentral.org/
[20] OntoBroker - http://www.semafora-systems.com/en/products/ontobroker/
[21] Eye - http://eulersharp.sourceforge.net/
[22] Vampire - http://www.vprover.org/
[23] N3-Logic - http://www.w3.org/DesignIssues/N3Logic
[24] SWRL - www.w3.org/Submission/SWRL
[25] Jess - http://www.jessrules.com/
[26] Drools - www.drools.org/
[27] IBM ILog - www.ibm.com/software/info/ilog
[28] Oracle Business Rules - http://www.oracle.com/technetwork/middleware/business-rules/overview/index.html

O intento dessa camada é fornecer uma interface única que englobe os diferentes modelos semânticos disponíveis nas camadas abaixo (RDF, RDFS, SPARQL, OWL e RIF). Isso permite que aplicações semânticas sejam facilmente desenvolvidas, pois são escritas para uma única interface e não para as diversas tecnologias que compõe a *Web* Semântica (POLLOCK, 2009).

***Proof e Trust*:**

A prova (*Proof*) visa validar a coerência lógica, para garantir que as descrições semânticas e os dados inferidos pelas aplicações semânticas estejam corretos. Já a confiança (*Trust*) destina-se a verificar a confiabilidade dos dados de modo que se tenha níveis de confiabilidade para os dados. Isso permite que as aplicações e pessoas selecionem melhor os dados que desejam usar (POLLOCK, 2009).

***Crypto*:**

Abrangendo grande parte das camadas da arquitetura da *Web* Semântica, a criptografia (*crypto*) tem como objetivo garantir a privacidade e a segurança das informações dispostas nos diversos níveis (JARDIM, 2010).

User Interface & Applications

Dados com anotações semânticas permitem que a máquina os compreenda e gere novos conhecimentos com base neles. Esses dados podem ser sobre um usuário, descrevendo assim, seu perfil. Isso permite que a máquina customize aplicações e interfaces de acordo com as preferências de cada usuário. Contribui para esse fim, o desenvolvimento semi-automatizado de ontologias pessoais com o uso de vocabulários controlados como o *Dublin Core* (DC) e o *Friend of a Friend* (FOAF) (PALAZZO *apud* JARDIM, 2010).

Todas essas tecnologias que foram descritas nessa seção visam permitir que a *Web* de Dados se torne real, com recursos sendo identificados mundialmente, dados sendo publicados em um padrão amplamente flexível e escalável, semântica sendo adicionada aos dados que permitam à máquina inferir

conhecimentos novos e meios que proveja validade, confiabilidade e privacidade dos dados.

2.1.4 *Linked Data*

O conceito de *Linked Data* consiste em publicar dados na *Web* de modo estruturado e interligando-os com outras fontes de dados, permitindo que a máquina “navegue” por esses dados e os compreenda (BIZER; HEATH; BERNERS-LEE, 2009).

Aplicando esse conceito, a *Web* de documentos destinada a humanos, se transformaria na *Web* de dados, destinada às máquinas, resultando em um banco de dados global.

Para que seja possível a criação desse grande banco de dados, Berners-Lee (2006), propõe algumas regras básicas para publicar e interligar dados na *Web*, a saber:

1. Use URIs para nomear coisas.
2. Use HTTP URIs para que um cliente (homem ou máquina) possa acessar estes nomes na *Web*.
3. Toda URI deve ser dereferenciável. Em outras palavras, quando alguém acessar este URI na *Web*, alguma informação útil deve sempre ser retornada, usando os padrões da Web Semântica (RDF, SPARQL).
4. Inclua links para outras URIs, para que o cliente possa descobrir novas coisas. Ou seja, navegar por RDF *links* entre diferentes fontes de dados.

Para publicar os dados na *Web* de modo estruturado, é utilizado o modelo RDF, que traz uma representação dos dados em grafos em formato de triplas: sujeito – predicado – objeto (MANOLA et al., 2004). De acordo com Heath e Bizer (2011), qualquer dado pode ser publicado em formato de triplas, permitindo que, independente de sistemas e domínios, esses dados sejam compartilhados e reusados por toda a *Web*, uma vez que todos os dados estão descritos sob um mesmo modelo padrão.

Jacyntho (2012) ressalta que, somando-se às quatro regras anteriores, é de extrema importância o reuso, sempre que possível, de URIs e de vocabulários

(ontologias) preexistentes, pois auxilia no processo de interconexão e compartilhamento dos dados entre as diversas fontes de dados publicadas.

O projeto *Linking Open Data Community* [LOD Project][29] tem contribuído significativamente para consolidação da *Web* de Dados, onde diversas bases de dados são interligadas seguindo as orientações citadas acima. A Figura 10 mostra os conjuntos de dados que foram publicados em formato *Linked Data*, conhecida como *LOD cloud diagram*[30].

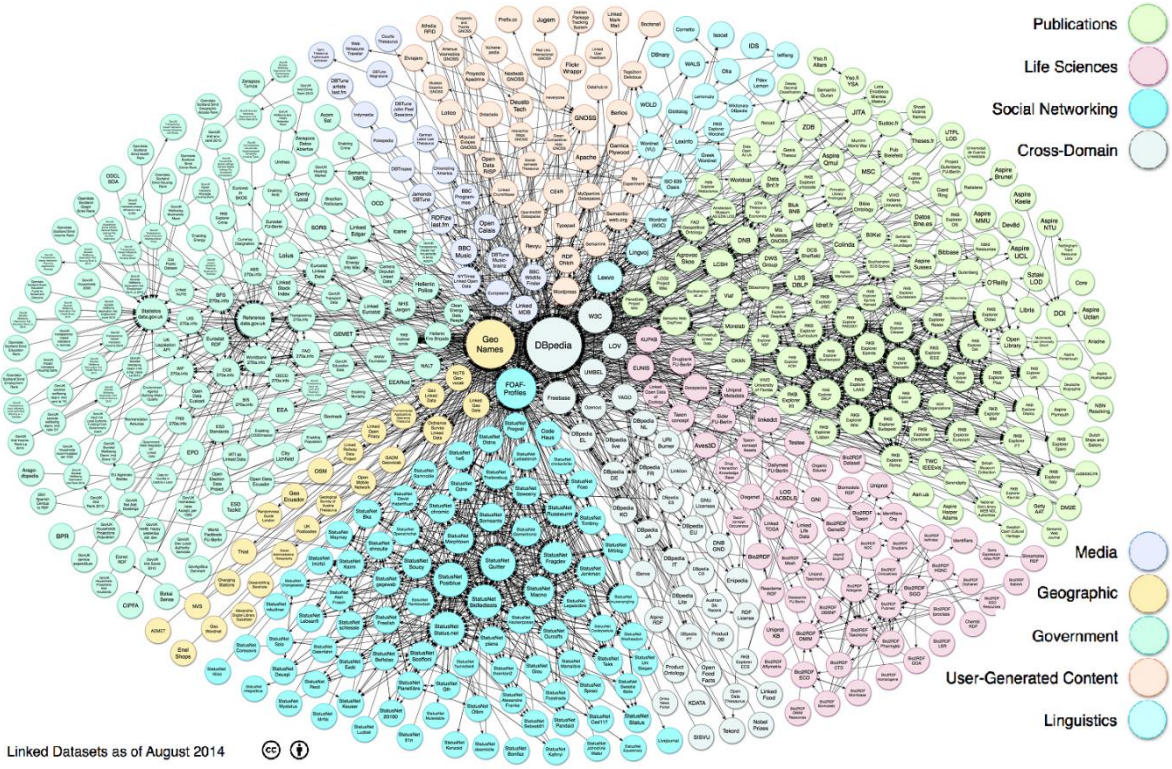

Figura 10 - *LOD cloud diagram* em abril de 2014
Fonte: *LOD Project*

2.2 ONTOLOGIAS

Uma vez que a principal contribuição deste trabalho é uma ontologia, o objetivo desta seção é apresentar este tema com uma cobertura mais abrangente, e, portanto, para cada tópico, detalhes suficientes serão oferecidos de tal forma que o leitor possa adquirir um conhecimento razoavelmente completo, sem precisar recorrer a fontes bibliográficas externas.

[29] LinkingOpenData - http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[30] LOD Cloud Diagram - http://lod-cloud.net/

Para que haja um compartilhamento dos dados publicados em diversos *Web sites*, faz-se necessário usar um vocabulário comum entre os dados publicados.

Caso os dados sejam publicados com vocabulários diversos, sem qualquer relacionamento, torna-se impraticável e extremamente difícil a máquina fazer correlações entre eles.

Para solucionar esse problema, é necessário criar vocabulários ou ontologias para serem reusadas pelas diversas fontes de dados RDF, na descrição dos seus dados. Isso permite que a máquina compreenda os dados publicados nas diversas bases e consiga compartilhá-los e/ou agregá-los.

Quando se descreve um recurso em triplas RDF, as propriedades (predicados) devem pertencer a uma ontologia e sempre que possível essa ontologia deve ser reusada, para o domínio de conhecimento em questão, visando interoperabilidade entre as diversas bases. Assim, uma linguagem ou vocabulário comum, com classes e propriedades pré-definidas, precisa ser definido, publicado e reusado. Este vocabulário permite às máquinas fazerem inferências, com base no conhecimento descrito nele.

2.2.1 Definição

Originalmente o termo “ontologia” designa uma área de estudo da filosofia, cujo objetivo é estudar os princípios e fundamentos últimos dos seres ou essência de toda a realidade (CHAUI, 1995).

Smith *apud* Viinikkala (2004) define ontologia filosófica como sendo a ciência do “o quê”, dos tipos e estruturas de objetos, propriedades, eventos, processos e relações em todas as áreas da realidade.

Studer; Benjamins; Fensel (1998) relata que desde o início dos anos 1990, as ontologias têm se popularizado como objeto de estudo em algumas áreas da inteligência artificial, como engenharia do conhecimento, processamento de linguagem natural e representação do conhecimento.

No contexto da *Web* Semântica, Heflin (2004) expõe que uma ontologia define formalmente um conjunto comum de termos que são utilizados para descrever e representar um domínio, ou ainda, que uma ontologia define os termos utilizados para descrever e representar uma área de conhecimento.

Uma ontologia representa um determinado domínio de conhecimento, definindo seus principais conceitos ou classes e relacionamentos entre essas classes, representando hierarquias (superclasses e subclasses). Outros relacionamentos também são encontrados nas ontologias, definidos por meio de propriedades que descrevem características ou atributos das classes, e também propriedades que relacionam instâncias das classes. Assim, uma ontologia codifica o conhecimento de um determinado domínio em um formato estruturado que possibilita a máquina entender esse conhecimento, viabilizando a *Web Semântica* (JACYNTHO, 2012).

2.2.2 Benefícios

Como visto na seção anterior, ontologia é uma representação formal de um domínio de conhecimento, utilizada pela máquina para compreender os dados publicados na *Web*.

A seguir serão listados os principais benefícios propiciados com o uso de ontologias na ciência da computação (HINZ, 2006):

- Ontologias fornecem um vocabulário para representação do conhecimento. Esse vocabulário possui uma conceitualização que o sustenta, evitando, assim, ambiguidades na sua interpretação.
- Ontologias permitem o compartilhamento de conhecimento. Uma vez que uma ontologia foi modelada para um dado domínio de conhecimento, é possível reutilizá-la para outras aplicações que também trabalham no mesmo domínio, evitando retrabalho.
- Uma ontologia fornece uma descrição exata do conhecimento, ou seja, numa linguagem natural pode ocorrer interpretações diferentes dependendo do contexto, porém, como a ontologia é descrita numa linguagem formal, não possibilita que ocorra interpretações semânticas errôneas.
- Uma ontologia pode ser expressa em várias linguagens, sem que para isso haja danos da sua conceitualização.
- Ontologias podem representar domínios de conhecimento mais abrangentes ou genéricos. Isso pode resultar em uma representação ruim para um domínio mais específico, sendo, portanto, possível especializar ontologias

para que melhor se adequem às especificidades de domínios mais peculiares.

Além desses benefícios, as ontologias permitem reuso, compartilhamento e integração de conhecimento, pois dados de diversas fontes, descritos por uma mesma ontologia, permitem que a máquina os compreenda e os integre, automaticamente, em uma grande base de dados (*mashup* semântico).

2.2.3 Do que se constitui uma ontologia?

Lassila e McGuinness (2001) salientam que uma ontologia deve possuir:

- Características obrigatórias:
  a) Vocabulário controlado finito dos termos extensíveis.
  b) Interpretação inequívoca de classes e seus relacionamentos.
  c) Relacionamentos hierárquicos estritos entre classes e subclasses.
- Características típicas, mas não obrigatórias:
  a) Especificação de propriedades nas classes.
  b) Inclusão de indivíduos ou instâncias da ontologia.
  c) Especificação de restrições (axiomas).
- Características desejáveis, mas não obrigatórias e nem típicas:
  a) Especificação de classes disjuntas.
  b) Especificação de relacionamentos lógicos arbitrários entre termos.

Para Gruber (1995), uma ontologia é composta por: classes, que são os principais termos de um dado domínio de conhecimento; relações entres essas classes; funções, que são formas especiais de relacionamento entre os termos; axiomas que representam afirmações sobre os termos do domínio; e instâncias, que são os indivíduos da ontologia.

Segundo NOY e MCGUINNESS (2001), uma ontologia representa um determinado domínio de conhecimento e consiste em: classes, que representam os principais conceitos do domínio; propriedades que representam relacionamentos e atributos das classes; axiomas, que são regras de restrição sobre o domínio e instâncias das classes.

Assim, em conformidade com os autores citados, ontologias são estruturas que possuem *Classes*, *Propriedades*, *Axiomas* (restrições e regras) e *Instâncias*.

Classes – representam os principais termos de um dado domínio do conhecimento. As classes agrupam indivíduos que possuem algo em comum. As classes podem se relacionar com outras classes através de uma relação hierárquica (superclasse – subclasse), representando uma taxonomia, que relaciona os termos mais genéricos com termos mais específicos (tem-tipo, é-um). Esse relacionamento provê à máquina o poder de atribuir classificações novas aos recursos. A Figura 11 ilustra uma hierarquia de classes.

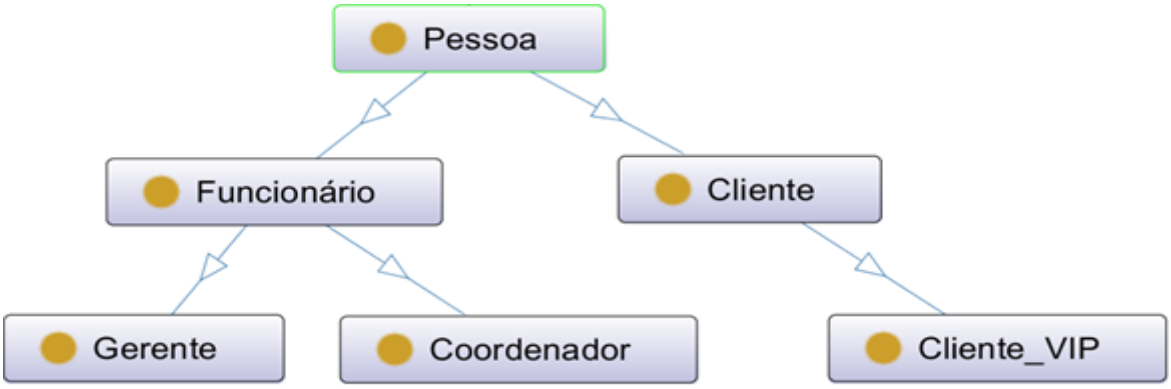

Figura 11 - Exemplo de hierarquia de classes
Fonte: O autor

Propriedades – são utilizadas para descrever uma entidade do domínio. As propriedades podem ser usadas para descrever atributos de um indivíduo (*Datatype Property*), relacionando-o com valores literais ou ainda ser relações binárias entre indivíduos (*Object Property*).

Na Figura 12, pode ser visto o uso de uma *Datatype Property* para relacionar um recurso (`http://www.exemplo.com/matheus`) com alguns valores que representam atributos do recurso, como nome, e-mail e idade. Note que as *Datatype Properties* relacionam um recurso com valores literais, como números, textos, valores lógicos e pré-definidos (opções preexistentes).

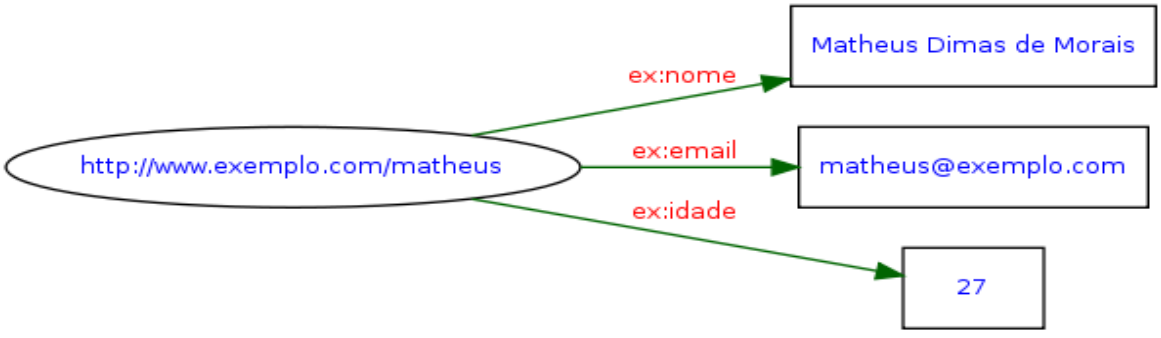

Figura 12 - Exemplo de *Datatype Property*
Fonte: O autor

Já na Figura 13, é possível visualizar o uso de duas *Object Properties* que relacionam um recurso com outros dois. A *Objetc Property* `ex:trabalhaCom` está relacionando o recurso `http://www.exemplo.com/matheus` com o recurso `http://www.exemplo.com/wesley`, indicando uma relação que um trabalha com o outro. Em conclusão, *Object Property* relaciona recursos com outros recursos, isto é, as *Object Properties* podem ser usadas para relacionar classe com classe (hierarquia entre classes), instância com instância (relacionamento entre indivíduos) ou instância com classe (classificação de indivíduos).

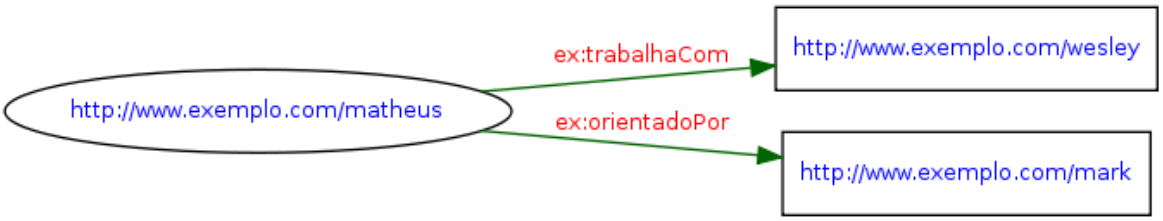

Figura 13 - Exemplo de Object Property
Fonte: O autor

Toda propriedade possui um domínio (*domain*) e um contradomínio (*range*). Tanto domínio quanto contradomínio são definidos por um conjunto de uma ou mais classes. Domínio se refere ao sujeito de qualquer tripla que use a propriedade, e contradomínio se refere ao objeto. Em outras palavras, quando usamos a propriedade, a máquina pode inferir que o sujeito da tripla é membro da(s) classe(s) do domínio e o objeto da tripla é membro da(s) classe(s) do contradomínio. Por exemplo, na sentença "`Matheus trabalha como Professor`", supondo que a propriedade "`trabalha como`" tenha sido definida tendo como domínio a classe `Pessoa` e como contradomínio a classe `Profissão`, a máquina, automaticamente, deduziria que o recurso `Matheus` é do tipo `Pessoa` e que o recurso *`Professor`* é do tipo `Profissão`. Para melhor compreensão, a Figura 14 ilustra esta definição.

Axiomas (restrições e regras) – especificam definições sobre os termos de um dado domínio e restrições sobre sua interpretação. Os axiomas especificam, através de lógica de primeira ordem, sentenças que são sempre verdadeiras dentro do domínio representado pela ontologia (ALMEIDA; BAX, 2003; FALBO, 1998).

São os axiomas que conferem um maior poder de dedução e inferência para as máquinas, propiciando a descoberta de novos conhecimentos. Como exemplo de axioma, pode-se citar: "Toda pessoa que é mãe, é uma mulher".

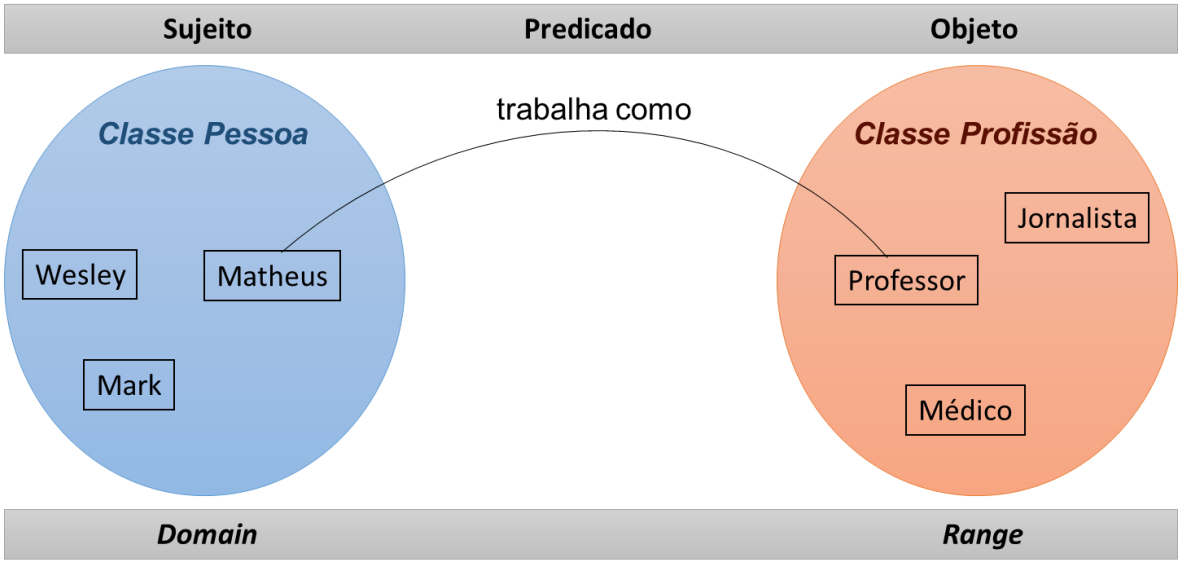

Figura 14 - Ilustração de Domain e Range de uma propriedade
Fonte: O autor

Instâncias – são indivíduos concretos que pertencem a uma classe. Como foi dito, a classe representa as características de um determinado grupo de indivíduos e os indivíduos são instâncias das classes. Por exemplo, a classe `Instituição_de_ensino` pode ter como instância o indivíduo `Instituto_Federal_Fluminense`, onde na classe ficam descritas as características comuns às diversas instituições de ensino e o indivíduo possui atributos e relacionamentos específicos dele. Na Figura 14, por exemplo, os elementos “Matheus”, “Mark” e “Wesley” são indivíduos (instâncias) da classe “Pessoa”.

2.2.4 Tipos de ontologia

Existem vários tipos de ontologia e essa tipificação leva em consideração sua função, seu grau de formalismo referente ao seu vocabulário, sua aplicação e sua estrutura e conteúdo da conceitualização (ALMEIDA; BAX, 2003).

O Quadro 2 sintetiza a revisão realizada por Almeida e Bax (2003), onde é descrito cada tipo de ontologia conforme as tipificações encontradas na literatura.

Abordagem	Classificação	Descrição
Quanto à função (MIZOGUCHI; VANWELKENHUYSEN; IKEDA, 1995)	Ontologias de domínio	Reutilizáveis no domínio, fornecem vocabulário sobre conceitos, seus relacionamentos, sobre atividades e regras que os governam.
	Ontologias de tarefa	Fornecem um vocabulário sistematizado de termos, especificando tarefas que podem ou não estar no mesmo domínio.
	Ontologias gerais	Incluem um vocabulário relacionado.
Quanto ao grau de formalismo (USCHOLD; GRUNINGER, 1996)	Ontologias altamente informais	Expressa livremente em linguagem natural.
	Ontologias semi-informais	Expressa em linguagem natural de forma restrita e estruturada.
	Ontologias semiformais	Expressa em linguagem artificial definida formalmente.
	Ontologia rigorosamente formal	Os termos são definidos com semântica formal, teoremas e provas.
Quanto à aplicação (JASPER; USCHOLD, 1999)	Ontologias de autoria neutra	Um aplicativo é escrito em uma única língua e depois é convertido para uso em diversos sistemas, reutilizando-se as informações.

Abordagem	Classificação	Descrição
	Ontologias como especificação	Cria-se uma ontologia para um domínio, a qual é usada para documentação e manutenção no desenvolvimento de softwares.
	Ontologia de acesso comum à informação	Quando o vocabulário é inacessível, a ontologia torna a informação inteligível, proporcionando vocabulário compartilhado dos termos.
Quanto à estrutura (HAAV; LUBI, 2001)	Ontologia de alto-nível	Descrevem conceitos gerais relacionados a todos os elementos da ontologia (espaço, tempo, matéria, objeto, evento, ação, etc.) os quais são independentes do problema ou domínio.
	Ontologia de domínio	Descrevem o vocabulário relacionado ao domínio, como, por exemplo, medicina ou automóveis.
	Ontologia de tarefa	Descrevem uma tarefa ou atividade, como, por exemplo, diagnósticos ou compras, mediante inserção de termos especializados na ontologia.
Quanto ao conteúdo (VAN HEIJST; SCHREIBER; WIELINGA, 1997)	Ontologias terminológicas	Especificam termos que serão usados para representar o conhecimento em um domínio (por exemplo, os léxicos).

Abordagem	Classificação	Descrição
	Ontologias de modelagem do conhecimento	Especificam conceituações do conhecimento, tem uma estrutura interna semanticamente rica e são refinadas para uso no domínio do conhecimento que descrevem.
	Ontologias de aplicação	Contém as definições necessárias para modelar o conhecimento em uma aplicação.
	Ontologias de domínio	Expressam conceituações que são específicas para um determinado domínio do conhecimento.
	Ontologias genéricas	Similares às ontologias de domínio, mas os conceitos que as definem são considerados genéricos e comuns a vários campos.
	Ontologias de representação	Explicam as conceituações que estão por trás dos formalismos de representação do conhecimento

Quadro 2 - Tipos de ontologias.
Fonte: Almeida e Bax (2003)

2.2.5 Linguagem OWL

São as ontologias que proveem a semântica para os dados publicados em RDF, ou seja, os dados publicados em RDF estão estruturados de uma forma que o computador consiga ler, navegar, agregar e compartilhar, porém para que o computador consiga entender o que representa aqueles dados e como inferir novos conhecimentos a partir deles, é necessário embutir sentido aos dados. A semântica

deve também estar estruturada em RDF para que a máquina consiga navegar tanto pelos dados publicados quanto pelos dados que dão semântica a eles.

Assim, para se criar uma ontologia, também é utilizado o padrão RDF juntamente com esquemas (linguagens de ontologias ou metaontologias) que permitam a definição dos diversos elementos que compõem a ontologias (classes, propriedades, axiomas e indivíduos).

Como já dito anteriormente, RDFSchema (RDFS) provê termos para descrever ontologias simples, com classes, hierarquia de classes, propriedades e tipos de dados. A seguir, são apresentados os termos RDFS agrupados pelas suas destinações (BRICKLEY; GUHA, 2014):

- Termos usados para definir classes e hierarquia de classes:
  `rdfs:Resource, rdfs:Class, rdfs:subClassOf`.
- Termos usados para definir propriedades, domínio e escopo, hierarquia de propriedades e tidos de dados:
  `rdfs:range, rdfs:domain, rdfs:Literal, rdfs:Datatype, rdfs:subPropertyOf`.
- Termos de usados para descrever, para humanos, os elementos da ontologia e relacionamento com outras ontologias:
  `rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs:isDefinedBy`.

Assim, RDF e RDFS fornecem um conjunto de termos que permitem a modelagem de ontologias simples, porém para se construir uma ontologia com maior poder de expressividade, faz-se necessário usar relações mais complexas que envolvem classes (como disjunções e equivalências), propriedades (como as relações simétricas, inversas e transitivas) e também restrições de valor e cardinalidade para as relações (ALLEMANG; HENDLER, 2011).

Para suprir essa deficiência, o W3C recomenda, desde 2004, a utilização da linguagem *Web Ontology Language* (OWL) (MCGUINNESS; VAN HARMELEN, 2004).

Atualmente a linguagem OWL já foi atualizada e está na segunda versão. OWL 2 (W3C - OWL WORKING GROUP, 2012) é uma linguagem projetada para representar um conhecimento rico e complexo sobre as “coisas”, grupos de “coisas”, e as relações entre as “coisas”. OWL 2 é baseada em lógica computacional de tal forma que o conhecimento expresso possa ser “raciocinado”

pela máquina, seja para verificar a consistência do conhecimento expresso, seja para tornar o conhecimento implícito em explícito. Ontologias OWL são documentos RDF que podem ser publicados na *Web* e podem fazer uso ou ser usados por outras ontologias (HITZLER et al., 2012).

OWL pode ser dividida em três sublinguagens: OWL *Lite*, OWL DL e OWL *Full*. Onde a sublinguagem OWL *Lite* é menos expressiva que a OWL DL e esta última, menos expressiva que a OWL *Full* (MCGUINNESS; VAN HARMELEN, 2004).

- OWL *Lite* – permite a descrição de classificação hierárquica e restrições simples. Embora dê suporte a restrições de cardinalidade, estas estão limitadas a valores de cardinalidade 0 ou 1. Sua adoção permite, inclusive, a migração rápida de tesauros e outras taxonomias para o formato de ontologias.
- OWL DL – possui o máximo de expressividade, porém mantendo a integridade computacional (todas as conclusões são garantidas de serem computáveis) e decidibilidade (todos os cálculos vão terminar em tempo finito). OWL DL possui todas as construções da linguagem OWL, no entanto, ela exige a separação de tipos, como classe, propriedade e indivíduo. OWL DL é assim chamado devido à sua correspondência com lógica descritiva, um campo de pesquisa que estuda a lógica que forma a base formal da OWL.
- OWL *Full* - garante o máximo de expressividade e a liberdade sintática do RDF, porém sem a garantia de integridade computacional. OWL *Full* permite aumentar o significado de um vocabulário (RDF ou OWL). Por exemplo, uma classe em OWL *Full* pode ser tratada, ao mesmo tempo, como uma coleção de indivíduos ou como um simples indivíduo.

Segundo (HITZLER et al., 2012; WELTY; MCGUINNESS; SMITH, 2004), OWL reusa alguns vocabulários e *schemas* para definição de seus construtos (que podem ser vistos no Quadro 3).

Prefix (XML namespace)	URI
owl	http://www.w3.org/2002/07/owl#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#"
rdfs	http://www.w3.org/2000/01/rdf-schema#
xsd	http://www.w3.org/2001/XMLSchema#

Quadro 3 - Lista de prefixos de *namespace* utilizados para construção de ontologias OWL
Fonte: O autor

Classes:

Uma classe é uma forma de abstração que serve para agrupar indivíduos com características semelhantes. Toda classe OWL é necessariamente subclasse da classe `owl:Thing` (conjunto universo), ou seja, todos os indivíduos das classes constituintes do domínio representado farão parte dessa classe. A linguagem OWL define também a classe `owl:Nothing` que representa um conjunto vazio.

Para se declarar uma classe, basta criar uma instância da metaclasse `owl:Class`. Um exemplo pode ser visto no fragmento de código a seguir:

```
<owl:Class rdf:about="http://www.exemplo.com#Pessoa"/>
```

No exemplo acima, é utilizado a sintaxe definida em RDF/XML (`rdf:about`) para definir o URI da classe `Pessoa` (`http://www.exemplo.com#Pessoa)` e o fato dessa propriedade estar dentro do elemento <`owl:Class`> determina que o recurso `http://www.exemplo.com#Pessoa` é membro da classe `owl:Class` e por isso é uma classe OWL. Uma outra sintaxe permite abreviar os URI para simplificar as declarações em RDF. Abaixo o mesmo exemplo acima, porém com o URI abreviado. Note que é preciso definir um URI base (`xml:base`) para que ao ser processado, a máquina consiga montar o URI completo do recurso, concatenando `xml:base` com `rdf:about`.

```
xml:base="http://www.exemplo.com"
<owl:Class rdf:about="#Pessoa"/>
```

É possível criar hierarquias de classes utilizando uma propriedade definida no RDFSchema (`rdfs:subClassOf`), onde, com `rdfs:subClassOf`, é possível declarar que uma classe é subclasse de outra. O fragmento abaixo demonstra a criação de uma hierarquia de classes, onde se define que a classe `Cliente` é subclasse de `Pessoa`.

```
<owl:Class rdf:about="#Cliente">
     <rdfs:subClassOf rdf:resource="#Pessoa"/>
</owl:Class>
```

Quando é necessário expressar que uma classe é equivalente a outra, ou seja, que os indivíduos pertencentes a uma dada classe também pertencem a uma outra, é possível utilizar a propriedade `owl:equivalentClass`. Já para definir que os indivíduos de uma classe nunca serão indivíduos de outra, se utiliza a propriedade `owl:disjointWith`. O fragmento abaixo exemplifica os dois casos, onde se define que a classe `SerHumano` é equivalente a classe `Pessoa` e que a classe `Cliente` é disjunta da classe `Funcionario`.

```
<owl:Class rdf:about="#SerHumano">
     <owl:equivalentClass rdf:resource="#Pessoa"/>
</owl:Class>

<owl:Class rdf:about="#Cliente">
     <owl:disjointWith rdf:resource="#Funcionario"/>
</owl:Class>
```

Uma classe também pode ser definida por um conjunto determinado de indivíduos enumerados. Esse tipo de declaração é denominado de classe enumerada e é utilizada a propriedade `owl:oneOf` para defini-la. Para exemplificar, o fragmento abaixo demonstra a criação da classe `DiasDaSemana` que é composta por uma coleção de indivíduos correspondentes aos dias da semana (Domingo, Segunda, Terça, Quarta, Quinta, Sexta, Sábado).

```
<owl:Class rdf:about="#DiasDaSemana">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Domingo"/>
    <owl:Thing rdf:about="#Segunda"/>
    <owl:Thing rdf:about="#Terca"/>
    <owl:Thing rdf:about="#Quarta"/>
    <owl:Thing rdf:about="#Quinta"/>
    <owl:Thing rdf:about="#Sexta"/>
    <owl:Thing rdf:about="#Sabado"/>
  </owl:oneOf>
</owl:Class>
```

Existe ainda a possibilidade de definir classes a partir de outras classes. Por exemplo, é possível definir uma classe que é a união de outras classes, através da propriedade `owl:unionOf`, ou uma classe que é a intercessão de outras classes, com o uso da `owl:intersectionOf`, ou ainda, definir que uma classe é o complemento de outra, utilizando a `owl:complementOf`.

Propriedades:

As propriedades em OWL estão divididas em duas categorias: Propriedades de Objetos (`owl:ObjectProperty`) e Propriedades de Dados (`owl:DataProperty`).

Datatype Properties são utilizadas para expressar atributos simples de um recurso, pois relacionam um recurso com um dado literal RDF ou um tipo de dado já predefinido pelo XML Schema[31], como decimal (`xsd:decimal`), string (`xsd:string`), data (`xsd:date`), entre outros.

Já as *Object Properties* descrevem um relacionamento entre recursos, como por exemplo, um relacionamento entre duas pessoas, onde uma pessoa trabalha com outra pessoa.

Como dito anteriormente, ao criar uma propriedade, podemos restringir seu uso para determinadas classes. Por exemplo, a propriedade `trabalhaCom` não

[31] XML Schema Data Types - http://www.w3.org/TR/xmlschema11-2/

poder ser aplicada a indivíduos da classe `Cadeira`, pois não faz sentido dizer que uma cadeira trabalha com outra. Assim, cada propriedade possui um domínio e um contradomínio, onde o domínio determina às quais classes essa propriedade se aplica e o contradomínio determina quais classes ou tipos de dados podem ser valores dessa propriedade. Para prover essa restrição a linguagem OWL utiliza as propriedades definidas pelo RDFS: `rdfs:domain` e `rdfs:range`, respectivamente.

As propriedades podem ser dispostas em hierarquias, ou seja, é possível criar uma propriedade e defini-la como sub propriedade (`rdfs:subPropertyOf`) de outra. Isso implica que: se um recurso A está relacionado pela propriedade P com um recurso B e esta propriedade P é sub propriedade de Q, então o recurso A também está relacionado pela propriedade Q com o recurso B. Porém, se A está relacionado pela propriedade Q com B, não implica que A está relacionado pela propriedade P com B:

$$P\ (A, B) \rightarrow Q\ (A, B)$$

Ainda é possível expressar que uma propriedade é equivalente (`owl:equivalentProprerty`) a outra, ou seja, se um recurso A está relacionado pela propriedade P com um recurso B e esta propriedade P é equivalente a propriedade Q, então o recurso A está relacionado pela propriedade Q com o recurso B. E se A está relacionado pela propriedade Q com B, implica que A está relacionado pela propriedade P com B.

$$P\ (A, B) \Leftrightarrow Q\ (A, B)$$

O fragmento abaixo demonstra a criação de duas *Object Properties* (`trabalhaCom` e `coordenadoPor`), onde a propriedade `coordenadoPor` é sub propriedade de `trabalhaCom`.

```
<owl:ObjectProperty rdf:about="#trabalhaCom"/>

<owl:ObjectProperty rdf:about="#coordenadoPor">
     <rdfs:subPropertyOf rdf:resource="#trabalhaCom"/>
</owl:ObjectProperty>
```

A linguagem OWL ainda provê mecanismos que permitem caracterizar melhor as propriedades, o que possibilita um maior poder de inferência para a máquina. São eles:

- Funcional (`owl:FunctionalProperty`) – Uma propriedade funcional indica que o valor (contradomínio) da propriedade é único, ou seja, se o recurso A está relacionado pela propriedade P com o recurso B e o mesmo recurso A está relacionado pela propriedade P com o recurso C, sendo P é uma propriedade funcional, isso implica que os recurso B e C são o mesmo recurso (estão relacionados pela propriedade `owl:sameAs`).

$$P(A, B) \text{ e } P(A, C) \rightarrow B = C$$

- Inversa Funcional (`owl:InverseFunctionalProperty`) – Uma propriedade inversa funcional indica que o sujeito da propriedade (domínio) é único. Sendo assim, se o recurso A está relacionado pela propriedade P com o recurso B e o recurso C está relacionado pela propriedade P com o recurso B, então, sendo P uma propriedade inversa funcional, os recursos A e C são o mesmo recurso (estão relacionados pela propriedade `owl:sameAs`).

$$P(A, B) \text{ e } P(C, B) \rightarrow A = C$$

- Transitiva (`owl:TransitiveProperty`) – Uma propriedade transitiva indica que uma sucessão de relacionamentos é reduzida em um único relacionamento. Por exemplo, para uma propriedade transitiva P, se um recurso A está relacionado pela propriedade P com o recurso B e o recurso B está relacionado pela propriedade P com o recurso C, então o recurso A está relacionado pela propriedade P com o recurso C.

$$P(A, B) \text{ e } P(B, C) \rightarrow P(A, C)$$

- Simétrica (`owl:SymmetricProperty`) – Uma propriedade simétrica indica um relacionamento onde o sujeito e o objeto da tripla RDF ocupam as duas posições necessariamente, ou seja, sendo P uma propriedade simétrica, se o recurso A está relacionado pela propriedade P com o recurso B, então o recurso B está relacionado pela propriedade P com o recurso A.

$$P(A, B) \Leftrightarrow P(B, A)$$

- Assimétrica (`owl:AsymmetricProperty`) – Uma propriedade assimétrica indica que o sujeito e o objeto da tripla RDF não podem ocupar as duas posições simultaneamente. Por exemplo, se o recurso A está relacionado pela propriedade P com o recurso B, implica que B não pode estar relacionado pela propriedade P com o recurso A.

$$P(A, B) \rightarrow \sim P(B, A)$$

- Reflexiva (`owl:ReflexiveProperty`) – Uma propriedade reflexiva relaciona um recurso consigo mesmo, ou seja, se o recurso A está relacionado pela propriedade P com o recurso B e P é uma propriedade reflexiva, então A está relacionada pela propriedade P consigo mesma.

$$P(A, B) \rightarrow P(A, A)$$

- Irreflexiva (`owl:IrreflexiveProperty`) – Uma propriedade irreflexiva indica que um recurso não pode estar relacionado com ele mesmo por meio dela. Por exemplo, se a propriedade P é irreflexiva, um recurso A não pode estar relacionado pela propriedade P com ele mesmo.

$$\sim P(A, A)$$

Ainda é possível definir que uma propriedade é inversa de outra. Isso permite explicitar que se o recurso A está relacionado pela propriedade P com o recurso B

e a propriedade P é inversa da propriedade Q, isso implica que o recurso B está relacionado pela propriedade Q com o recurso A, e vice-versa.

$$P(A, B) \Leftrightarrow Q(B, A)$$

Por fim, OWL permite descrever propriedades que são disjuntas, ou seja, propriedades que não podem relacionar os mesmos indivíduos ao mesmo tempo. Por exemplo, se o recurso A está relacionado pela propriedade P com o recurso B e P é disjunta da propriedade Q, então o recurso A não pode estar relacionado pela propriedade Q com o recurso B.

$$P(A, B) \rightarrow \sim Q(A, B)$$

Todas as características apresentadas são aplicadas às *Object Properties*, com exceção da característica funcional, que pode ser aplicada também às *Datatype Properties*.

Restrições de propriedade em OWL:

As restrições de propriedade (*property restriction*) permite a definição de classes a partir de restrições (axiomas) aplicadas sobre propriedades, de modo que para um indivíduo pertencer a essa classe, têm que obedecer às restrições estabelecidas.

Todas as restrições de propriedade descrevem uma classe anônima, na qual todos os seus indivíduos devem obedecer às restrições de propriedade que definem essa classe. Ao se descrever uma classe por meio de uma restrição de propriedade, o que se faz de fato, é definir uma superclasse anônima daquela classe.

Existem dois tipos de restrições de propriedade: as restrições de valor e as restrições de cardinalidade.

As restrições de valor são:

- Restrição existencial (`owl:someValuesFrom`) – essa restrição especifica que indivíduos da classe que está sendo descrita possuem pelo menos um relacionamento com indivíduos da classe destino pela referida

propriedade. Por exemplo, no fragmento abaixo, a classe `Funcionario` é subclasse de uma classe anônima que contém indivíduos que possuem pelo menos um relacionamento com indivíduos da classe `Empresa` através da propriedade `trabalhaEm`. É importante ressaltar que um indivíduo que pertença a classe `Funcionário`, além de um relacionamento com a classe `Empresa`, pode possuir um relacionamento com outra classe, através da mesma propriedade `trabalhaEm`.

```
<owl:Class rdf:about="#Funcionario">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#trabalhaEm"/>
         <owl:someValuesFrom rdf:resource="#Empresa"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

- Restrição universal (`owl:allValuesFrom`) – essa restrição define que os indivíduos da classe que está sendo descrita possuem relacionamentos com somente indivíduos de uma dada classe por meio da propriedade em questão. É possível, porém, que um indivíduo que não tenha qualquer relacionamento utilizando a referida propriedade, pertença, mesmo assim, a classe descrita por esta restrição (satisfação trivial). O fragmento abaixo demonstra a definição da classe `Funcionario` como subclasse de uma classe anônima que contém indivíduos que, através da propriedade `trabalhaEm`, possuem relacionamentos apenas com indivíduos da classe `Empresa`.

```
<owl:Class rdf:about="#Funcionario">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#trabalhaEm"/>
         <owl:allValuesFrom rdf:resource="#Empresa"/>
```

```
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

- Restrição de indivíduo (`owl:hasValue`) – essa restrição especifica que os indivíduos da classe que está sendo descrita possuem relacionamentos com um determinado indivíduo através da referida propriedade. O fragmento abaixo demostra a definição da classe `Funcionario` como subclasse de uma classe anônima que contém indivíduos que possuem um relacionamento, através da propriedade `trabalhaEm`, com o indivíduo `InstitutoFederalFluminense`.

```
<owl:Class rdf:about="#Funcionario">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#trabalhaEm"/>
         <owl:hasValue
rdf:resource="#InstitutoFederalFluminense"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Já as restrições de cardinalidade definem um número mínimo, máximo ou exato de relacionamentos que um indivíduo pode ter ao pertencer a classe que está sendo descrita.

- Cardinalidade mínima (`owl:minQualifiedCardinality`) – define que um indivíduo que pertence a classe que está sendo descrita, necessariamente tem uma quantidade mínima de relacionamentos, por meio da referida propriedade, com indivíduos de uma dada classe.
- Cardinalidade máxima (`owl:maxQualifiedCardinality`) – define que um indivíduo que pertence a classe que está sendo descrita, necessariamente tem uma quantidade máxima de relacionamentos, por meio da referida propriedade, com indivíduos de uma dada classe.

- Cardinalidade exata (`owl:qualifiedCardinality`) – define que um indivíduo que pertence a classe que está sendo descrita, necessariamente tem uma quantidade exata de relacionamentos, por meio da referida propriedade, com indivíduos de uma dada classe.

O fragmento abaixo demonstra um exemplo da classe `Carro` como subclasse de uma classe anônima que contém indivíduos que possuem: (i) no mínimo dois relacionamentos com indivíduos da classe `Porta` utilizando a propriedade `temPorta`; (ii) no máximo um relacionamento com indivíduos da classe `Estepe` utilizando a propriedade `temEstepe`; (iii) exatamente quatro relacionamentos com indivíduos da classe `Roda` utilizando a propriedade `temRoda`.

```
<owl:Class rdf:about="#Carro">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#temRoda"/>
         <owl:onClass rdf:resource="#Roda"/>
         <owl:qualifiedCardinality
          rdf:datatype="&xsd;nonNegativeInteger">4</owl:
          qualifiedCardinality>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#temPorta"/>
         <owl:onClass rdf:resource="#Porta"/>
         <owl:minQualifiedCardinality
          rdf:datatype="&xsd;nonNegativeInteger">2</owl:
          minQualifiedCardinality>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#temEstepe"/>
```

```
            <owl:onClass rdf:resource="#Estepe"/>
            <owl:maxQualifiedCardinality
             rdf:datatype="&xsd;nonNegativeInteger">1</owl:
             maxQualifiedCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Cadeia de propriedades:

Semelhante às propriedades transitivas, onde uma cadeia de relacionamentos é reduzida em um único relacionamento, uma cadeia de propriedades (`owl:propertyChainAxiom`) permite deduzir, em uma sucessão de relacionamentos (de propriedades iguais ou diferentes), um outro relacionamento. Por exemplo, um relacionamento de `ehIrmao` pode ser deduzido da seguinte maneira: se o recurso A está relacionado pela propriedade `ehPai` com o recurso B e o mesmo recurso A está relacionado pela mesma propriedade com o recurso C, então o recurso B está relacionado pela propriedade `ehIrmao` com o recurso C.

Essa sequência de relacionamentos pode ter vários relacionamentos e fazer uso de variadas propriedades.

ehPai (A, B) e ehPai (A, C) → ehIrmao (B, C)

Elemento chave de uma classe:

É possível assinalar, na descrição de uma classe, que uma propriedade possui a característica de relacionar o indivíduo dessa referida classe com outro indivíduo ou valor que o identifique unicamente. Um exemplo seria o valor do ISBN para um livro, ou ainda o CPF para uma pessoa.

O fragmento abaixo demostra a criação da classe `Pessoa` onde se descreve a propriedade `temCPF` como chave para esta classe.

```
<owl:Class rdf:about="#Pessoa">
        <owl:hasKey rdf:parseType="Collection">
```

```
            <rdf:Description rdf:about="#temCPF"/>
        </owl:hasKey>
</owl:Class>
```

Os principais recursos que a linguagem OWL disponibiliza para a construção de ontologias foram expostos nessa seção, porém ainda existem outros recursos, como por exemplo as propriedades de anotação, que permitem descrever os próprios elementos criados (classes e propriedades), além de poder criar expressões mais complexas ao se utilizar operadores lógicos como *AND*, *OR* e *NOT*.

Devido a essa gama de recursos e possibilidades para descrever ontologias, a linguagem OWL foi utilizada para projetar a ontologia proposta nesta dissertação.

2.2.6 Metodologias de construção de ontologias

O processo de construção de ontologias ainda não possui uma estrutura bem definida. A grande maioria das ontologias existentes utilizam processos próprios, focados em suas necessidades e determinados pela experiência de seus desenvolvedores.

Noy e McGuinness (2001) afirmam que “não existe um único modo correto para modelar um domínio – existem alternativas viáveis. A melhor solução quase sempre depende da aplicação que se tem em mente”. Além disso, a construção de uma ontologia é um processo orgânico, onde, a medida que está vai sendo utilizada, novas características inerentes ao domínio representado vão sendo identificadas e incorporadas, ou seja, o desenvolvimento de ontologias é um processo contínuo de aperfeiçoamento.

Contudo, apesar de ainda não existir metodologias totalmente maduras, algumas metodologias foram estudadas de modo a servir como base para o desenvolvimento dessa dissertação. Uma breve descrição dessas metodologias é dada a seguir:

- ***Ontology Development 101*:** A metodologia *Ontology Development 101* foi proposta por Noy e McGuinness (2001), a partir da experiência no desenvolvimento de uma ontologia abordando o domínio de vinhos e

alimentos, com o uso do ambiente de desenvolvimento de ontologias *Protégé*[32]. Essa metodologia propõe basicamente sete atividades para o desenvolvimento de uma ontologia, a saber: i) Determine o domínio e o escopo da ontologia; ii) Considere o reuso de ontologias existentes; iii) Enumere os termos importantes da ontologia; iv) Defina as classes e a hierarquia de classes; v) Defina as propriedades das classes; vi) Defina as restrições; vii) Crie instâncias. Tais atividades se apresentam dentro de um ciclo iterativo e incremental.

- ***Methontology:*** A metodologia *Methontology* foi desenvolvida por um grupo de pesquisadores ao desenvolverem uma ontologia dentro do domínio da química (FERNÁNDEZ-LÓPEZ; GÓMEZ-PÉREZ; JURISTO, 1997; GÓMEZ-PÉREZ; FERNÁNDEZ; VICENTE, 1996). A *Methontology* propõe o desenvolvimento em seis atividades: i) Especificação; ii) Conceitualização; iii) Formalização; iv) Integração; v) Implementação; vi) Manutenção. Antes de se iniciar essas atividades, a metodologia propõe que seja feito um planejamento de como essas atividades serão realizadas. Além disso, durante a execução das atividades acima enumeradas, *Methontology* propõe algumas atividades paralelas visando apoiar todo o processo de construção da ontologia. São elas: i) Aquisição de conhecimento (ontologias existentes, por exemplo); ii) Documentação; iii) Avaliação.
- **Metodologia de Grüninger e Fox:** A metodologia de *Grüninger e Fox* (1995) foi utilizada no *Enterprise Integration Laboratory* da *University of Toronto* durante o desenvolvimento do projeto *Toronto Virtual Enterprise*[33] (Tove). Essa metodologia propõe as seguintes etapas: i) elaboração de cenários de motivação; ii) especificação de questões de competência informal, que objetivam estabelecer requisitos para a ontologia; iii) concepção da terminologia formal, onde os conceitos e suas propriedades são organizados em uma taxonomia; iv) especificação de questões de competência formal, onde os requisitos são estabelecidos de modo coerente diante dos axiomas estabelecidos

[32] *Protégé* – http://protege.stanford.edu/
[33] *Toronto Virtual Enterprise* – http://www.eil.utoronto.ca/enterprise-modelling/tove/

na ontologia; v) especificação de axiomas formais, que visam atender às questões de competência formais; vi) Verificação de teoremas completos, onde são verificadas as resposta das questões de competência especificadas nas etapas anteriores.

- ***Enterprise Ontology:*** Proposta por Uschold e King (1995), essa metodologia é composta por quatro etapas, a saber: i) identificação do propósito da ontologia, que visa identificar a necessidade, grau de formalismo e tipos de usuários da ontologia; ii) construção da ontologia, onde a ontologia será conceitualizada, codificada e integrada com ontologias existentes; iii) avaliação da ontologia; iv) documentação da ontologia.

A metodologia escolhida para ser utilizada no desenvolvimento deste trabalho foi a *Ontology Development 101* e será descrita com mais detalhes em seção própria.

2.2.7 Raciocinadores

Como exposto anteriormente, ontologias OWL possuem classes, hierarquia de classes, propriedades, diversas características de propriedades e axiomas. Todo esse ferramental permite a construção de ontologias ricas em informações semânticas e lógica descritiva. Assim, um raciocinador pode ajudar tanto na construção da ontologia quanto na descoberta de novos conhecimentos quando em uma base de dados semânticos. Um raciocinador semântico, ou somente raciocinador (*reasoner*) é um software capaz de inferir consequências lógicas a partir de um conjunto de fatos ou axiomas descritos e muitos utilizam lógica descritiva de primeira ordem para inferir novos conhecimentos (ABBURU, 2012).

Vários racionadores foram desenvolvidos, tais como: *FaCT*++[34], *HermiT*[35], *Pellet*[36] e *Racer*[37]. Especificamente para auxiliar neste trabalho, foi utilizado o raciocinador *Pellet*.

[34] *FaCT*++ – h ttp://owl.man.ac.uk/factplusplus/
[35] *HermiT* – http://hermit-reasoner.com/
[36] *Pellet* – http://clarkparsia.com/pellet/
[37] *Racer* – https://www.ifis.uni-luebeck.de/index.php?id=385

O raciocinador *Pellet* é um software livre escrito na linguagem Java baseado OWL-DL, desenvolvido pelo grupo de pesquisa *MINDSWAP*. O *Pellet* foi desenvolvido para trabalhar com OWL-DL, abordando todas as suas funcionalidades semânticas, oferecendo raciocínio sobre tipos de dados e instâncias, além de outras funcionalidades, como a classificação automática de classes e propriedades, e verificação de inconsistências na ontologia (PARSIA; SIRIN, 2004).

2.3 SINTOMAS, PROBLEMAS E SOLUÇÕES

A ontologia desenvolvida nessa dissertação tem como objetivo representar o domínio de sintomas, problemas e soluções. Por ser uma *core ontology*, ou seja, uma ontologia que pretende prover uma generalização para diversos domínios de problemas, há que se compreender como são percebidos esses elementos (sintomas, problemas e soluções) em diversos contextos.

Segundo o dicionário Michaelis (2009), um problema é uma questão levantada para inquirição, consideração, discussão, decisão ou solução. Acrescenta ainda que um problema é um tema cuja solução ou decisão requer considerável meditação ou habilidade.

A definição de problema está intimamente relacionada com o contexto em que se aplica, assim, no contexto da administração, por exemplo, um problema é indicado por alguma frustração, irritação, percepção de diferença entre a situação ideal e a real e perspectivas de prejuízos. Entende-se que um problema gera sempre uma decisão, ou seja, a partir da definição do problema, há que se buscar alternativas, ou soluções, para solucioná-lo (MAXIMIANO, 2004). Como exemplo, pode considerar um problema na fabricação de um produto, onde um sinal do problema pode ser a percepção de que o produto final não corresponde ao esperado (projetado). Com a identificação do problema, é aplicada a solução que pode ser a correta parametrização dos equipamentos evolvidos na fabricação do produto.

Ao se analisar o contexto de Tecnologia da Informação (TI), a especificação ITIL v3 (CARTLIDGE et al., 2007) define um problema como uma causa não conhecida de um ou mais incidentes, ou seja, um incidente sinaliza um ou mais

problemas. Um incidente pode ser considerado, dentro deste contexto, como uma interrupção não planejada ou redução na qualidade de um serviço de TI. Assim, com base nos incidentes (sinais da ocorrência de um problema) é identificado o problema que é a causa dos incidentes. Após a identificação do problema, é aplicada uma solução, geralmente em forma de procedimentos (*workflow*). Por exemplo, considerando uma imperfeição de impressão, como sinal da ocorrência do problema, um usuário pode relatar que seus documentos estão apresentando falhas no texto ao serem impressos. Como solução, o técnico de TI pode efetuar a troca do cartucho de tinta, efetuar alinhamento e limpeza das cabeças de impressão.

Já na medicina, essa distinção fica bem clara, onde os problemas são identificados como doenças. Assim, um paciente reporta alguns sintomas, ou sinais, que indicam a existência de uma doença. O médico, a partir dos sintomas, identifica a doença que está causando os sintomas. Já diagnosticado a doença, o paciente é submetido ao tratamento adequado. Um tratamento pode ser representado por uma sequência de passos como um *workflow*. Por exemplo, um paciente chega ao consultório de um médico reportando que está com alguns sintomas, como febre e coriza, o médico realiza o seu diagnóstico identificando a doença, que pode ser uma gripe. Sabendo da gripe, o médico indica um tratamento que o paciente deve seguir (MAYO CLINIC STAFF, 2014).

Com base nas abordagens de problemas nos contextos citados acima, conclui-se que um problema é algo ou alguém que não está no seu estado normal ou ideal. Quando algo ou alguém não se apresenta em seu estado ideal, sinais ou sintomas podem ocorrer evidenciando a existência do problema. Além disso, após a descoberta da causa dos sintomas, ou seja, a descoberta do problema em si, inicia-se o procedimento para solucionar o mesmo, onde, caso já exista uma solução previamente conhecida e catalogada, basta aplicá-la. Contudo, caso o problema seja desconhecido, é necessário estudar o problema visando o entendimento do mesmo para, posteriormente, elaborar e testar uma ou mais soluções. Um processo muitas vezes demorado e oneroso, cuja solução encontrada, obviamente, deve ser cuidadosamente catalogada para outras eventuais ocorrências futuras. A Figura 15 ilustra essa concepção.

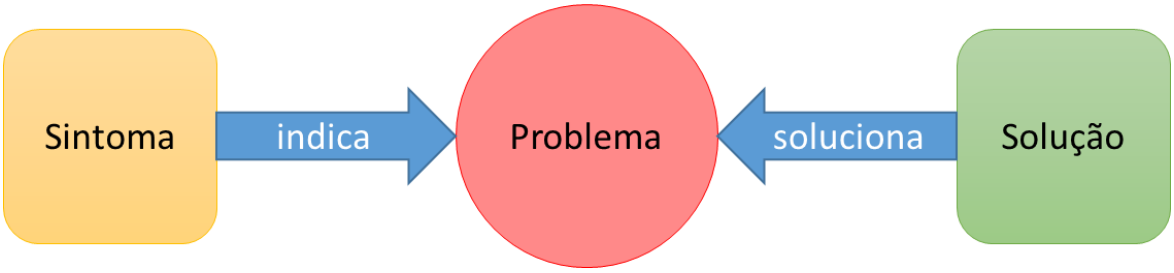

Figura 15 - Sintoma, problema e solução.
Fonte: O autor.

Para que outras pessoas possam aplicar uma solução ao problema, este deve ser apresentado de modo estruturado e não ambíguo. Assim, uma forma muito utilizada é a estruturação das ações a serem tomadas em uma sequência cuidadosamente organizada de passos, conhecida pelo termo *workflow*. Um *workflow* permite não somente expressar passos de forma linear sequencial, mas também criar construções mais complexas como estruturas condicionais e de repetição.

3 *ISSUE PROCEDURE ONTOLOGY* (IPO)

3.1 PROCEDIMENTOS METODOLÓGICOS

Para nortear o processo de desenvolvimento da ontologia proposta - *Issue Procedure Ontology* (IPO), foi empregada a metodologia *Ontology Development 101*. Para tal, fora realizada uma pesquisa bibliográfica que permitiu um melhor entendimento envolvendo os conceitos fundamentais acerca do tema, na qual foram priorizados os autores seminais e formuladores das teorias, citados na fundamentação teórica, embora outros estejam presentes.

A metodologia *Ontology Development 101*, proposta por Noy e McGuinness (2001), propõe um processo de construção iterativo e incremental de sete etapas, ilustrado na Figura 16.

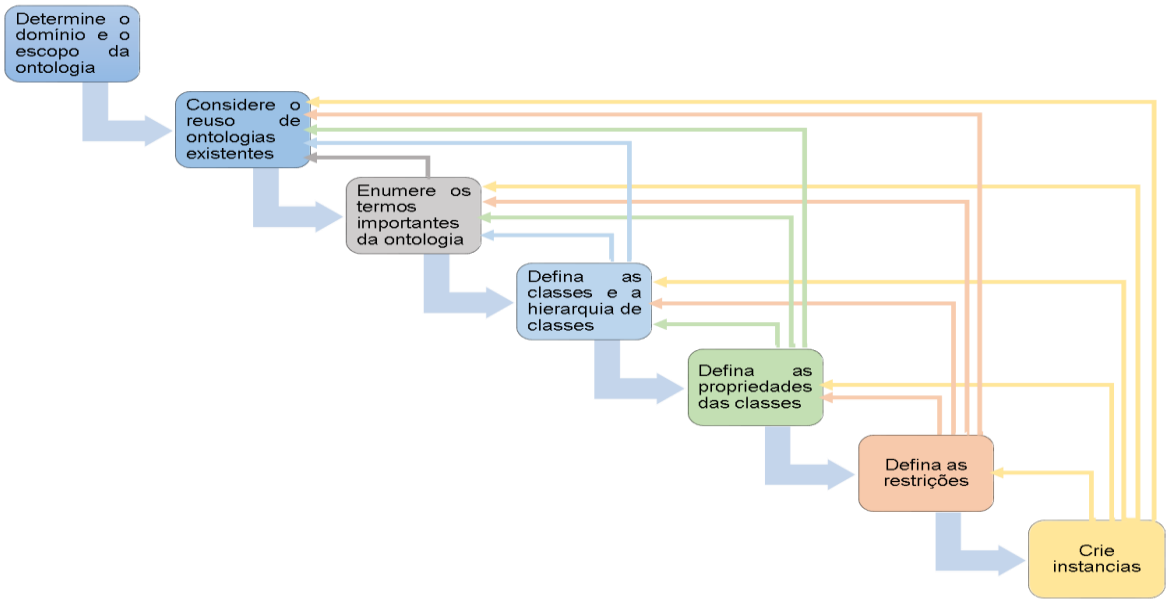

Figura 16 - Processo de desenvolvimento da *Ontology Development 101*.
Fonte: O autor.

1. *Determine o domínio e o escopo da ontologia*: antes de se iniciar a construção de uma ontologia, deve se estabelecer os limites do conhecimento que esta irá representar. Primeiramente, é determinado o domínio, ou seja, qual área do conhecimento a ontologia irá abordar. É importante também delimitar essa área representada, o que se denomina escopo. O escopo estabelece o que se pretende representar dentro do domínio para que a ontologia não tenha a pretensão de representar um campo muito vasto, complicando sua construção. Para facilitar a determinação do escopo, algumas questões de competência podem ser utilizadas, são elas: (i) Qual o domínio que a ontologia vai representar? (ii) Para que nós vamos usar a ontologia? (iii) Para quais tipos de questão a ontologia precisa prover respostas? (iv) Quem vai usar ou manter a ontologia?
2. *Considere o reuso de ontologias existentes*: após a determinação do domínio e a delimitação do escopo, é importante verificar o reuso de ontologias existentes. É possível que já exista uma ontologia que já represente o domínio desejado, assim, não se faz necessário o desenvolvimento de outra, gerando duplicidade. Além disso, ao se reutilizar ontologias, permite uma melhor interoperabilidade entre os diversos sistemas semânticos que fazem uso da ontologia reusada. Outra forma de reuso seria estender uma ontologia existente, ou seja, a ontologia não atende completamente o domínio desejado e sim parte dele, dessa forma, pode-se estender ou especializar essa ontologia para se adequar ao domínio que se deseja representar.
3. *Enumere os termos importantes da ontologia*: é bastante útil na construção de uma ontologia, elaborar uma lista de todos os termos que se deseja contemplar na ontologia, ou seja, termos que se gostaria de falar ou propriedades que esses termos possuem. Deve ser elaborada uma lista abrangente, sem se preocupar com sobreposição entre conceitos, relações entre termos, ou quaisquer propriedades que os conceitos possam ter.
4. *Defina as classes e a hierarquia de classes*: a partir da lista elaborada na etapa anterior, são identificados os termos que classificam os objetos

(indivíduos), ou seja, que são categorias dos objetos. Por exemplo, dentro do domínio de TI, o termo *Computador* pode ser uma classe, pois define uma categoria de objetos. Esses termos serão classes na ontologia e serão organizados em uma taxonomia, onde existe um relacionamento de "é um" entre instâncias da subclasse com a superclasse, ou seja, toda instância da subclasse é uma instância da superclasse. A Figura 17 apresenta um exemplo de hierarquia de classes extraída de FOAF (BRICKLEY; MILLER, 2014) - ontologia voltada para descrição de pessoas e seus relacionamentos com objetos e outras pessoas.

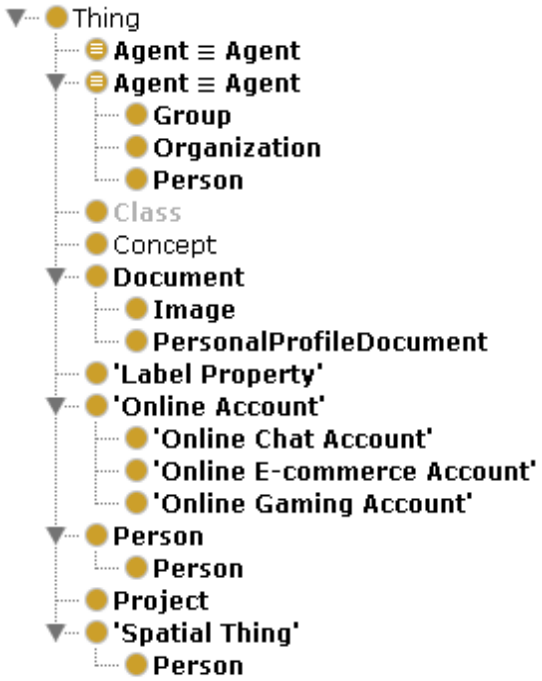

Figura 17 - Hierarquia de classes da ontologia FOAF (BRICKLEY; MILLER, 2014).
Fonte: O autor

5. *Defina as propriedades das classes*: a classe por si só não provê toda a informação necessária para que seja possível responder as questões de competência. Assim, a partir das classes definidas na etapa anterior, é necessário descrever suas características e relacionamentos com outras classes. Com base na lista elencada na etapa 3, parte dos termos já foram identificados como classes, assim, provavelmente, os termos restantes serão propriedades das classes. Por exemplo, ainda no domínio de TI, para a classe *Computador*, podemos citar como propriedades: *marca*, *cor*, *fabricante*, etc. Desses exemplos, as duas primeiras propriedades

representam propriedades de dados (*Datatype Property*), pois representam atributos da classe, já a última, seria um relacionamento (*Object Property*), pois representa um relacionamento entre classes (classe *Computador* com a classe *Organização*). É importante ressaltar o conceito de herança entre classes e subclasses, pois as subclasses herdam as propriedades de suas superclasses.

6. *Defina as restrições*: As propriedades podem ter diferentes restrições que descrevem o tipo de valor, os valores permitidos, o número dos valores (cardinalidade), entre outros. Por exemplo, o valor da propriedade *marca* (tal como em "a marca de um computador") é uma cadeia de caracteres. Isto é, *marca* é uma propriedade que tem como tipo de valor literal (texto). Outro exemplo seria a propriedade *fabricante* (como em "um computador tem como fabricante uma organização") pode ter vários valores e os valores são instâncias da classe *Organização*. Como já foi detalhado ao se discorrer sobre a linguagem OWL, nessa etapa são adicionadas todas as características OWL para as propriedades e também são especificadas restrições para as classes (*someValues, allValuesFrom*, etc.). A Figura 18 mostra a tela do editor de ontologias *Protégé*, onde é possível descrever as características OWL para as propriedades.
7. *Crie instâncias*: nessa etapa, deve-se identificar classes que possuem instâncias padrão. Por exemplo, classes enumeradas como a classe Dias da Semana, que tem como instâncias: domingo, segunda-feira, ..., sábado. Assim, como já se sabe de antemão as instâncias de determinadas classes, é possível já criá-las para serem reusadas. Além de classes enumeradas, também são criadas instâncias a fim de testar a ontologia e outras instâncias para reuso.

Após o desenvolvimento da ontologia, se faz necessário avaliar se a mesma provê a semântica necessária para atender os requisitos levantados no início de seu desenvolvimento. Para realizar essa avaliação, se faz necessário verificar se a ontologia é expressiva o suficiente para prover respostas para as questões de competência levantadas anteriormente e verificar se ela consegue representar todos os dados pertinentes ao domínio.

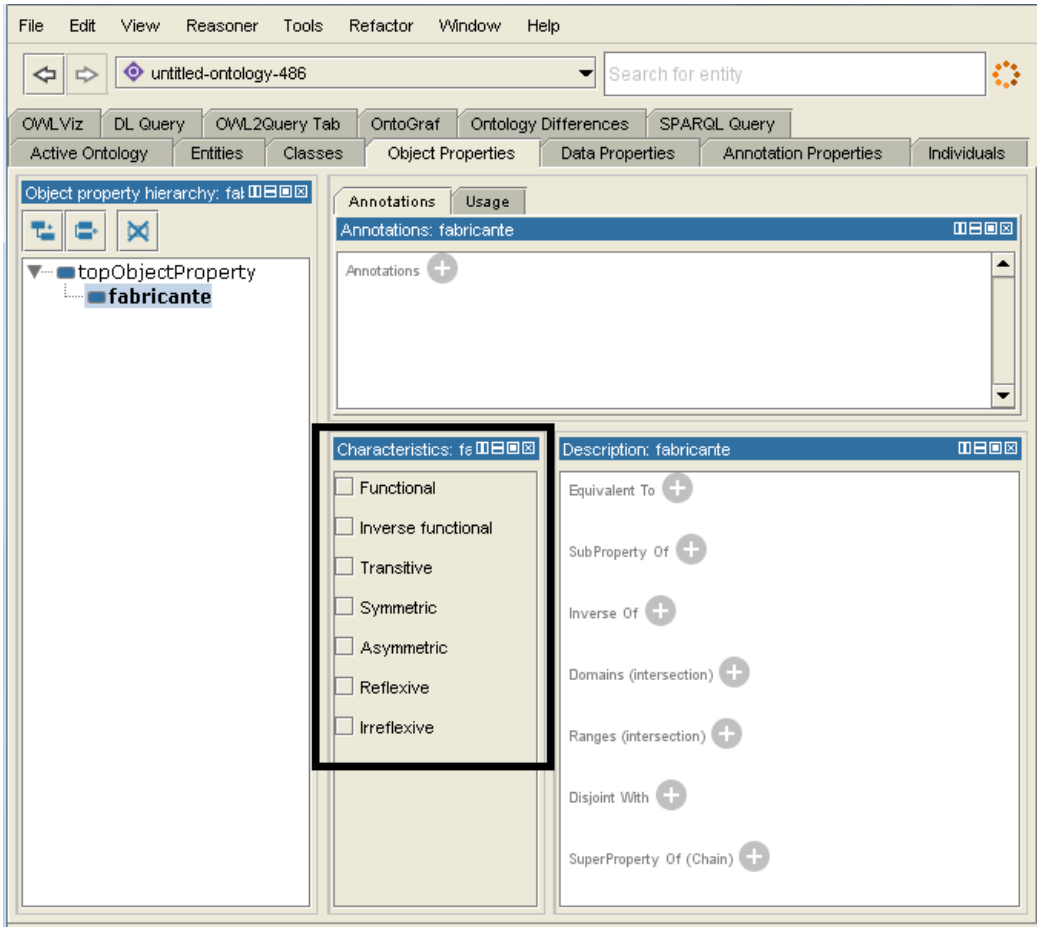

Figura 18 - Tela do *Protégé* para especificação das características OWL.
Fonte: O autor

Além dessa avaliação, nesse trabalho, foi feito um estudo de caso realista demonstrando como especializar a ontologia IPO para o domínio da medicina. Por ser uma *core ontology*, essa especialização para um determinado domínio pode se fazer necessária para que a ontologia se adeque às especificidades do domínio representado, porém, é importante ressaltar que é possível utilizar a IPO sem qualquer especialização. Essa necessidade deve ser avaliada por um especialista de domínio, levando em consideração as peculiaridades e o poder de expressividade esperado pelo sistema semântico a ser construído.

3.2 ESCOPO

A ontologia IPO pretende representar o domínio de problema, relacionando sintomas e soluções. Visando especificar o escopo e o domínio da ontologia, foram considerados alguns cenários motivacionais, dentre os quais, setor de suporte de

TI de uma empresa (*help desk*), diagnóstico de doenças por um médico, identificação de defeitos na linha de produção de uma indústria e problemas no *design* de um software. Após analisá-los, foram elencadas algumas questões de competência para as quais a ontologia IPO deveria prover as respostas. Estas questões servem para definir o escopo da ontologia, bem como avaliar sua expressividade. São elas:

QC1. Qual(is) categoria(s) de um problema/sintoma/solução?

QC2. Quais são sintomas de um problema?

QC3. Quais são as soluções de um problema?

QC4. Qual o causador do problema?

QC5. Qual o hospedeiro do problema?

QC6. Quem criou/registrou esse problema/sintoma/solução?

QC7. Quais ações (*workflow*) a serem tomadas para solucionar o problema?

QC8. Um problema causa outro problema?

QC9. Um problema depende de outro problema?

QC10. Quais os possíveis problemas, a partir de um conjunto de sintomas?

QC11. Qual(is) classe(s) de um determinado problema a partir dos seus sintomas?

A fim de se aplicar a diretriz de reuso de ontologias existentes, foi feito um levantamento das ontologias *Linked Data* relacionadas no indexador *Linked Open Vocabularies* (LOV)[38], onde verificou-se que nenhuma ontologia se propunha a descrever o domínio proposto com a expressividade necessária para responder as questões levantadas acima. No entanto, é possível reusar parte de ontologias existentes, evitando retrabalho e facilitando o mapeamento dos dados entre ontologias.

A seguir é exposta uma breve descrição de ontologias de referência que, de alguma forma, se relacionam com a ontologia IPO.

***Friend of a Friend* (FOAF)[39]:** O vocabulário FOAF foi desenvolvido no ano de 2000 a partir do *FOAF Project* criado por *Dan Brickley* e *Libby Miller*. FOAF é uma ontologia voltada para descrição de pessoas e seus relacionamentos com

[38] LOV – *Linked Open Vocabularies* – http://lov.okfn.org/dataset/lov/
[39] *Friend of a Friend* (FOAF) – http://www.foaf-project.org/

objetos e outras pessoas, usando ideias simples inspiradas na *Web*. FOAF pode ser dividida em três categorias (BRICKLEY; MILLER, 2014):

1. *Core* - essas classes e propriedades formam o núcleo da ontologia FOAF. Eles descrevem características das pessoas e grupos sociais que são independentes do tempo e da tecnologia; como tal, eles podem ser usados para descrever as informações básicas sobre as pessoas no dia a dia, patrimônio histórico, cultural e contextos de bibliotecas digitais.
2. *Web Social* - para além dos termos principais de FOAF, há uma série de termos para descrever contas de Internet, catálogos de endereços e outras atividades baseadas na *Web*.
3. *Linked Data Utilities* - FOAF estabelece diversos termos que são usados para a representação de “coisas” que são úteis para as pessoas na *Web*, porém, mantendo ênfase na ideia central de FOAF, que é interligar redes de informações com redes de pessoas.

FOAF é amplamente utilizada na nuvem LOD, assim, visando uma maior integração entre os dados RDF, a ontologia IPO possui relações de equivalência e especialização com algumas classes e propriedades de FOAF.

***Dublin Core* (DC)[40]:** Em meados de 1990, *Dublin Core* começou com a ideia de se estabelecer metadados para a descrição simples e genérica de recursos eletrônicos. O nome "*Dublin*" é devido a sua origem em um *workshop* em 1995, em *Dublin*, *Ohio*; "*Core*" porque os seus elementos são genéricos, utilizável para descrever uma ampla gama de recursos. Um grupo internacional, interdisciplinar de profissionais de biblioteconomia, ciência da computação, codificação de texto e comunidade de museus e outras áreas afins, estabeleceram quinze elementos que são a base (*core*) para descrever recursos na *Web*[41]. Desde então a ontologia *Dublin Core* vem sendo atualizada em concordância com as novas recomendações do W3C. Atualmente o âmbito de aplicação de metadados *Dublin Core* foi ampliado de "recursos eletrônicos" para abranger, em princípio, qualquer objeto que possa ser identificado, no mundo real, eletrônico ou conceitual (RÜHLE; BAKER; JOHNSTON, 2011).

[40] *Dublin Core* (DC) – http://dublincore.org/documents/dcmi-terms/
[41] Os 15 elementos formam o *Dublin Core Metadata Element Set* – http://dublincore.org/documents/dces/

O vocabulário *Dublin Core* é muito utilizado na nuvem LOD e a IPO especializa algumas propriedades dessa ontologia de referência, como as propriedades *title* e *description*.

***Simple Knowledge Organization System* (SKOS)[42]:** SKOS é um vocabulário RDF para representar sistemas semiformais de organização do conhecimento (KOSs), tais como tesauros, taxonomias, esquemas de classificação, entre outros. SKOS foi projetado para fornecer um caminho de migração de baixo custo para portar sistemas de organização existentes para a *Web* Semântica. SKOS pode ser usado de forma independente ou em combinação com linguagens mais formais, tais como a OWL. SKOS também pode ser visto como uma tecnologia de transição, fornecendo o elo entre o formalismo lógico rigoroso de linguagens ontológicas como OWL e do mundo caótico, informal e fracamente estruturado de ferramentas de colaboração baseadas na *Web*. Assim, o objetivo de SKOS não é substituir vocabulários conceituais originais no seu contexto inicial de utilização, mas lhes permitir serem portados para um espaço comum, com base em um modelo simplificado, permitindo mais ampla reutilização e melhor interoperabilidade (ISAAC; SUMMERS, 2009).

SKOS, através da classe *Concept,* permite criar esquemas de categorização bastante elaborados e a ontologia IPO faz uso da classe *Concept* para possibilitar criar esses esquemas de categorização para suas principais entidades.

3.3 *DESIGN*

Para a ontologia criada neste trabalho, foi adotado o prefixo `ipo` para representar o seu *namespace* `http://purl.org/ipo/core#`. Assim, as classes e propriedades pertencentes a esta ontologia serão precedidas do prefixo `ipo`. Trata-se de uma ontologia OWL 2, cujas especificações técnicas, bem como o código fonte em RDF/XML, de todos os termos e propriedades da ontologia foram publicadas na *Web* e podem ser encontradas no endereço `http://purl.org/ipo/core`.

[42] *Simple Knowledge Organization System* (SKOS) – http://www.w3.org/2004/02/skos/

No Quadro 4 estão relacionadas todas as ontologias reusadas ou utilizadas para descrever a ontologia IPO, em ordem alfabética, relacionando o prefixo, nome e o *namespace*.

Prefixo	Nome	Namespace
dcterms	*Dublin Core - terms*	http://purl.org/dc/terms/
foaf	*Friend of a friend*	http://xmlns.com/foaf/0.1/
owl	*Ontology Web Language*	http://www.w3.org/2002/07/owl#
rdfs	*RDF-Schema*	http://www.w3.org/2000/01/rdf-schema#
skos	*Simple knowledge organization systems*	http://www.w3.org/2004/02/skos/core#
xsd	*XML Schema*	http://www.w3.org/2001/XMLSchema#

Quadro 4 - Ontologias reusadas ou utilizadas na para descrever a ontologia IPO.
Fonte: O autor

O modelo conceitual da *Issue Procedure Ontology* pode ser visto na Figura 19, onde são apresentados seus termos (classes) e seus atributos e relações (propriedades), por meio de um diagrama de classes UML[43].

[43] UML - *Unified Modeling Language* - http://www.uml.org/

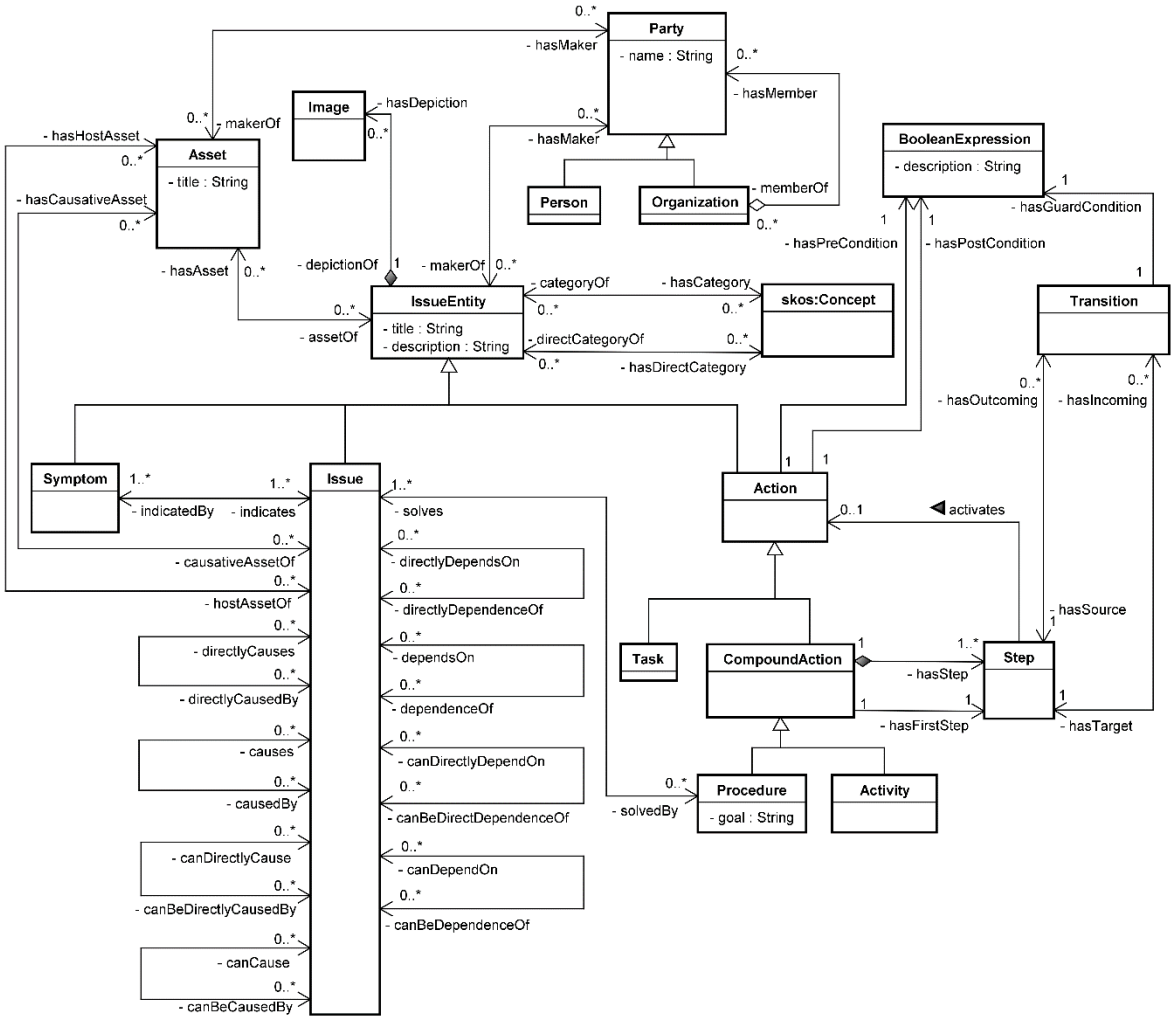

Figura 19 - Modelo conceitual da IPO
Fonte: O autor

A seguir serão descritas as classes (instâncias da metaclasse *owl:Class*) e as propriedades (instâncias das metaclasses *owl:ObjectProperty* ou *owl:DatatypeProperty*) da ontologia IPO. Para tal, seguindo uma abordagem *top-down*[44], a ontologia foi dividida em quatro módulos:

- **IssueEntity - Super Tipo Raiz**

  Classes: *IssueEntity, Asset, Image, skos:Concept*.

  Propriedades: *directCategoryOf, hasDirectCategory, categoryOf, hasCategory, hasMaker, makerOf, hasAsset, assetOf, hasDepiction, depictionOf, title, description*.

[44] A abordagem *top-down* indica que será analisado primeiro as classes mais superiores da hierarquia, ou seja, a classes mais genéricas, e posteriormente serão analisadas as classes mais específicas. Esse é termo é bastante utilizado no desenvolvimento de ontologias para se determinar o ponto de partida para se elencar as classes da ontologia. De acordo com Noy e McGuinness (2001), existem 3 tipos de abordagem: *top-down, bottom-up* e *combination*.

- **Pessoas e Organizações**
  Classes: *Party, Person e Organization.*
  Propriedades: *hasMember, memberOf, name*.

- **Problemas e Sintomas**
  Classes: *Symptom, Issue*.
  Propriedades: *directlyCauses, directlyCausedBy, causes, causedBy, dependenceOf, dependsOn, directDependenceOf, directlyDependsOn, canDirectlyCause, canBeDirectlyCausedBy, canCause, canBeCausedBy, canBeDependenceOf, canDependOn, canBeDirectDependenceOf, canDirectlyDependOn, indicatedBy, indicates, hasCausativeAsset, causativeAssetOf, hasHostAsset, hostAssetOf*.

- **Problemas e Soluções**
  Classes: *Action, Task, CompoundAction, Procedure, Activity, Step, Transition, BooleanExpression*.
  Propriedades: *solves, solvedBy, hasFirstStep, hasStep, activates, hasSource, hasTarget, hasIncoming, hasOutcoming, hasGuardCondition, hasPostCondition, hasPreCondition, goal*.

3.3.1 IssueEntity - Super Tipo Raiz

As classes e propriedades deste módulo visam descrever diversos relacionamentos que são comuns aos três principais termos da ontologia: Sintoma, Problema e Ação (Solução).

Classe: *IssueEntity*

Superconceito (supertipo) que reúne as características comuns aos três principais conceitos dentro do domínio da ontologia: Sintoma (*Symptom*), Problema (*Issue*) e Ação (*Action*).

Sobre a classe *IssueEntity*:

URI:	ipo:IssueEntity
type:	owl:Class
subClassOf:	ipo:description some[45] xsd:string, ipo:title some xsd:string
disjointClass:	ipo:Step, ipo:Image, ipo:Transition, ipo:BooleanExpression, ipo:Party, skos:Concept.

Classe: *Asset*

Qualquer "coisa" de valor relacionada a uma *IssueEntity*[46]. Por exemplo, um problema (doença) diagnosticado em um paciente, pode ter um vírus e o paciente como *Asset*, pois o vírus é o agente causador do problema e o paciente é o hospedeiro no qual o problema se manifesta.

Um *Asset* pode ser uma pessoa, um objeto, um relatório, um documento, etc.

Sobre a classe *Asset*:

URI:	ipo:Asset
type:	owl:Class
subClassOf:	ipo:title some xsd:string

Classe: *Image*

Um artefato que ilustra ou registra uma percepção visual.

Ela pode ser usada para ilustrar uma *IssueEntity* visando uma melhor compreensão dela.

Sobre a classe *Image*:

URI:	ipo:Image
type:	owl:Class
equivalentClass:	foaf:Image

[45] some significa restrição existencial (owl:someValuesFrom).
[46] Como a classe *IssueEntity* já foi descrita anteriormente, será utilizada o próprio nome da classe, em inglês, para citá-la no texto. Para os elementos ainda não descritos, serão citados em português e entre parênteses o nome da classe em inglês.

disjointClass:	`ipo:Step, ipo:IssueEntity, ipo:Transition, ipo:BooleanExpression, ipo:Party, skos:Concept`

Classe: *skos:Concept*

A ontologia IPO reusa a ontologia SKOS para definir esquemas de classificação, ou seja, um conjunto de categorias (ou conceitos) relacionadas (hierarquicamente e de outras formas) formando um tesauro, sob as quais instâncias da classe *IssueEntity* podem ser agrupadas.

A classe *Concept* da ontologia SKOS possui propriedades que permitem a criação de hierarquias de categorias (*Concepts*), permitindo ainda expressar transitividade entre as categorias. Essa abordagem deve ser utilizada como uma alternativa para classificação por subclasse de *IssueEntity*, quando se tratar de classificações não intrínsecas, apenas de agrupamento. Por exemplo, no domínio da medicina, podemos agrupar as doenças como doenças virais, doenças bacterianas, etc.

É importante ressaltar que existe outra forma de classificação, por meio da criação de subclasses de *IssueEntity*. O uso de subclasses leva a uma capacidade de inferência mais refinada e deve ser utilizado quando se tratar de classificação intrínseca de tipo/subtipo, onde novas classes, com novas restrições, precisam ser criadas para descrição de tipos de problema mais específicos de um contexto particular.

Sobre a classe *skos:Concept*:

URI:	`skos:Concept`
type:	`owl:Class`
disjointClass:	`ipo:Step, ipo:Party, ipo:BooleanExpression, ipo:Image, ipo:Transition, ipo:IssueEntity`

Propriedades: *hasCategory e categoryOf*

Uma *IssueEntity* pode ser agrupada em várias categorias, utilizando um esquema de classificação por meio de um tesauro de categorias (ou conceitos). A propriedade *hasCategory* relaciona uma *IssueEntity* com sua(s) categoria(s) (*skos:Concept*). *categoryOf* é propriedade inversa de *hasCategory*.

Por exemplo, no domínio da medicina, podemos classificar os problemas (ou doenças) como doenças virais, doenças bacterianas, etc.

É válido ressaltar que a propriedade *hasCategory* possui uma restrição de cadeia de propriedades (*propertyChainAxiom*). Este axioma indica que, dentro de uma hierarquia de categorias, se uma *IssueEntity* está categorizada por uma subcategoria, ela também está categorizada pela super categoria.

Supondo, por exemplo, uma hierarquia de categorias composta por *DoençasViraisHumanas* e *DoençasVirais*, onde a categoria *DoençasViraisHumanas* é uma subcategoria de *DoençasVirais*, se uma doença (*IssueEntity*) têm como categoria *DoençasViraisHumanas*, também terá como categoria *DoençasVirais*, pois toda doença viral humana é uma doença viral. Por meio dessa restrição, a máquina consegue inferir novas categorias para uma *IssueEntity*.

Sobre a propriedade *hasCategory*:

URI:	`ipo:hasCategory`
type:	`owl:ObjectProperty`
domain:	`ipo:IssueEntity`
range:	`skos:Concept`
inverseOf:	`ipo:categoryOf`
propertyChainAxiom:	`ipo:hasCategory - skos:broaderTransitive`

Sobre a propriedade *categoryOf*:

URI:	`ipo:categoryOf`
type:	`owl:ObjectProperty`
domain:	`skos:Concept`
range:	`ipo:IssueEntity`
inverseOf:	`ipo:hasCategory`

Propriedades: *hasDirectCategory e directCategoryOf*

hasDirectCategory é subpropriedade de *hasCategory* e indica uma categoria à qual a *IssueEntity* está diretamente relacionada. *directCategoryOf* é subpropriedade de *categoryOf* e propriedade inversa de *hasDirectCategory*.

Sobre a propriedade *hasDirectCategory*:

URI:	`ipo:hasDirectCategory`
type:	`owl:ObjectProperty`
domain:	`ipo:IssueEntity`
range:	`skos:Concept`
subProprertyOf:	`ipo:hasCategory`
inverseOf:	`ipo:directCategoryOf`

Sobre a propriedade *directCategoryOf:*

URI:	`ipo:directCategoryOf`
type:	`owl:ObjectProperty`
domain:	`skos:Concept`
range:	`ipo:IssueEntity`
subProprertyOf:	`ipo:categoryOf`
inverseOf:	`ipo:hasDirectCategory`

Propriedades: *hasAsset e assetOf*

hasAsset relaciona uma *IssueEntity* com um *Asset*. A propriedade *assetOf* é propriedade inversa de *hasAsset*. *assetOf* pode ser utilizada para facilitar a recuperação de registros de problemas relacionados ao *Asset*.

Sobre a propriedade *hasAsset*:

URI:	`ipo:hasAsset`
type:	`owl:ObjectProperty`
domain:	`ipo:IssueEntity`
range:	`ipo:Asset`
inverseOf:	`ipo:assetOf`

Sobre a propriedade *assetOf*:

URI:	`ipo:assetOf`
type:	`owl:ObjectProperty`
domain:	`ipo:Asset`
range:	`ipo:IssueEntity`
inverseOf:	`ipo:hasAsset`

Propriedades: *hasDepiction e depictionOf*

hasDepiction relaciona uma *IssueEntity* com uma *Image*, visando uma melhor descrição. *depictionOf* é propriedade inversa de *hasDepiction* e é funcional, ou seja, uma *Image* está relacionada por essa propriedade com no máximo uma única *IssueEntity*.

Sobre a propriedade *hasDepiction*:

URI:	`ipo:hasDepiction`
type:	`owl:ObjectProperty,` `owl:InverseFunctionalProperty`
domain:	`ipo:IssueEntity`
range:	`ipo:Image`
subPropertyOf:	`foaf:depiction`
inverseOf:	`ipo:depictionOf`

Sobre a propriedade *depictionOf*:

URI:	`ipo:depictionOf`
type:	`owl:ObjectProperty,` `owl:FunctionalProperty`
domain:	`ipo:Image`
range:	`ipo:IssueEntity`
inverseOf:	`ipo:hasDepiction`
subPropertyOf:	`foaf:depicts`

Propriedades: *hasMaker e makerOf*

hasMaker associa uma *IssueEntity* com uma Pessoa ou Organização (*Party*) que a criou ou registrou. Essa propriedade também é utilizada pela classe *Asset* para relacionar seu fabricante, desenvolvedor, inventor, etc. *makerOf* é propriedade inversa de *hasMaker*.

Sobre a propriedade *hasMaker*:

URI:	`ipo:hasMaker`
type:	`owl:ObjectProperty`
domain:	`ipo:IssueEntity or ipo:Asset`
range:	`ipo:Party`
subPropertyOf:	`foaf:maker`
inverseOf:	`ipo:makerOf`

Sobre a propriedade *makerOf*:

URI:	`ipo:makerOf`
type:	`owl:ObjectProperty`
domain:	`ipo:Party`
range:	`ipo:Asset or ipo:IssueEntity`
subPropertyOf:	`foaf:made`
inverseOf:	`ipo:hasMaker`

Propriedade: *description*

Uma descrição textual que descreve algo com mais detalhes.

Sobre a propriedade *description*:

URI:	`ipo:description`
type:	`owl:DatatypeProperty`
range:	`xsd:string`
subPropertyOf:	`dcterms:description, rdfs:comment`

Propriedade: *title*

Título (palavra ou frase) que resumidamente descreve algo.

Sobre a propriedade *title*:

URI:	`ipo:title`
type:	`owl:DatatypeProperty`
range:	`xsd:string`
subPropertyOf:	`dcterms:title, rdfs:label`

3.3.2 Pessoas e Organizações

Esse módulo provê termos para descrever pessoas e organizações, permitindo a representação de funcionários e colaboradores de empresas, definição de departamentos, entre outros.

Classe: *Party*

Superconceito (supertipo) comum aos conceitos Pessoa (*Person*) e Organização (*Organization*), que assume um papel de agente (*Party*) dentro do domínio abordado. Um agente é uma pessoa ou organização atuante, ou seja, capaz de realizar algo.

Sobre a classe *Party*:

URI:	`ipo:Party`
type:	`owl:Class`
equivalentClass:	`ipo:Organization or ipo:Person`
subClassOf:	`foaf:Agent, ipo:name some xsd:string`
disjointClass:	`ipo:Step, ipo:BooleanExpression, ipo:IssueEntity, ipo:Transition, ipo:Image, skos:Concept`

Classe: *Person*

Esta classe representa pessoas. Um exemplo seria uma pessoa que trabalha em uma organização e produz algum *Asset* ou registra uma *IssueEntity*.

Sobre a classe *Person*:

URI:	`ipo:Person`
type:	`owl:Class`
equivalentClass:	`foaf:Person`
subClassOf:	`ipo:Party`
disjointClass:	`ipo:Organization`

Classe: *Organization*

Representa um grupo de pessoas organizadas visando um objetivo em comum: social, econômico ou político. Por exemplo, uma empresa ou um departamento de uma empresa.

Sobre a classe *Organization*:

URI:	`ipo:Organization`
type:	`owl:Class`
equivalentClass:	`foaf:Organization`
subClassOf:	`ipo:Party`
disjointClass:	`ipo:Person`

Propriedades: *hasMember* e *memberOf*

A propriedade *hasMember* relaciona uma *Organization* com seus membros, que são instâncias de *Party*, podendo ser *Persons* ou outras *Organizations*. Por exemplo, esta propriedade poderia ser usada para registrar que um funcionário é membro de um departamento. Outra abordagem é utilizar essa propriedade para relacionar duas organizações, representado, por exemplo, um departamento que é membro de sua empresa. *memberOf* é propriedade inversa de *hasMember*.

Sobre a propriedade *hasMember*:

URI:	`ipo:hasMember`
type:	`owl:ObjectProperty`
domain:	`ipo:Organization`
range:	`ipo:Party`
equivalentProperty:	`foaf:member`
inverseOf:	`ipo:memberOf`

Sobre a propriedade *memberOf*:

URI:	`ipo:memberOf`
type:	`owl:ObjectProperty`
domain:	`ipo:Party`
range:	`ipo:Organization`
inverseOf:	`ipo:hasMember`

Propriedade: *name*

Indica um nome para identificação de algo.

Sobre a propriedade *name*:

URI:	`ipo:name`
type:	`owl:DatatypeProperty`
range:	`xsd:string`
equivalentProperty:	`foaf:name`

3.3.3 Problemas e Sintomas

Esse é o módulo principal, pois contém termos que representam sintomas e problemas (*Symptom, Issue*) e os relacionam, permitindo que a máquina deduza, a partir dos sintomas, os possíveis problemas.

Classe: *Issue*

Um problema ou questão a ser resolvida. Por exemplo, algo que não está operando normalmente ou um impedimento para realização de alguma tarefa.

Um *Issue* pode ser causa e/ou causado, direta ou indiretamente, por outro *Issue*, da mesma forma que o *Issue* A pode depender do *Issue* B, precisando que o *Issue* B seja solucionado antes de solucionar A.

O *Issue* pode ter um conjunto de ações (*Procedure*) que irá[47] solucioná-lo e ainda, pode ser indicado por vários sintomas (*Symptoms*), onde um conjunto de *Symptoms* pode identificar um *Issue*.

Sobre a classe *Issue*:

URI:	`ipo:Issue`
type:	`owl:Class`
subClassOf:	`ipo:IssueEntity, ipo:indecatedBy some ipo:Symptom`
disjointClass:	`ipo:Action, ipo:Symptom`

Classe: *Symptom*

Representa um sintoma (sinal ou indicação) de um ou vários problemas (*Issues)*. Sinal que é percebido quando um problema ocorre.

Sobre a classe *Symptom*:

URI:	`ipo:Symptom`
type:	`owl:Class`
subClassOf:	`ipo:IssueEntity, ipo:indicates some ipo:Issue`
disjointClass:	`ipo:Action, ipo:Issue.`

Propriedades: *indicates e indicatedBy*

indicates relaciona um *Symptom* com um *Issue que* ele indica. *indicatedBy* é propriedade inversa de *indicates*.

Um *Symptom* pode indicar vários *Issues*, como, por exemplo, um sintoma de febre pode indicar diversas doenças.

Sobre a propriedade *indicates*:

URI:	`ipo:indicates`
type:	`owl:ObjectProperty`

[47] Irá ou poderá solucioná-lo. Trata-se do registro de uma possível solução que já obteve sucesso em outras ocorrências.

domain:	`ipo:Symptom`
range:	`ipo:Issue`
inverseOf:	`ipo:indicatedBy`

Sobre a propriedade *indicatedBy*:

URI:	`ipo:indicatedBy`
type:	`owl:ObjectProperty`
domain:	`ipo:Issue`
range:	`ipo:Symptom`
inverseOf:	`ipo:indicates`

Propriedade: *causes e causedBy*

causes registra que um *Issue* causa outro *Issue* de forma direta ou indireta. Esta propriedade expressa uma relação de causalidade entre *Issues* e possui a característica de transitividade, ou seja, se o *Issue* A causa o *Issue* B e o *Issue* B causa o *Issue* C, então o *Issue* A causa o *Issue* C. *causedBy* é propriedade inversa de *causes*.

Sobre a propriedade *causes*:

URI:	`ipo:causes`
type:	`owl:ObjectProperty,` `owl:TransitiveProperty`
domain:	`ipo:Issue`
range:	`ipo:Issue`
inverseOf:	`ipo:causedBy`

Sobre a propriedade *causedBy*:

URI:	`ipo:causedBy`
type:	`owl:ObjectProperty,` `owl:TransitiveProperty`
domain:	`ipo:Issue`
range:	`ipo:Issue`
inverseOf:	`ipo:causes`

Propriedades: *directlyCauses e directlyCausedBy*

directlyCauses é subpropriedade de *causes* e indica que um *Issue* causa outro *Issue* de forma direta. *direcltyCausedBy* é subpropriedade de *causedBy* e propriedade inversa de *directlyCauses*.

Por exemplo, considere um paciente que está com Pneumonia devido a uma Gripe mal curada, neste caso a Gripe foi *causa direta* da Pneumonia. Pelo axioma de subpropriedade, infere-se também que a Gripe foi a *causa* da Pneumonia.

Sobre a propriedade *directlyCauses*:

URI:	`ipo:directlyCauses`
type:	`owl:ObjectProperty`
domain:	`ipo:Issue`
range:	`ipo:Issue`
subProprertyOf:	`ipo:causes`
inverseOf:	`ipo:directlyCausedBy`

Sobre a propriedade *directlyCausedBy*:

URI:	`ipo:directlyCausedBy`
type:	`owl:ObjectProperty`
domain:	`ipo:Issue`
range:	`ipo:Issue`
subProprertyOf:	`ipo:causedBy`
inverseOf:	`ipo:directlyCauses`

Propriedades: *dependsOn e dependenceOf*

dependsOn relaciona um *Issue* com outro *Issue* do qual o primeiro depende direta ou indiretamente. Esta propriedade expressa uma relação de dependência entre *Issues* e possui a característica de transitividade, ou seja, se o *Issue* A depende do *Issue* B e o *Issue* B depende do *Issue* C, então o *Issue* A depende do *Issue* C. *dependenceOf* é propriedade inversa de *dependsOn*.

Sobre a propriedade *dependsOn*:

URI:	ipo:dependsOn
type:	owl:ObjectProperty, owl:TransitiveProperty
domain:	ipo:Issue
range:	ipo:Issue
inverseOf:	ipo:dependenceOf

Sobre a propriedade *dependenceOf*:

URI:	ipo:dependenceOf
type:	owl:ObjectProperty, owl:TransitiveProperty
domain:	ipo:Issue
range:	ipo:Issue
inverseOf:	ipo:dependsOn

Propriedades: *directlyDependsOn e directDependenceOf*

direcltyDependsOn é subpropriedade de *dependsOn* e indica que um *Issue* depende diretamente de outro *Issue*. *directDependenceOf* é subpropriedade de *dependenceOf* e propriedade inversa de *directlyDependsOn*.

Com base no exemplo exposto ao descrever a propriedade *directlyCauses*, a Pneumonia *depende diretamente* que a Gripe seja curada para, enfim, ser tratada. Pelo axioma de subpropriedade, deduz-se também que a Pneumonia *depende* da Gripe.

Sobre a propriedade *directlyDependsOn*:

URI:	ipo:directlyDependsOn
type:	owl:ObjectProperty
domain:	ipo:Issue
range:	ipo:Issue
subPropertyOf:	ipo:dependsOn
inverseOf:	ipo:directDependenceOf

Sobre a propriedade *directDependenceOf*:

URI:	ipo:directDependenceOf
type:	owl:ObjectProperty
domain:	ipo:Issue
range:	ipo:Issue
subPropertyOf:	ipo:dependenceOf
inverseOf:	ipo:directlyDependsOn

Propriedades: *canCause* e *canBeCausedBy*

canCause indica que um *Issue* pode causar outro *Issue* de forma direta ou indireta. Esta propriedade visa expressar uma relação de possível causalidade e possui a característica de transitividade, ou seja, se o *Issue* A pode causar o *Issue* B e o *Issue* B pode causar o *Issue* C, então o *Issue* A pode causar o *Issue* C. *canBeCausedBy* é propriedade inversa de *canCause*.

Sobre a propriedade *canCause*:

URI:	ipo:canCause
type:	owl:ObjectProperty, owl:TransitiveProperty
domain:	ipo:Issue
range:	ipo:Issue
inverseOf:	ipo:canBeCausedBy

Sobre a propriedade *canBeCausedBy*:

URI:	ipo:canBeCausedBy
type:	owl:ObjectProperty, owl:TransitiveProperty
domain:	ipo:Issue
range:	ipo:Issue
inverseOf:	ipo:canCause

Propriedades: *canDirectlyCause* e *canBeDirectlyCausedBy*

canDirectlyCause é subpropriedade de *canCause* e indica que um *Issue* pode causar outro *Issue* de forma direta. Esta propriedade visa expressar uma

relação de possível causalidade, onde um *Issue* pode ser a causa direta de outro *Issue*. *canBeDirectlyCausedBy* é subpropriedade de *canBeCausedBy* e propriedade inversa de *canDirectlyCause*.

Por exemplo, a doença Gripe pode ser *causa direta* da Pneumonia, ou seja, em alguns casos a Gripe causa a Pneumonia e em outros casos não. Pelo axioma de subpropriedade, infere-se também que Gripe *pode causar* Pneumonia.

Sobre a propriedade *canDirectlyCause*:

URI:	`ipo:canDirectlyCause`
type:	`owl:ObjectProperty`
domain:	`ipo:Issue`
range:	`ipo:Issue`
subPropertyOf:	`ipo:canCause`
inverseOf:	`ipo:canBeDirectlyCausedBy`

Sobre a propriedade *canBeDirectlyCausedBy*:

URI:	`ipo:canBeDirectlyCausedBy`
type:	`owl:ObjectProperty`
domain:	`ipo:Issue`
range:	`ipo:Issue`
subPropertyOf:	`ipo:canBeCausedBy`
inverseOf:	`ipo:canDirectlyCause`

Propriedades: *canDependOn* e *canBeDependenceOf*

canDependOn indica que um *Issue* pode depender de outro *Issue* direta ou indiretamente. Esta propriedade expressa uma relação de possível dependência entre *Issues* e possui a característica de transitividade, ou seja, se o *Issue* A pode depender do *Issue* B e o *Issue* B pode depender do *Issue* C, então o *Issue* A pode depender do *Issue* C. *canBeDependenceOf* é propriedade inversa de *canDependOn*.

Sobre a propriedade *canDependOn*:

URI:	`ipo:canDependOn`

type:	owl:ObjectProperty, owl:TransitiveProperty
domain:	ipo:Issue
range:	ipo:Issue
inverseOf:	ipo:canBeDependenceOf

Sobre a propriedade *canBeDependenceOf*:

URI:	ipo:canBeDependenceOf
type:	owl:ObjectProperty, owl:TransitiveProperty
domain:	ipo:Issue
range:	ipo:Issue
inverseOf:	ipo:canDependOn

Propriedades: *canDirectlyDependOn* e *canBeDirectDependenceOf*

canDirectlyDependOn é subpropriedade de *canDependOn* e indica que um *Issue* pode depender diretamente de outro *Issue*. *canBeDirectDependenceOf* é subpropriedade de *canBeDependenceOf* e propriedade inversa de *canDirectlyDependOn*.

Com base no exemplo exposto ao descrever a propriedade *canDirectlyCause*, a Pneumonia *pode depender*, *de forma direta*, que a Gripe seja curada para, enfim, ser tratada. Pelo axioma de subpropriedade, Pneumonia *pode depender* de Gripe.

Sobre a propriedade *canDirectlyDependOn*:

URI:	ipo:canDirectlyDependOn
type:	owl:ObjectProperty
domain:	ipo:Issue
range:	ipo:Issue
subPropertyOf:	ipo:canDependOn
inverseOf:	ipo:canBeDirectDependenceOf

Sobre a propriedade *canBeDirectDependenceOf*:

URI:	ipo:canBeDirectDependenceOf
type:	owl:ObjectProperty
domain:	ipo:Issue
range:	ipo:Issue
subPropertyOf:	ipo:canBeDependenceOf
inverseOf:	ipo:canDirectlyDependOn

Propriedades: *hasCausativeAsset* e *causativeAssetOf*

hasCausativeAsset relaciona um *Issue* com algum agente (*Asset*) causador deste *Issue*. *causativeAssetOf* é propriedade inversa de *hasCausativeAsset*.

Por exemplo, uma doença relacionada com seu o agente causador (um vírus, bactéria, etc.).

Sobre a propriedade *hasCausativeAsset*:

URI:	ipo:hasCausativeAsset
type:	owl:ObjectProperty
domain:	ipo:Issue
range:	ipo:Asset
subPropertyOf:	ipo:hasAsset
inverseOf:	ipo:causativeAssetOf

Sobre a propriedade *causativeAssetOf*:

URI:	ipo:causativeAssetOf
type:	owl:ObjectProperty
domain:	ipo:Asset
range:	ipo:Issue
subPropertyOf:	ipo:assetOf
inverseOf:	ipo:hasCausativeAsset

Propriedades: *hasHostAsset* e *hostAssetOf*

hasHostAsset relaciona um *Issue* com hospedeiro (*Asset*) no qual este *Issue* ocorreu. *hostAssetOf* é propriedade inversa de *hasHostAsset*.

Por exemplo, uma doença relacionada com a pessoa doente, ou seja, a pessoa é o hospedeiro da doença.

Sobre a propriedade *hasHostAsset*:

URI:	`ipo:hasHostAsset`
type:	`owl:ObjectProperty`
domain:	`ipo:Issue`
range:	`ipo:Asset`
subPropertyOf:	`ipo:hasAsset`
inverseOf:	`ipo:hostAssetOf`

Sobre a propriedade *hostAssetOf*:

URI:	`ipo:hostAssetOf`
type:	`owl:ObjectProperty`
domain:	`ipo:Asset`
range:	`ipo:Issue`
subPropertyOf:	`ipo:assetOf`
inverseOf:	`ipo:hasHostAsset`

3.3.4 Problemas e Soluções

As classes e propriedades desse módulo relacionam a classe *Issue* com a classe *Procedure* que representa uma possível solução para o problema. Ainda provê meios para delinear uma solução por meio de um *workflow*.

Classe: *Action*

Representa uma ação a ser realizada para resolver o problema. Uma *Action* pode ser apenas uma ação primitiva ou tarefa (*Task*) ou um conjunto de ações (*CompoundAction*).

Sobre a classe *Action*:

URI:	`ipo:Action`

type:	`owl:Class`

subClassOf:	`ipo:IssueEntity,`
	`ipo:hasPostCondition some`
	`ipo:BooleanExpression,`
	`ipo:hasPreCondition some`
	`ipo:BooleanExpression`

disjointClass:	`ipo:Issue, ipo:Symptom`

Classe: *Task*

Uma ação elementar e atômica, que executa algo simples.

Sobre a classe *Task*:

URI:	`ipo:Task`
type:	`owl:Class`
subClassOf:	`ipo:Action`
disjointClass:	`ipo:CompoundAction`

Classe: *CompoundAction*

Uma ação composta por várias outras ações. Uma *CompoundAction* pode ter o objetivo de solucionar um ou mais *Issues*, representando um procedimento (*Procedure*) ou não ter um objetivo explícito, apenas ser um grupo de *Actions* a ser reusado, se comportando como uma atividade (*Activity)*. Uma *CompoundAction* possui um ou mais passos de execução (*Steps*) que ativam uma *Action* (*Task* ou outra *CompoundAction*), permitindo assim, que uma *CompoundAction* reuse outras *Actions*.

Uma *CompoundAction* pode ser utilizada para criar uma estrutura de *workflow*, visando uma melhor estruturação das ações que a compõem.

Sobre a classe *CompoundAction*:

URI:	`ipo:CompoundAction`
type:	`owl:Class`

subClassOf:	ipo:Action, ipo:hasFirstStep some ipo:Step, ipo:hasStep some ipo:Step
disjointClass:	ipo:Task

Classe: *Procedure*

Uma sequência de passos *(Steps)* com objetivo explícito que após executados solucionam um ou mais *Issues.*

Sobre a classe *Procedure*:

URI:	ipo:Procedure
type:	owl:Class
subClassOf:	ipo:CompoundAction, ipo:solves some ipo:Issue
disjointClass:	ipo:Activity

Classe: *Activity*

Um conjunto de passos (*Steps*) que realizam uma atividade, porém não tem um objetivo explícito e, portanto, não visa solucionar um determinado *Issue*. Na verdade, trata-se de um agrupamento de ações tendo em vista o reuso.

Sobre a classe *Activity*:

URI:	ipo:Activity
type:	owl:Class
subClassOf:	ipo:CompoundAction
disjointClass:	ipo:Procedure

Classe: *Step*

Um passo ou etapa a ser realizada dentro de uma *CompoundAction*. Todo *Step* possui uma *Action* a ser executada e uma transição para outro *Step* a ser efetuada depois de finalizada a execução da *Action*.

Com os *Steps* é possível estabelecer uma ordem para a execução das *Actions*, pois cada *Step* possui uma transição (*Transition*) que define o *Step* de

origem e de destino. Uma vez que uma *CompoundAction* possui um *Step* inicial (*hasFirstStep*), a partir desse *Step*, é possível executar todos os outros *Steps* que compõe a *CompoundAction,* seguindo a transição entre eles.

Sobre a classe *Step*:

URI:	`ipo:Step`
type:	`owl:Class`
disjointClass:	`ipo:IssueEntity, ipo:Image, ipo:Transition, ipo:BooleanExpression, ipo:Party, skos:Concept`

Classe: *Transition*

Transition representa a transição entre dois *Steps*. Cada *Transition* tem um *Step* de origem (*source*) e um *Step* de destino (*target*). Uma *Transition* possui uma condição de guarda que especifica uma expressão booleana (*BooleanExpression*) que tem que ser verdadeira para que a transição ocorra.

Através da condição de guarda, pode-se implementar, de maneira simplificada, um *workflow*, estabelecendo estruturas de decisão, repetição, escolha, etc.

Sobre a classe *Transition*:

URI:	`ipo:Transition`
type:	`owl:Class`
subClassOf:	`ipo:hasGuardCondition some ipo:BooleanExpression, ipo:hasSource some ipo:Step, ipo:hasTarget some ipo:Step`
disjointClass:	`ipo:IssueEntity, ipo:Image, ipo:Step, ipo:BooleanExpression, ipo:Party, skos:Concept`

Classe: *BooleanExpression*

Uma expressão lógica cujo valor é verdadeiro ou falso, usada para validar uma transição entre dois *Steps,* ou servir com pré-condição para a execução de uma *Action* ou ainda, servir como pós-condição que valide a execução de uma *Action*.

Essa classe possui uma descrição da expressão e dois valores possíveis: *true* ou *false*. Para esses valores, foram criadas duas instâncias para serem reusadas.

Sobre a classe *BooleanExpression*:

URI:	`ipo:BooleanExpression`
type:	`owl:Class`
subClassOf:	`ipo:description some xsd:string`
disjointClass:	`ipo:IssueEntity, ipo:Image, ipo:Step, ipo:Transition, ipo:Party, skos:Concept`
instancesValues:	`ipo:true, ipo:false`

Propriedade: *hasPreCondition*

Propriedade funcional e funcional inversa que indica a pré-condição para que a *Action* seja executada. Pré-condição representa o pré-requisito definido por uma expressão booleana (*BooleanExpression*) para execução da *Action*, sem o qual não é garantida a execução correta da mesma.

Sobre a propriedade *hasPreCondition*:

URI:	`ipo:hasPreCondition`
type:	`owl:ObjectProperty, owl:FunctionalProperty, owl:InverseFunctionalProperty`
domain:	`ipo:Action`
range:	`ipo:BooleanExpression`

Propriedade: *hasPostCondition*

Propriedade funcional e funcional inversa que indica a pós-condição (efeito) definida por uma expressão booleana (*BooleanExpression*) que será atingida após a execução da *Action*, desde que a pré-condição tenha sido respeitada.

Sobre a propriedade *hasPostCondition*:

URI:	`ipo:hasPostCondition`

type:	`owl:ObjectProperty,`
	`owl:FunctionalProperty,`
	`owl:InverseFunctionalProperty`

domain:	`ipo:Action`
range:	`ipo:BooleanExpression`

Propriedade: *hasStep*

Indica um passo de execução (*Step*) que compõe a *CompoundAction*.

Essa propriedade possui uma restrição de cadeia de propriedades (*propertyChainAxiom*). Este axioma, exposto abaixo, provê à máquina condições para, a partir de um *Step* que aciona outro *Step* por meio de uma *Transition*, relacionar ambos os *Steps* como *Steps* do *Procedure*.

Por exemplo, supondo que o *Procedure* P possui um *Step* A e o *Step* A aciona o *Step* B por meio de uma *Transition* T, com base nesta *property chain*, a máquina conseguirá relacionar ambos os *Steps* (A e B) como sendo *Steps* (*hasStep*) do *Procedure* P.

Essa propriedade seria funcional inversa, porém, esse tipo de restrição não pode coexistir com uma *property chain*. Isso se deve aos raciocinadores serem OWL 2 DL (*Descriptions Logics*), isto é, por se basearem em lógica descritiva, algumas exigências são necessárias para garantir decidibilidade (todas as computações terminarem em tempo finito). Dentre elas, alguns axiomas, listados a seguir, somente podem ocorrer em propriedade simples, ou seja, propriedade que não possui, direta ou indiretamente, superpropriedades que são transitivas ou definidas por *property chains*.

- *ObjectMinCardinality, ObjectMaxCardinality, ObjectExactCardinality* e *ObjectHasSelf.*

- *FunctionalObjectProperty, InverseFunctionalObjectProperty, IrreflexiveObjectProperty, AsymmetricObjectProperty,* e *DisjointObjectProperties.*

Sobre a propriedade *hasStep*:

URI:	`ipo:hasStep`
type:	`owl:ObjectProperty`
domain:	`ipo:CompoundAction`
range:	`ipo:Step`
propertyChainAxiom:	`ipo:hasStep – ipo:hasOutcoming – ipo:hasTarget`

Propriedade: *hasFirstStep*

Uma *CompoundAction* possui um ou mais passos (*Steps*), assim essa propriedade indica o primeiro passo pelo qual a execução da *CompoundAction* é iniciada. Trata-se de uma propriedade funcional e funcional inversa, pois cada *CompoundAction* possui apenas um único primeiro passo e este também pertence a uma única *CompoundAction*.

Sobre a propriedade *hasFirstStep*:

URI:	`ipo:hasFirstStep`
type:	`owl:ObjectProperty,` `owl:FunctionalProperty,` `owl:InverseFunctionalProperty`
domain:	`ipo:CompoundAction`
range:	`ipo:Step`
subPropertyOf:	`ipo:hasStep`

Propriedade: *goal*

Descrição do objetivo que se deseja alcançar após a execução do *Procedure*.

Sobre a propriedade *goal*:

URI:	ipo:goal
type:	owl:DatatypeProperty
domain:	ipo:Procedure
range:	xsd:string

Propriedades: *solves e solvedBy*

solves relaciona a *Procedure* com o *Issue* que a ela soluciona. *solvedBy* é propriedade inversa de *solves*.

Sobre a propriedade *solves*:

URI:	ipo:solves
type:	owl:ObjectProperty
domain:	ipo:Procedure
range:	ipo:Issue
inverseOf:	ipo:solvedBy

Sobre a propriedade *solvedBy*:

URI:	ipo:solvedBy
type:	owl:ObjectProperty
domain:	ipo:Issue
range:	ipo:Procedure
inverseOf:	ipo:solves

Propriedade: *activates*

Propriedade funcional que indica a ação a ser ativada por um *Step*.

Sobre a propriedade *activates*:

URI:	ipo:activates
type:	owl:ObjectProperty, owl:FunctionalProperty
domain:	ipo:Step
range:	ipo:Action

Propriedades: *hasIncoming e hasTarget*

hasIncoming indica as transições (*Transitions*) de entrada do *Step*, ou seja, as que iniciam a execução deste *Step*. *hasTarget* é propriedade inversa de *hasIncoming*, indicando o *Step* destino da *Transition*. *hasTarget* é também uma propriedade funcional, uma vez que uma Transition tem um único *Step* destino.

Sobre a propriedade *hasIncoming*:

URI:	`ipo:hasIncoming`
type:	`owl:ObjectProperty,` `owl:InverseFunctionalProperty`
domain:	`ipo:Step`
range:	`ipo:Transition`
inverseOf:	`ipo:hasTarget`

Sobre a propriedade *hasTarget*:

URI:	`ipo:hasTarget`
type:	`owl:ObjectProperty,` `owl:FunctionalProperty`
domain:	`ipo:Transition`
range:	`ipo:Step`
inverseOf:	`ipo:hasIncoming`

Propriedades: *hasOutcoming e hasSource*

hasOutcoming indica as transições (*Transitions*) de saída do *Step,* ou seja, as que ocorrem após a execução deste *Step* e que acionam os próximos *Steps* a serem executados. *hasSource* é propriedade inversa de *hasOutcoming*, indicando o *Step* de origem da *Transition*. *hasSource* é também uma propriedade funcional, uma vez que uma *Transition* tem um único *Step* de origem.

Sobre a propriedade *hasOutcoming*:

URI:	`ipo:hasOutcoming`
type:	`owl:ObjectProperty,` `owl:InverseFunctionalProperty`

domain:	`ipo:Step`
range:	`ipo:Transition`
inverseOf:	`ipo:hasSource`

Sobre a propriedade *hasSource*:

URI:	`ipo:hasSource`
type:	`owl:ObjectProperty,` `owl:FunctionalProperty`
domain:	`ipo:Transition`
range:	`ipo:Step`
inverseOf:	`ipo:hasOutcoming`

Propriedade: *hasGuardCondition*

Propriedade funcional e funcional inversa que indica uma condição booleana (*BooleanExpression*) para que uma *Transition* ocorra.

Por exemplo, uma *Transition* pode ter como *guardCondition* que o *Step* de origem seja executado 10 vezes. Assim, enquanto este *Step* não for executado 10 vezes, não será iniciando o *Step* destino. Este exemplo ilustra uma estrutura de repetição dentro de um *workflow*.

Sobre a propriedade *hasGuardCondition*:

URI:	`ipo:hasGuardCondition`
type:	`owl:ObjectProperty,` `owl:FunctionalProperty,` `owl:InverseFunctionalProperty`
domain:	`ipo:Transition`
range:	`ipo:BooleanExpression`

3.4 EXEMPLO DE INSTANCIAÇÃO DA ONTOLOGIA

Para demonstrar o uso da ontologia IPO, a Figura 20 ilustra o mapeamento de instâncias (RDF) para classes/propriedades da ontologia (OWL), para o domínio de problemas médicos (doenças), onde foi abordada a *Gripe* como exemplo de instância da classe *Issue*. É demonstrado o relacionamento entre o problema *Gripe*

com o sintoma *Febre*, instância da classe *Symptom*. O problema Gripe também é relacionado com um possível tratamento, instância da classe *Procedure*. Outro relacionamento representado é o vírus *Influenza,* instância da classe *Asset,* que possui uma relação de causador da Gripe. Além disso, é demonstrado um relacionamento entre o problema Gripe com seu hospedeiro, no caso, o Ser Humano, também instância da classe *Asset*. Como forma de organização, a Gripe foi agrupada na categoria *Doenças Virais*, instância da classe *skos:Concept*. Para demonstrar a dependência entre problemas, foi adicionado o problema Pneumonia, onde a Gripe pode causar Pneumonia e Pneumonia pode depender que a Gripe seja solucionada para, então, ser tratada. Os relacionamentos representados na cor verde são inferidos automaticamente pela máquina com base no axioma de subpropriedade. Para todos os outros relacionamentos, com base no axioma de propriedade inversa, os respectivos relacionamentos inversos também seriam inferidos, mas não foram ilustrados para não sobrecarregar a Figura 20.

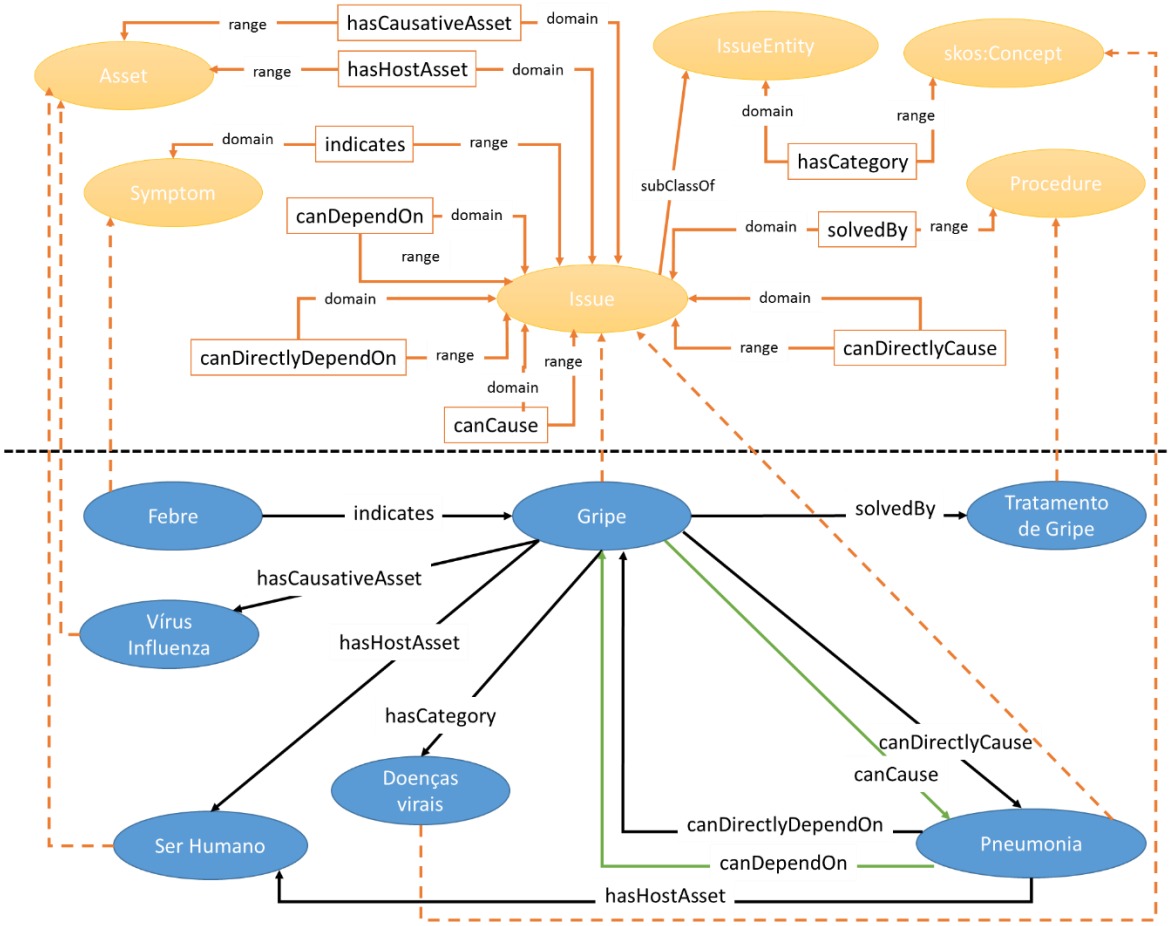

Figura 20 - Mapeamento entre dados RDF e a ontologia IPO
Fonte: O autor

3.5 AVALIAÇÃO DA *ISSUE PROCEDURE ONTOLOGY*

A avaliação é uma etapa importante no processo de criação de uma ontologia. Para realizar essa avaliação, se faz necessário verificar se a ontologia é expressiva o suficiente para prover respostas para as questões de competência levantadas anteriormente e verificar se ela consegue representar os dados de interesses pertinentes ao domínio.

Como dito anteriormente, existem duas formas de classificação de uma *IssueEntity:* (1) Uma classificação não intrínseca, ou seja, uma classificação visando um simples agrupamento de *IssueEntities,* onde a ontologia IPO provê a utilização da propriedade *hasCategory* relacionando com *skos:Concept*, possibilitando a criação de esquemas de categorias e subcategorias de *IssueEntities*. (2) Uma classificação intrínseca a um *IssueEntity*, de modo que essa classificação implica na criação de subclasses de *IssueEntity*, com restrições específicas. Por exemplo, a classe *Gripe*, sendo definida por uma restrição específica que é estar relacionado a pelo menos um dos sintomas *Febre* ou *Coriza* (instância da classe *Symptom*). Desta forma, ocorrências podem ser automaticamente classificadas, com base na descrição de seus sintomas, como sendo uma instância da nova classe *Gripe*.

A seguir, para cada questão de competência, será efetuada uma instanciação da ontologia visando validar se a ontologia provê meios para respondê-las. Cada instanciação será demonstrada por meio de uma figura e uma consulta SPARQL será utilizada para obter a resposta.

Os relacionamentos expostos nas figuras, na cor preta, indicam que são relacionamentos informados pelo ser humano. Já os relacionamentos na cor verde, indicam relacionamentos obtidos automaticamente pela máquina por meio de softwares raciocinadores OWL. Os relacionamentos indicados por uma linha tracejada ilustram uma relação de um recurso RDF com uma classe OWL, isto é, uma instância com a classe à qual pertence.

Os prefixos utilizados nos exemplos são:

```
@prefix ipo: <https://purl.org/ipo/core#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: < http://www.w3.org/2000/01/rdf-schema#>.
```

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix ex: <http://www.example.org/>
```

QC1. Qual(is) categoria(s) de um problema/sintoma/solução?

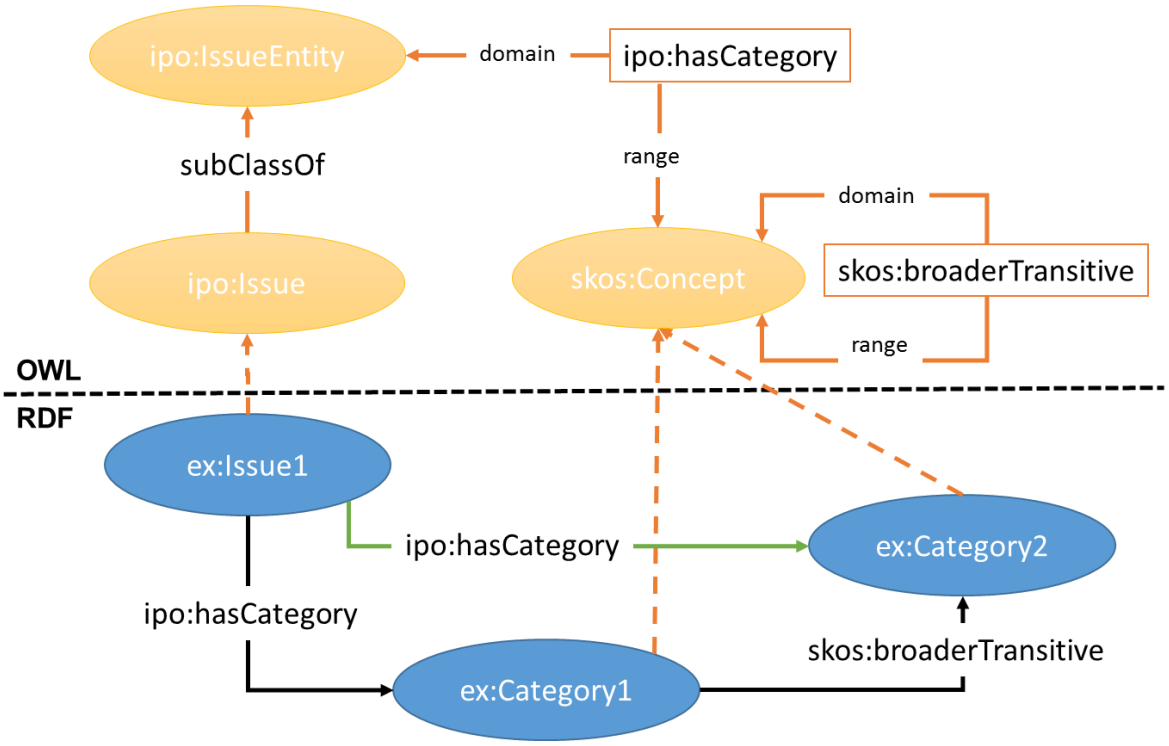

Figura 21 - Exemplo de instanciação usando a propriedade *hasCategory*
Fonte: O autor

Consulta SPARQL:

```
SELECT distinct ?category
WHERE {
  ex:Issue1 ipo:hasCategory ?category .
}
```

A consulta acima terá como resposta a categoria `ex:Category1` devido a relação `ipo:hasCategory` relacionando o *Issue* `ex:Issue1` com a categoria `ex:Category1`. Contudo, o vocabulário SKOS, possui a propriedade `skos:broaderTransitive` que indica uma relação de hierarquia, onde `ex:Category1 skos:broaderTransitive ex:Category2` expressa que a categoria `ex:Category1` é uma subcategoria de `ex:Category2`. Assim, por meio da restrição de *property chain* da propriedade hasCategory, o raciocinador irá

inferir a seguinte tripla: `ex:Issue1 ipo:hasCategory ex:Category2`. Com essa nova tripla, a consulta SPARQL acima terá como resposta as duas categorias (`ex:Category1 e ex:Category2`).

Este exemplo mostra a possibilidade de se usar o vocabulário SKOS para criação de esquemas de classificação.

QC2. Quais são sintomas de um problema?

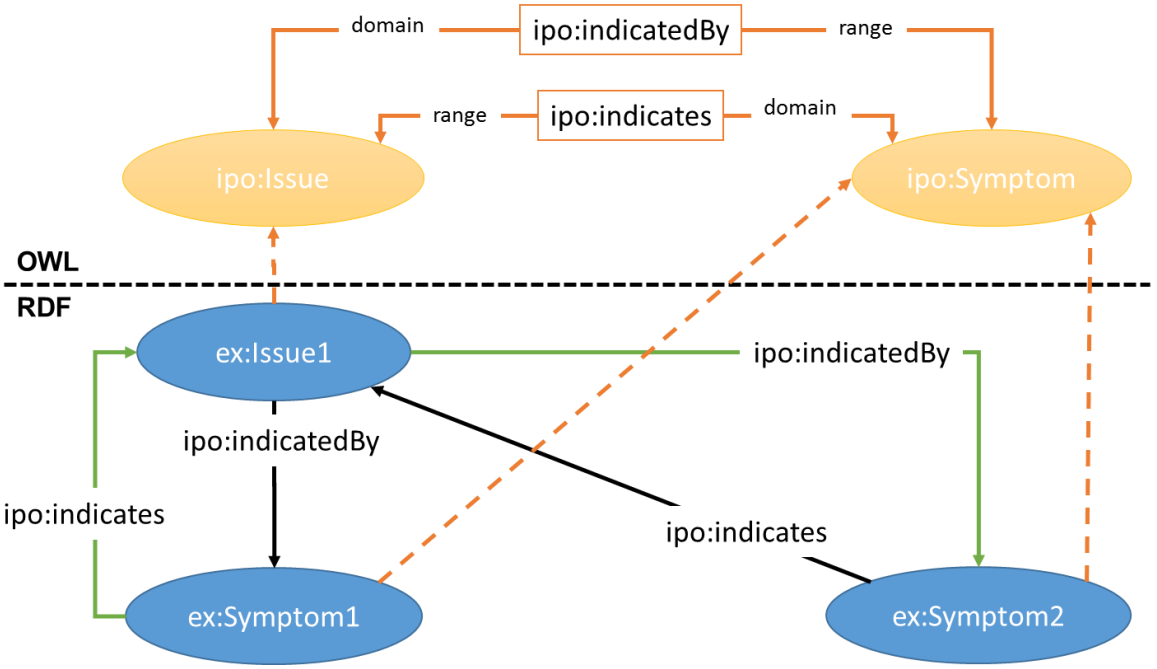

Figura 22 - Exemplo de instanciação usando as propriedades *indicatedBy* e *indicates*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?symptom
WHERE {
    ex:Issue1 ipo:indicatedBy ?symptom .
}
```

A consulta acima retornará o *Symptom* `ex:Symptom1` devido a relação `ipo:indicatedBy` relacionando o *Issue* `ex:Issue1` com o *Symptom* `ex:Symptom1`. Porém, ao utilizar um raciocinador, a resposta será composta por dois *Symptoms*: `ex:Symptom1` e `ex:Symptom2`. Isso ocorre porque a propriedade `ipo:indicates` é inversa da propriedade `ipo:indicatedBy`, assim o

raciocinador inferirá a seguinte tripla: `ex:Issue1 ipo:indicatedBy ex:Symptom2`[48].

QC3. Quais são as soluções de um problema?

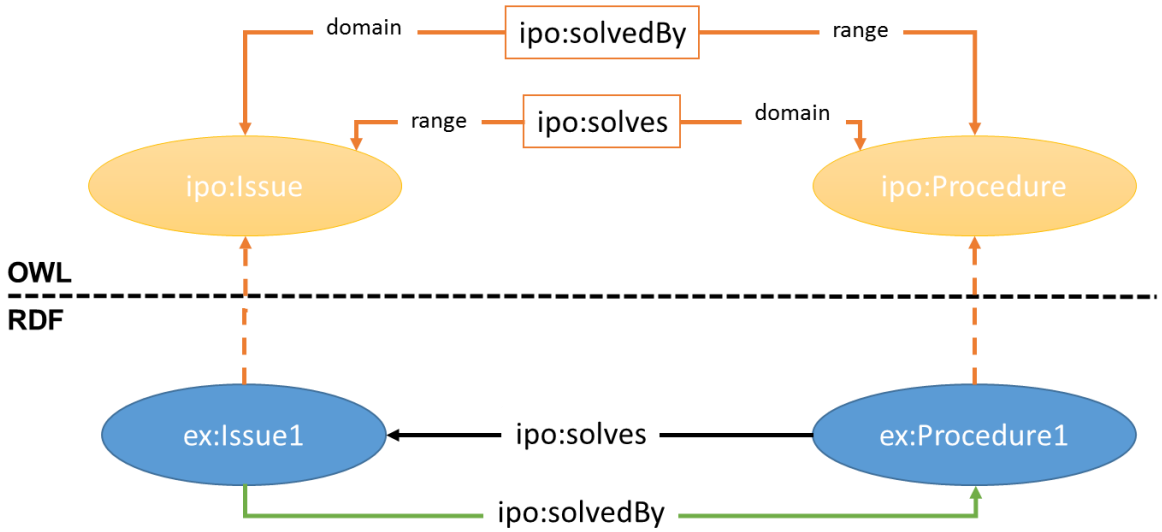

Figura 23 - Exemplo de instanciação usando as propriedades *solves* e *solvedBy*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?solution
WHERE {
    ex:Issue1 ipo:solvedBy ?solution .
}
```

A princípio, a consulta acima não teria êxito devido a relação `ipo:solvedBy` não estar expressamente indicada na instanciação. Porém, ao utilizar um raciocinador, a resposta será composta pelo procedimento: `ex:Procedure1`. Isso ocorre porque a propriedade `ipo:solves` é inversa da propriedade `ipo:solvedBy`, assim o raciocinador inferirá a seguinte tripla: `ex:Issue1 ipo:solvedBy ex:Procedure1`.

[48] Como já foi dito, os relacionamentos obtidos através de um raciocinador são demonstrados nas figuras por meio da cor verde, contudo, é importante ressaltar que outros relacionamentos podem ser inferidos ao se utilizar um raciocinador, porém, para não sobrecarregar a figura, optou-se por expor somente os relacionamentos que estão relacionados com a questão de competência que se deseja responder.

QC4. Qual o causador do problema?

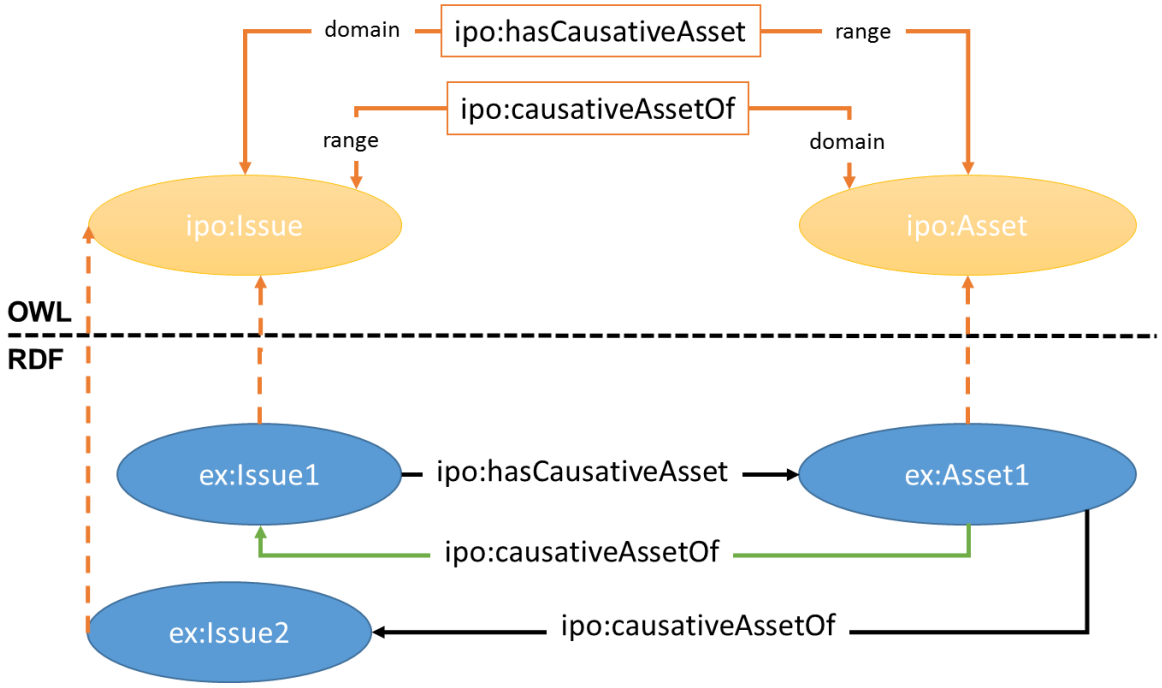

Figura 24 - Exemplo de instanciação usando a propriedade *hasCausativeAsset* e *causativeAssetOf*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?causativeAsset
WHERE {
    ex:Issue1 ipo:hasCausativeAsset ?causativeAsset .
}
```

A consulta acima terá como resposta o *Asset* `ex:Asset1`, devido a relação entre o *Issue* `ipo:Issue1` e o *Asset* `ipo:Asset1` através da propriedade `ipo:hasCausativeAsset`. Por meio da propriedade inversa `ipo:causativeAssetOf`, também seria possível obter todos os problemas causados pelo `ex:Asset1`, no exemplo, os *Issues* `ex:Issue1` e `ex:Issue2`.

QC5. Qual o hospedeiro do problema?

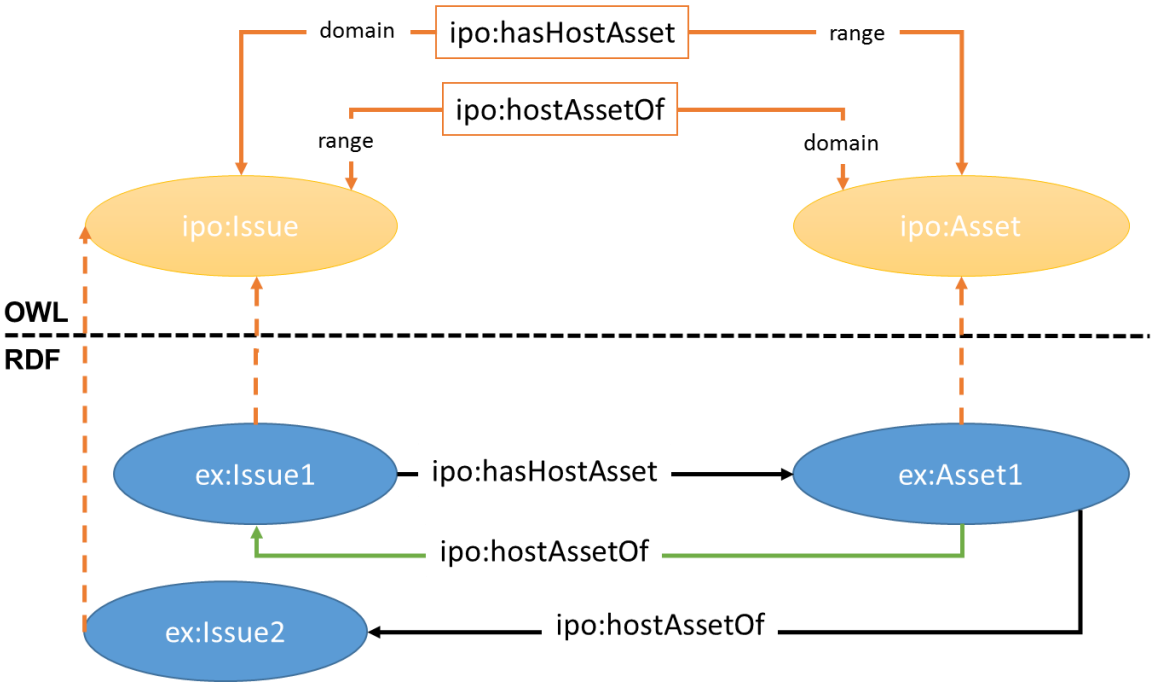

Figura 25 - Exemplo de instanciação usando a propriedade *hasHostAsset* e *hostAssetOf*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?hostAsset
WHERE {
    ex:Issue1 ipo:hasHostAsset ?hostAsset .
}
```

A consulta acima terá como resposta o *Asset* `ex:Asset1`, devido a relação entre o *Issue* `ex:Issue1` e o *Asset* `ex:Asset1` através da propriedade `ipo:hasHostAsset`. Por meio da propriedade inversa `ipo:hostAssetOf`, também seria possível obter todos os problemas que ocorreram no `ex:Asset1`, no exemplo, os *Issues* `ex:Issue1` e `ex:Issue2`.

QC6. Quem criou/registrou esse problema/sintoma/solução?

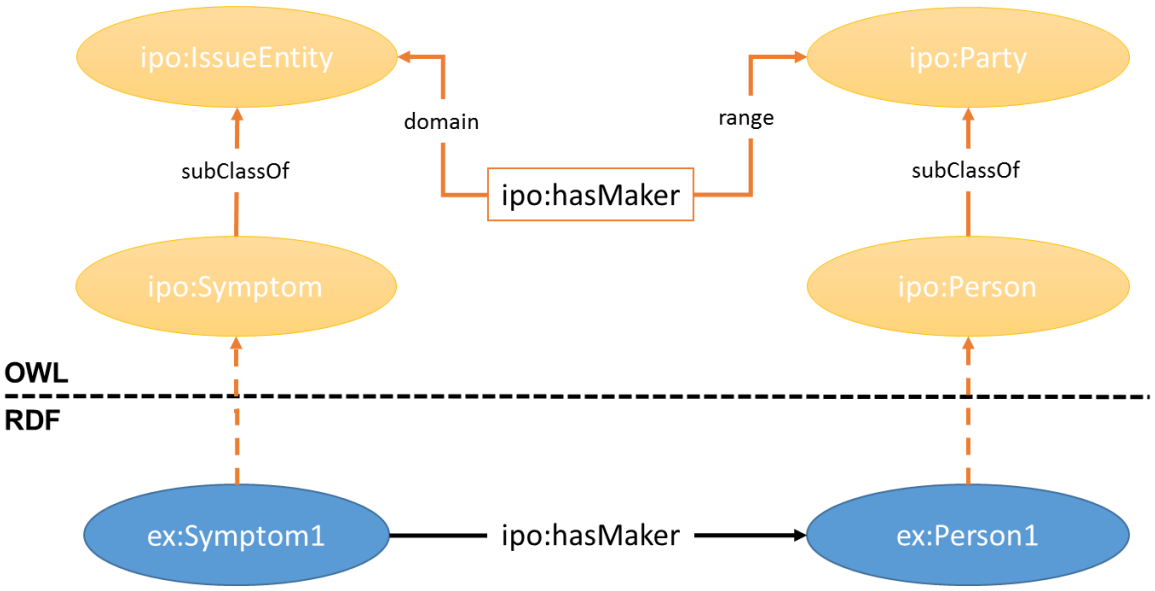

Figura 26 - Exemplo de instanciação usando a propriedade *hasMaker*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?creator
WHERE {
    ipo:Symptom1 ipo:hasMaker ?creator .
}
```

A consulta acima terá como resposta a *Person* `ex:Person1`, devido a relação entre o *Symptom* `ex:Symptom1` e a *Person* `ex:Person1` através da propriedade `ipo:hasMaker`. Como a propriedade `ipo:hasMaker` possui como domínio (*domain*) a classe `ipo:IssueEntity`, tanto um *Symptom*, quanto um *Issue* ou *Action* podem fazer uso dessa propriedade para indicar uma *Party* (*Person* ou *Organization*) como seu criador.

QC7. Quais ações (*workflow*) a serem tomadas para solucionar o problema?

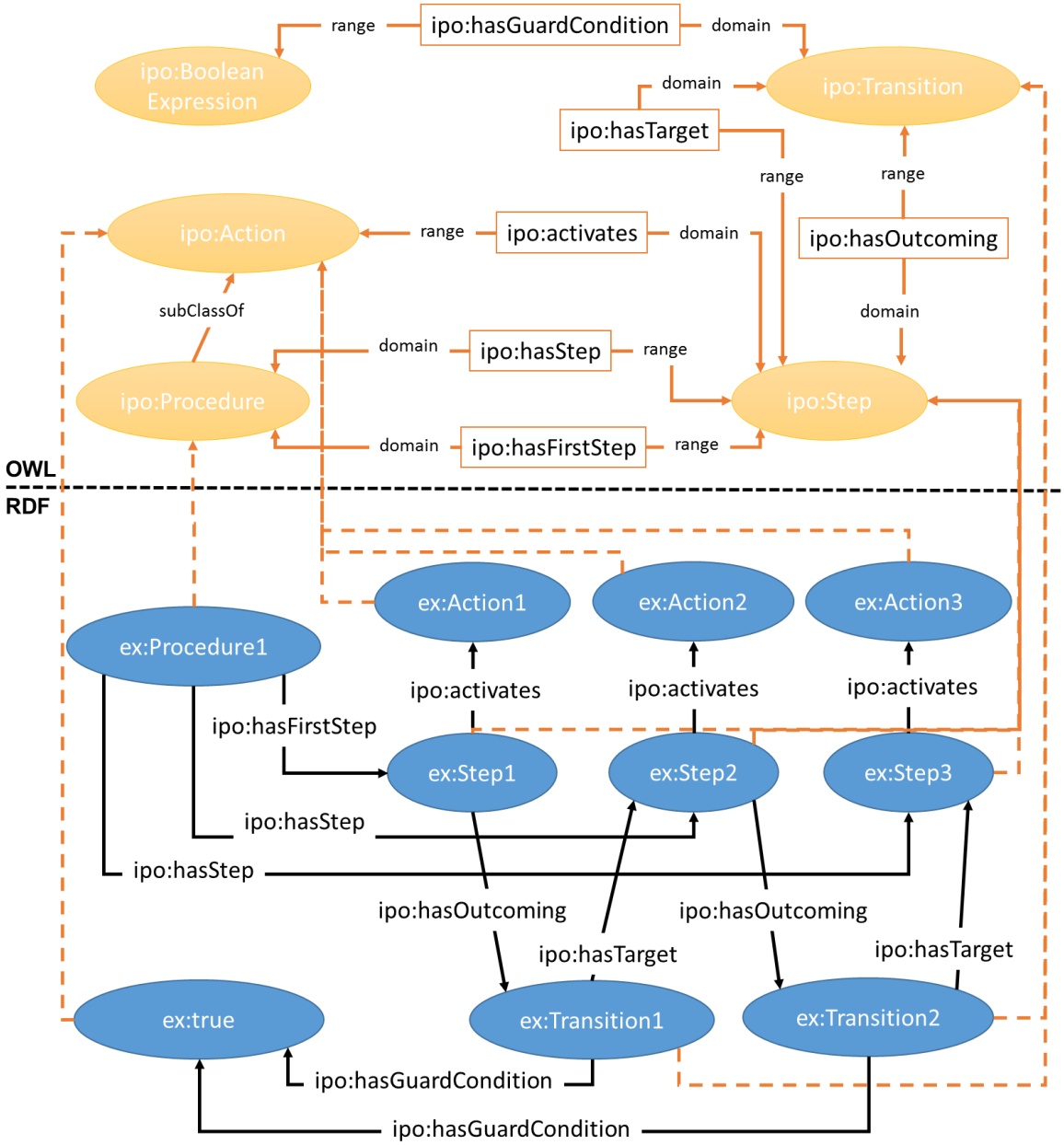

Figura 27 - Exemplo de instanciação de um *workflow* simples
Fonte: O autor

(1) Consulta SPARQL para obter o primeiro passo do *workflow* do procedimento (*Procedure*):

```
SELECT ?firstStep
WHERE {
    ex:Procedure1 ipo:hasFirstStep ?firstStep.
}
```

(2) Consulta SPARQL recursiva parametrizada para obter a ação e próximos passos de cada passo (parâmetro *currentStep*):

```
SELECT ?action ?transition ?guard ?nextStep
WHERE {
    <currentStep> ipo:activates ?action.
    OPTIONAL {
         <currentStep> ipo:hasOutcoming ?transition.
         ?transition ipo:hasGuardCondition ?guard.
         ?transition ipo:hasTarget ?nextStep
    }
}
```

Para se obter as ações de um *Procedure* observando a ordem de execução, se faz necessário realizar várias consultas SPARQL. Acima estão representadas duas consultas SPARQL para obter essa resposta.

A consulta (1) obtém o primeiro *Step* do *Procedure* `ex:Procedure1` através da propriedade `ipo:hasFirstStep`, que relaciona o *Procedure* `ex:Procedure1` com o *Step* `ex:Step1`. Ao se obter o primeiro *Step*, é possível identificar a *Action* a ser executada nesse *Step* com a consulta (2), além de retornar a *Transition* para o próximo *Step*, a condição de guarda (*BooleanExpression*) dessa *Transition* e o próximo *Step*. Essas informações são relevantes, pois só poderá ocorrer a transição, ou seja, executar o próximo *Step*, caso a condição de guarda seja verdadeira. No exemplo de instanciação acima, as condições de guarda utilizam uma instância de *BooleanExpression* (`ipo:true`) predefinida pela ontologia IPO.

A consulta (2) deve ser executada tantas vezes quanto forem necessárias até que todos os *Steps* sejam recuperados.

QC8. Um problema causa outro problema?

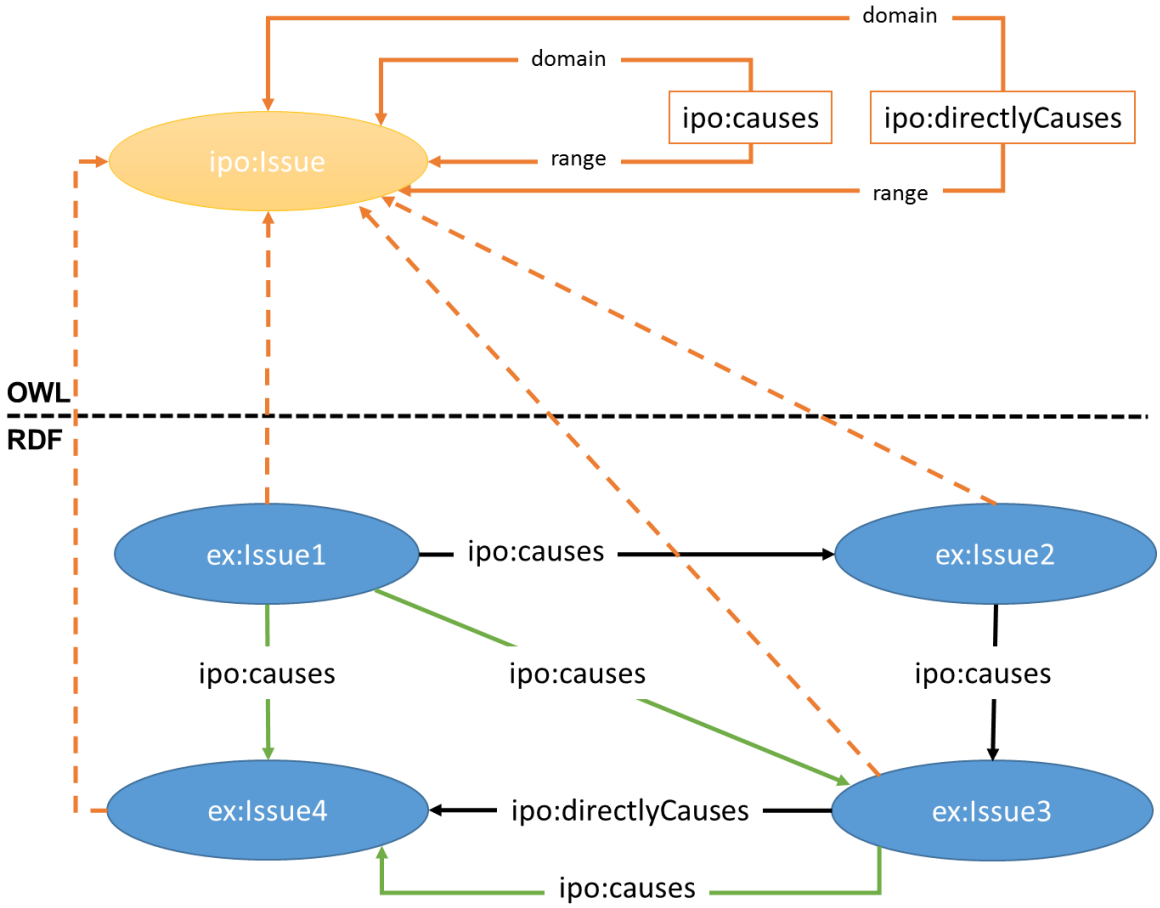

Figura 28 - Exemplo de instanciação usando a propriedade *causes* e *directlyCauses*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?anotherIssue
WHERE {
    ex:Issue1 ipo:causes ?anotherIssue .
}
```

Baseado somente nas triplas acima, a consulta SPARQL retornará como resposta somente o *Issue* `ex:Issue2`, devido a tripla: `ex:Issue1 ipo:causes ex:Issue2`. Porém, ao se utilizar um raciocinador, a resposta conterá os *Issues* `ex:Issue2`, `ex:Issue3`, `ex:Issue4`. Isso ocorre porque a propriedade `ipo:causes` é transitiva, ou seja, se `ex:Issue1` causa `ex:Issue2` e `ex:Issue2` causa `ex:Issue3`, então `ex:Issue1` causa `ex:Issue3`. O *Issue* `ex:Issue3` não está explicitamente relacionado com o `ex:Issue4` através da propriedade `ipo:causes`, porém, como a propriedade `ipo:directlyCauses` é

subpropriedade de `ipo:causes`, o raciocinador irá inferir a tripla: `ex:Issue3 ipo:causes ex:Issue4`. Em seguida, novamente por causa do caráter transitivo da propriedade `ipo:causes`, a tripla `ex:Issue1 ipo:causes ex:Issue4` também será inferida.

QC9. Um problema depende de outro problema?

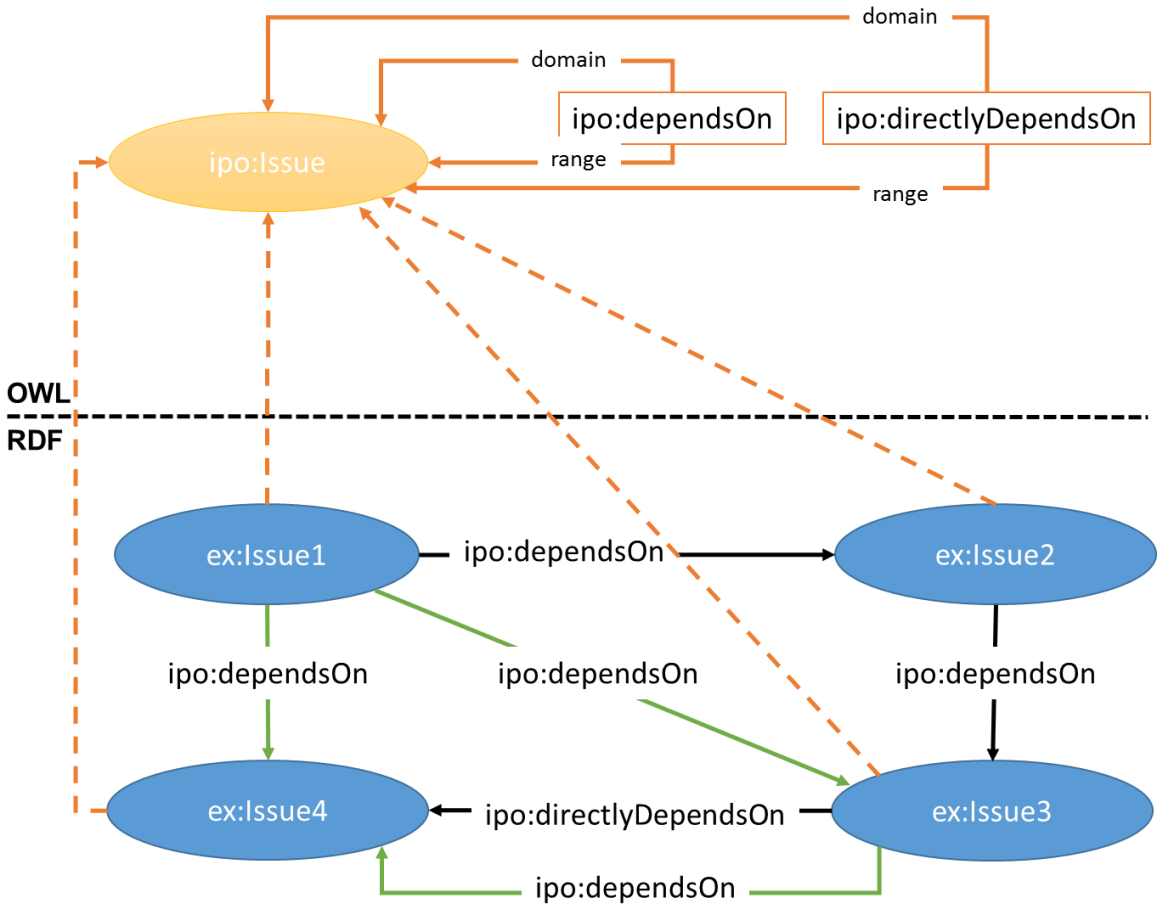

Figura 29 - Exemplo de instanciação usando a propriedade *dependsOn* e *directlyDependsOn*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?anotherIssue
WHERE {
    ex:Issue1 ipo:dependsOn ?anotherIssue .
}
```

Baseado somente nas triplas explicitamente declaradas, a consulta SPARQL retornará somente o *Issue* `ex:Issue2`, devido a tripla: `ex:Issue1`

`ipo:dependsOn ex:Issue2`. Contudo, ao se utilizar um raciocinador, a resposta conterá os *Issues* `ex:Issue2, ex:Issue3, ex:Issue4`. Isso ocorre porque a propriedade `ipo:dependsOn` é transitiva, ou seja, se `ex:Issue1` depende de `ex:Issue2` e `ex:Issue2` depende de `ex:Issue3`, então `ex:Issue1` depende `ex:Issue3`. O Issue `ex:Issue3` não está relacionado explicitamente com o `ex:Issue4` através da propriedade `ipo:dependsOn`, porém, como a propriedade `ipo:directlyDependsOn` é subpropriedade de `ipo:dependsOn`, o raciocinador irá inferir a tripla: `ex:Issue3 ipo:dependsOn ex:Issue4.` Em seguida, novamente por causa do caráter transitivo da propriedade `ipo:dependsOn`, a tripla `ex:Issue1 ipo:dependsOn ex:Issue4` também será inferida.

QC10. Quais os possíveis problemas, a partir de um conjunto de sintomas?

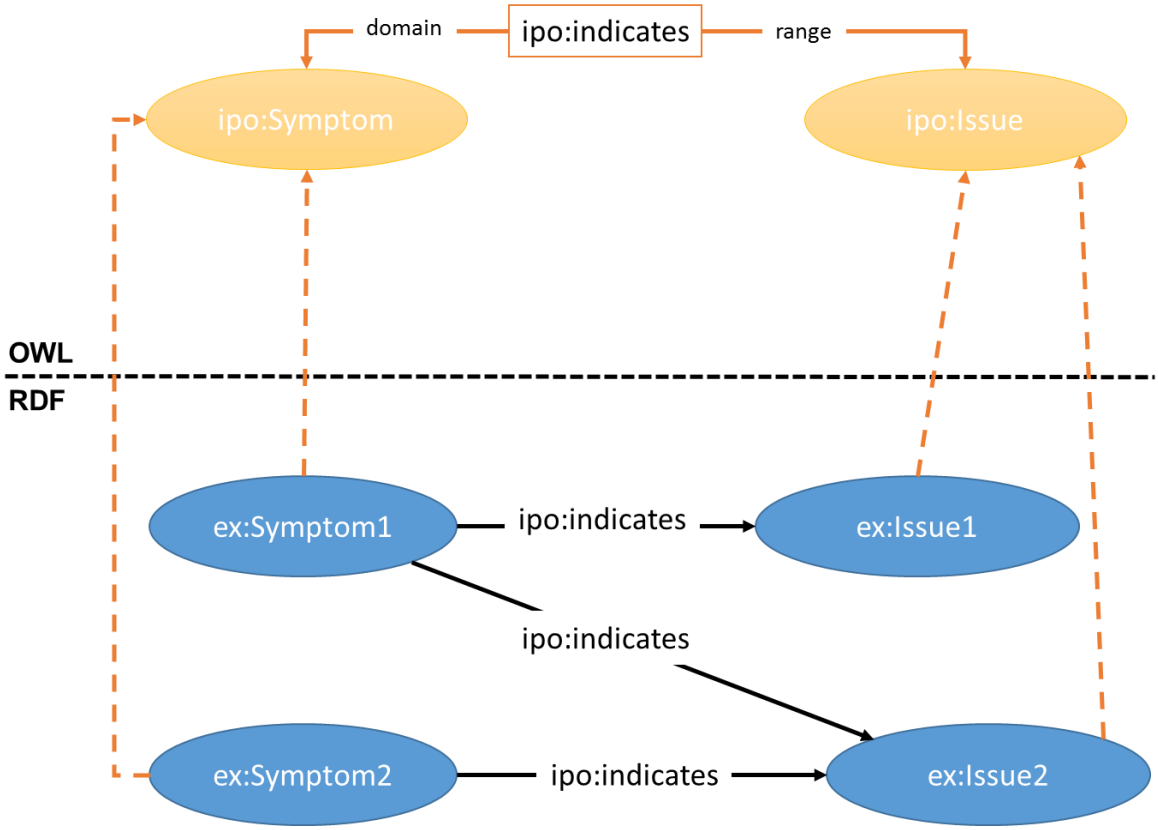

Figura 30 - Exemplo de instanciação usando a propriedade *indicates*
Fonte: O autor

Consulta SPARQL:

```
SELECT ?issue (COUNT(?issue) AS ?symptomQuantity)
WHERE {
    {ex:Symptom1 ipo:indicates ?issue .}
```

```
        UNION
        {ex:Symptom2 ipo:indicates ?issue .}
    }
GROUP BY ?issue
ORDER BY DESC(?symptomQuantity)
```

Fortemente relacionada à questão QC2, esta questão pode ser respondida por meio da propriedade *indicates* que relaciona um *Symptom* com o *Issue* que ele indica. Dado um conjunto de *Symptoms*, é possível obter todos os *Issues indicados* por estes *Symptoms*.

Na consulta acima, é feito a união de todos os *Issues* indicados pelos *Symptoms* do conjunto: `ex:Symptom1` e `ex:Symptom2`.

Para melhorar ainda mais, os *Issues* retornados são ordenados, decrescentemente, pela quantidade de *Symptoms* presentes no referido conjunto, obtendo-se, assim, uma lista de *Issues* ordenados do mais provável para o menos provável. Para tal é usada cláusula `GROUP BY` para agrupar os resultados por problema (`?issue`) e a função agregadora `COUNT()` para contar a quantidade de sintomas por problema (`?symptomQuantity`), ordenando os resultados decrescentemente por esta quantidade.

QC11. Qual(is) classe(s) de um determinado problema a partir dos seus sintomas?

A classe *Issue* tem como objetivo descrever problemas. Para que possamos registrar semanticamente as ocorrências no espaço/tempo de problemas, faz-se necessário criar subclasses da classe *Issue* que representem os problemas e essas subclasses terão como instâncias as ocorrências.

Diante disso, visando permitir a classificação automática de ocorrências de problemas a partir de seus sintomas em classes específicas definidas pelo usuário da ontologia, a título de exemplo de classificação intrínseca em um domínio específico, foi criada uma subclasse da classe *Issue* denominada *Gripe*, definida como equivalente[49] ao conjunto de recursos que apresentam pelo menos um dos sintomas (instâncias da classe *Symptom*): *Febre* ou *Coriza*. Em outras palavras,

[49] Equivalência significa condições necessárias e suficientes para ser instância da classe `Gripe`.

qualquer ocorrência que apresente pelo menos um destes dois sintomas, a máquina, automaticamente, a classificará como sendo uma instância da classe especializada *Gripe*. A Figura 31 demonstra esse exemplo.

As relações na cor verde são obtidas automaticamente quando a máquina classifica a ocorrência como instância da classe Gripe. Perceba que o ser humano apenas precisa informar os sintomas (relacionamentos na cor preta) e a máquina decide, com base na ontologia, quais possíveis doenças se aplica àquela ocorrência.

Sobre a classe *Gripe*:

URI:	`ex:Gripe`
type:	`owl:Class`
equivalentClass:	`(ipo:indicatedBy value`[50] `ex:Febre) or (ipo:indicatedBy value ex:Coriza)`
subClassOf:	`ipo:Issue, ipo:title value "Gripe"^^string, ipo:solvedBy value ex:Tratamento_de_Gripe, ipo:description value "A gripe é uma infecção contagiosa de nariz, garganta e pulmões causada pelo vírus *influenza*."^^string`

[50] `value` significa restrição de indivíduo (owl:hasValue).

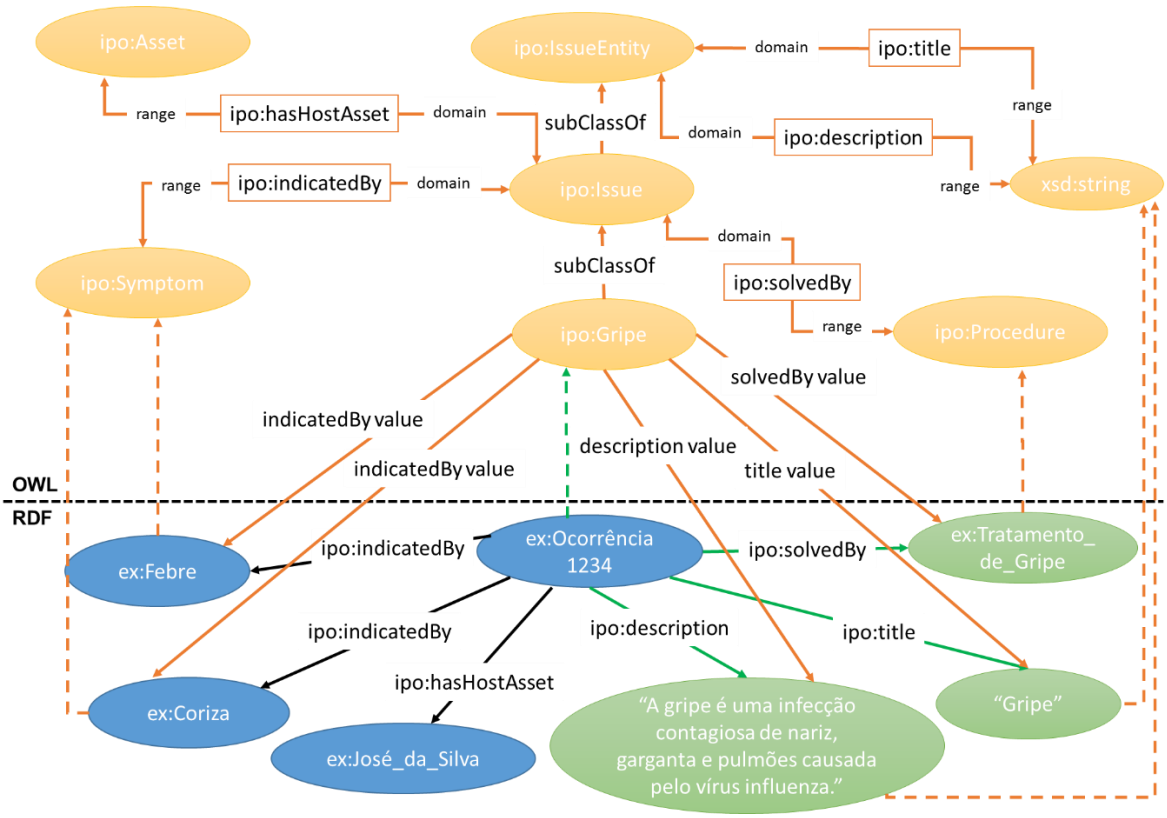

Figura 31 - Exemplo de instanciação com a subclasse Gripe
Fonte: O autor

Consulta SPARQL:

```
SELECT ?ocorrencia_gripe
WHERE {
    ?ocorrencia_gripe rdf:type ex:Gripe .
}
```

A consulta SPARQL tem por objetivo obter os problemas de Gripe (instâncias de `Gripe`). Como se pode ver na instanciação acima, somente está descrito um recurso (`ex:Ocorrência1234`) que possui um relacionamento com os recursos `ex:Febre` e `ex:Coriza` por meio da propriedade `ipo:indicatedBy`. Somente com essa informação, a consulta SPARQL não terá resposta. Porém, como já foi mencionado anteriormente, baseando-se na restrição de "classe equivalente" da classe `ex:Gripe`, novas triplas são inferidas. Assim uma nova tripla será

adicionada automaticamente: `ex:Issue1 rdf:type ex:Gripe`[51] e, portanto, o recurso `ex:Ocorrência1234` será retornado pela consulta.

3.6 ESTUDO DE CASO REALISTA

Para corroborar o poder de expressividade da ontologia IPO, é apresentado um estudo de caso, no qual uma especialização realista da ontologia é delineada. É importante deixar claro que, em nenhum momento, se pensou em desenvolver uma ontologia que pudesse ser aplicada em um sistema semântico real, pois para isso, se faria necessário a presença de um especialista do domínio. Sendo assim, trata-se de um contexto *apenas* realista, que mostra como a ontologia IPO pode viabilizar a classificação de recursos semânticos de forma autônoma pela máquina.

O domínio escolhido foi o domínio de diagnóstico de doenças na medicina, onde se espera que, com a ontologia IPO, a máquina seja capaz de classificar, automaticamente, ocorrências de doenças, a partir de sintomas apresentados pelos pacientes e, em seguida, propor possíveis tratamentos.

Em linhas gerais, em um consultório médico, o paciente reporta sintomas e o médico deve, com base nesses sintomas, identificar a doença correspondente e a partir desse diagnóstico, indicar um tratamento. Porém a quantidade de doenças existentes torna essa tarefa bastante difícil. Com o intuito de ajudar o profissional da saúde em sua tomada de decisão, sistemas computacionais podem ser desenvolvidos para, como base nos sintomas relatados pelo paciente, sugerir possíveis doenças. Esses sistemas são também chamados de sistemas especialistas (ANDRADE, 1999).

Com o uso de ontologias é possível desenvolver sistemas especialistas semânticos, onde a “inteligência” desse sistema ficaria a cargo dos axiomas definidos na ontologia.

Conforme exposto, a ontologia IPO provê uma descrição semântica para a tríade *sintomas, problemas e soluções*. Estes três componentes podem ser diretamente mapeados para o domínio de medicina como sintomas, doenças e tratamentos.

---

[51] Esse relacionamento está representado na figura através da linha tracejada, de cor verde, que liga o recurso RDF `ex:Ocorrência1234` com a classe OWL `ex:Gripe`.

A seguir será detalhada como foi a construção desta extensão da ontologia IPO para medicina, denominada de *Issue Procedure Ontology – Medicine* (IPO-M).

3.6.1 Domínio e Escopo da Ontologia

Como o objetivo é representar o domínio de diagnóstico de doenças, relacionando sintomas e tratamentos, visando um melhor entendimento do escopo e do domínio da ontologia, foram considerados alguns cenários motivacionais, dentre os quais, três são brevemente descritos a seguir. Em seguida, com base nestes cenários, foram elencadas algumas questões de competência, para as quais a ontologia deveria prover as respostas.

Cenário 1: Um paciente chega ao hospital reportando problemas de saúde, o médico o atende e pergunta o que está sentido. O paciente reporta que está com febre e calafrios. O médico analisa o quadro e identifica que o paciente está com Gripe. Como tratamento, o médico informa ao paciente que pode tomar alguns analgésicos e antitérmicos.

Cenário 2: Um paciente chega ao hospital reportando problemas de saúde, o médico o atende e pergunta o que está sentido. O paciente reporta que está com febre, tosse e deficiência respiratória. O médico analisa o quadro e identifica que o paciente está com Pneumonia. Então o médico receita o tratamento com amoxicilina.

Cenário 3: Um paciente chega ao hospital reportando problemas de saúde, o médico o atende e pergunta o que está sentido. O paciente reporta que está com deficiência respiratória e frequência cardíaca acelerada. O médico analisa o quadro e identifica que o paciente está com Hipertensão Pulmonar. Como tratamento, o médico receita o tratamento tradicional com anticoagulante, diuréticos e oxigênio.

Diante dos cenários acima, foram elaboradas algumas questões de competência para orientar a construção da ontologia e servir como uma forma de avaliação futura:

QC1. Para uma determinada doença, quais seus causadores?

QC2. Quais as ocorrências de uma doença atendidas por um determinado médico?

QC3. Quais ocorrências registradas para um determinado paciente?

QC4. Qual a data e hora de uma ocorrência?

QC5. Dados os sintomas, quais são as possíveis doenças?

QC6. Dada a doença, quais seus tratamentos?

Para melhor compreender as doenças citadas nos cenários acima (Gripe, Pneumonia e Hipertensão Pulmonar), foi realizada uma pesquisa para levantar os principais sintomas e tratamentos, para servir como base para a construção da ontologia e a *posteriori* para se fazer testes de validação.

Gripe

Definição: A gripe ou *influenza* é uma infecção contagiosa que ataca o sistema respiratório, como nariz, garganta e pulmões causada pelo vírus influenza.

Sintomas: Febre, Dores do corpo, Calafrios, Dor de cabeça, Falta de energia, Congestão nasal, Tosse seca.

Tratamento: Descanse, Tome medicamentos que aliviam os sintomas e ajudam você a descansar, Beba líquidos em abundância, Evite aspirina (especialmente adolescentes e crianças), Evite álcool e fumo, Evite antibióticos (a menos que seja necessário para outra doença).

Essas informações sobre Gripe foram extraídas de Mayo Clinic Staff (2014) e Portal IG (2011).

Pneumonia

Definição: A pneumonia é uma infecção que inflama os alvéolos em um ou ambos os pulmões. Os alvéolos podem encher de líquido ou pus, causando tosse com catarro ou pus, febre, calafrios e dificuldade em respirar. Uma variedade de organismos, incluindo bactérias, vírus e fungos, podem provocar uma pneumonia.

Sintomas: Febre, Calafrios, Tosse (com muco), Dor no peito, Falta de ar, Fadiga e dores musculares, Náuseas, vómitos ou diarreia, Dor de cabeça.

Tratamento: O algoritmo de tratamento da pneumonia pode ser visto na Figura *32*.

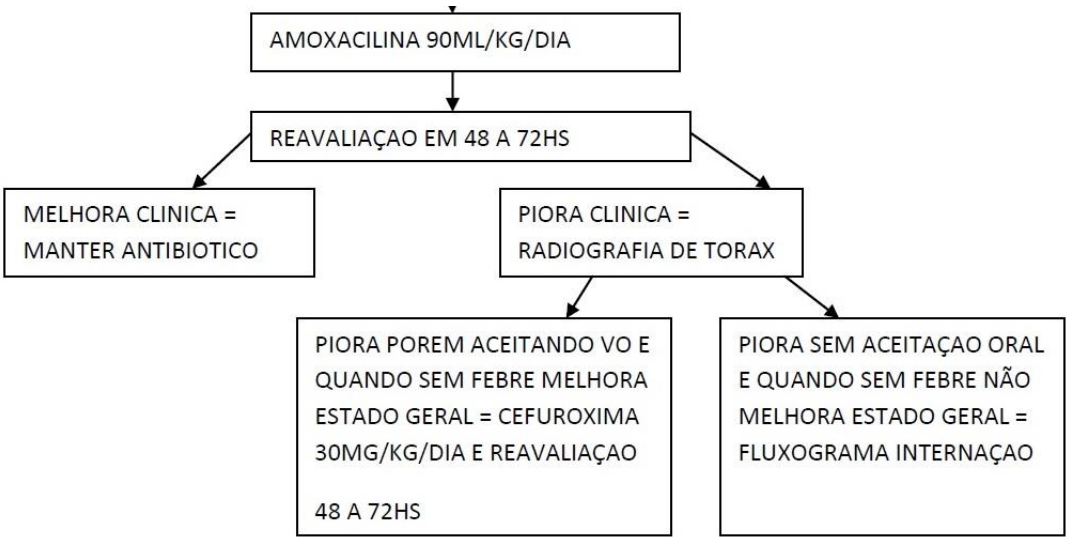

Figura 32 – Algoritmo de Tratamento da Pneumonia
Fonte: Mayo Clinic Staff (2015)

Os dados sobre Pneumonia foram obtidos a partir de Mayo Clinic Staff (2015) e Guimarães e Lopes (2005).

Hipertensão Pulmonar

Definição: A hipertensão pulmonar é um tipo de pressão arterial elevada, que afeta as artérias nos pulmões e no lado direito do seu coração.

Sintomas: Falta de ar, Fadiga, Tonturas, Pressão ou dor no peito, Inchaço em tornozelos, pernas e, eventualmente, no abdome, Cor azulada em seus lábios e pele (cianose), Pulso acelerado ou palpitações cardíacas

Tratamento: O tratamento pode ser visualizado na Figura 33.

Informações sobre Hipertensão Pulmonar foram obtidas em Kayser (2009) e MAYO CLINIC STAFF(2013).

Cabe informar que para as doenças acima, existem outros sintomas (menos comuns) e tratamentos, porém para este estudo foram analisados somente os dados acima, considerados suficientes para o exemplo de uso que se pretende expor neste trabalho.

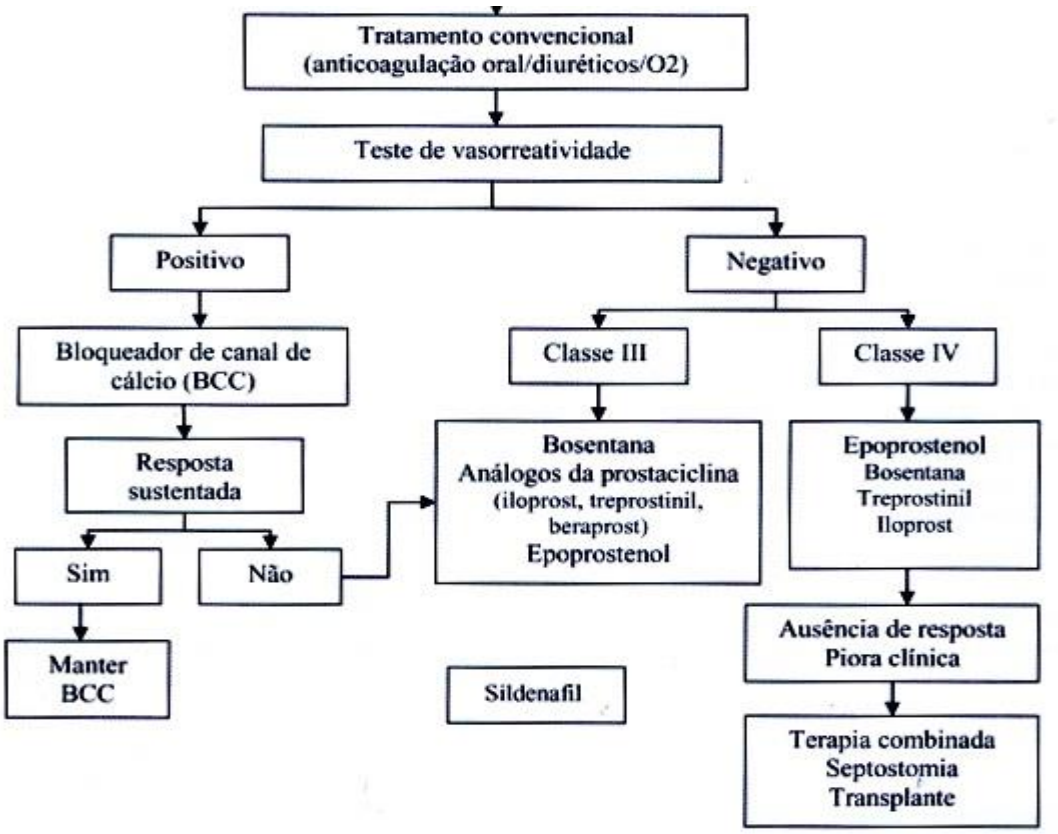

Figura 33 - Algoritmo de tratamento da Hipertensão Pulmonar
Fonte: KAYSER (2009)

3.6.2 Metodologia

A metodologia utilizada para guiar o desenvolvimento da ontologia IPO-M, foi a mesma utilizada para o desenvolvimento da ontologia IPO: *Ontology Development 101*.

O domínio e o escopo foram definidos com a descrição de cenários e com a enumeração de questões de competência, como exposto na seção 3.6.1.

Outra etapa da metodologia é considerar o reuso de outras ontologias já existentes. Para que a ontologia IPO-M tenha condições de descrever ocorrências das doenças, faz-se necessário registrar data e hora dos atendimentos médicos. Portanto, além da ontologia IPO, foram reusadas duas outras ontologias, a saber:

Event Ontology[52]**:** A *Event Ontology* foi desenvolvida no Centro de Música Digital no Queen Mary, Universidade de Londres 2004. Essa ontologia é centrada em torno da noção de evento, visto aqui como a maneira pela qual os agentes

[52] Event – http://motools.sourceforge.net/event/event.html

cognitivos classificam, arbitrariamente, regiões de tempo/espaço. Esta ontologia tem se mostrado bastante útil em uma ampla gama de contextos, devido à sua simplicidade e facilidade de uso: desde de palestras em uma conferência, descrição de um concerto, ou acordes tocados em uma apresentação de Jazz (quando usado com a ontologia *Timeline*), festivais, entre outros (RAIMOND; ABDALLAH, 2007).

A ontologia IPO-M tem a intenção de registrar as ocorrências das doenças que podem ser mapeadas como eventos pela *Event Ontology*. Assim, toda a parte de registro temporal da IPO-M é delegada à *Event Ontology*.

***Time Ontology*[53]:** Informações temporais são tão comuns que é difícil encontrar um serviço *Web* do mundo real sem elas. Por exemplo, sempre que você fizer um pedido on-line, a data do pedido será sempre parte da sua encomenda. Quando se reserva um carro em uma empresa de aluguel de carros, é preciso especificar as datas que se precisará do veículo. Dessa forma, a *Time Ontology* fornece um vocabulário para expressar fatos sobre relações topológicas entre instantes e intervalos de tempo, e sobre informações de data e hora (HOBBS; PAN, 2006).

A ontologia IPO-M não reusa diretamente a *Time Ontology*, porém a *Event Ontology* a utiliza para descrever intervalos de tempo.

3.6.3 *Design* da *Issue Procedure Ontology – Medicine* (IPO-M)

Para a ontologia criada neste exemplo, foi adotado o prefixo `ipo-m` para representar o *namespace* `https://purl.org/ipo/medice#`. Assim, as classes e propriedades pertencentes a esta ontologia são precedidas do prefixo ipo-m.

No Quadro 5 estão relacionadas todas as ontologias reusadas ou utilizadas para descrever a ontologia IPO aplicado ao domínio da medicina (IPO-M), em ordem alfabética, relacionando o prefixo, nome e o *namespace*.

[53] *Time Ontology* – http://www.w3.org/TR/owl-time/

Prefixo	Nome	Namespace
event	The Event Ontology	http://purl.org/NET/c4dm/event.owl#
ipo	Issue Procedure Ontology	https://purl.org/ipo/core#
ipo-m	Issue Procedure Ontology - Medicine	https://purl.org/ipo/core/medicine#
owl	Ontology Web Language	http://www.w3.org/2002/07/owl#
time	Time Ontology	http://www.w3.org/2006/time#
xsd	XML Schema	http://www.w3.org/2001/XMLSchema#

Quadro 5- Ontologias reusadas ou utilizadas na para descrever a IPO-M.
Fonte: O autor

A ontologia IPO foi especializada com a criação de subclasses da classe *Issue* e foi estendida com a criação de novas propriedades. Além disso, as ontologias *Event* e *Time* foram usadas para o tratamento das ocorrências das doenças.

A ontologia IPO adaptada ao domínio da medicina (IPO-M) pode ser vista na Figura 34 através de um diagrama de classes UML.

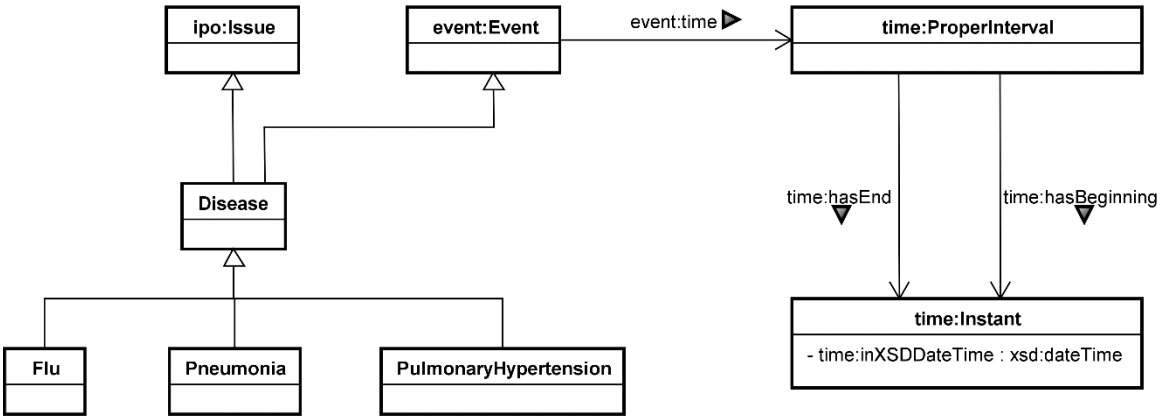

Figura 34- Modelo conceitual da IPO-M.
Fonte: O autor

A seguir serão descritas as classes (instâncias da metaclasse *owl:Class*) e as propriedades (instância das metaclasses *owl:ObjectProperty* ou

owl:DatatypeProperty) da ontologia IPO-M. Para tal, objetivando uma melhor compreensão, a ontologia foi dividida em dois módulos:

- **Descrição das ocorrências**
  Classes: *event:Event, time:ProperInterval e time:Instant.*
  Propriedades: *event:time, time:hasBeginning, time:hasEnd e time:inXSDDateTime*.
- **Descrição das doenças**
  Classes: *Disease, Flu, Pneumonia e PulmonaryHipertension*.

3.6.3.1 Descrição das Ocorrências

As classes e propriedades desse módulo têm como objetivo prover meios para ontologia IPO-M registrar as ocorrências de doenças como eventos ao declarar a classe *event:Event* como superclasse da classe *Disease.*

Classe: *event:Event*

Uma classificação arbitrária de uma região do espaço / tempo, por um agente cognitivo. Um evento pode ter agentes que participam ativamente, fatores passivos, produtos gerados e uma localização no espaço/tempo.

Esta classe é usada para descrever uma ocorrência de uma doença, ou seja, um atendimento. Através dessa classe, podemos registrar a data e hora que ocorreu o atendimento do paciente. Isso permite que posteriormente se consiga obter informações das doenças mais diagnosticadas, detectar epidemias, entre outros dados importantes.

Sobre a classe *event:Event*:

URI:	`event:Event`
type:	`owl:Class`

Classe: *time:ProperInterval*

A classe *time:ProperInterval* representa um intervalo de tempo, usado para demarcar o início e o fim de um evento. Desta forma, esta classe é usada para registrar o início e o fim do atendimento de um paciente.

Sobre a classe *time:ProperInterval*:

URI:	`time:ProperInterval`
type:	`owl:Class`
subClassOf:	`time:Interval`

Classe: *time:Instant*

Um instante de tempo – um ponto (data e hora) no eixo do tempo. Esta classe é utilizada para representar os instantes nos quais o atendimento é iniciado e finalizado, respectivamente.

Sobre a classe *time:Instant*:

URI:	`time:Instant`
type:	`owl:Class`
subClassOf:	`time:TemporalEntity`

Propriedade: *event:time*

Relaciona um evento com um objeto de tempo, classificando a região de tempo. Neste trabalho, essa propriedade associa a ocorrência de uma doença (*event:Event*) com um intervalo de tempo (*time:ProperInterval*).

Sobre a propriedade *event:time*:

URI:	`event:time`
type:	`owl:ObjectProperty`
domain:	`event:Event`
range:	`time:TemporalEntity`

Propriedade: *time:hasBeginning*

Indica o instante inicial de um intervalo de tempo. Essa propriedade é usada para indicar o instante inicial do atendimento médico.

Sobre a propriedade *time:hasBeginning*:

URI:	`time:hasBeginning`

type:	`owl:ObjectProperty`
domain:	`time:TemporalEntity`
range:	`time:Instant`

Propriedade: *time:hasEnd*

Indica o instante final de um intervalo de tempo. Essa propriedade é usada para indicar o instante final do atendimento médico.

Sobre a propriedade *time:hasEnd*:

URI:	`time:hasEnd`
type:	`owl:ObjectProperty`
domain:	`time:TemporalEntity`
range:	`time:Instant`

Propriedade: *time:inXSDDateTime*

Indica a data e hora do instante.

Sobre a propriedade *time:inXSDDateTime*:

URI:	`time:inXSDDateTime`
type:	`owl:DatatypeProperty`
domain:	`time:Instant`
range:	`xsd:dateTime`

3.6.3.2 Descrição das Doenças

Classe: *Disease*

Uma classe genérica que representa todos os tipos de doença. Assim, todas as restrições comuns a todas as doenças podem ser definidas nessa classe, pois todas as outras classes de doenças específicas serão subclasses desta.

Essa classe é subclasse de *ipo:Issue* e de *event:Event*, definindo que todas as ocorrências (instâncias) de uma classe de doença são, ao mesmo tempo, tratadas como um problema pela ontologia IPO e como um evento pela ontologia *Event*.

Sobre a classe *Disease*:

URI:	`ipo-m:Disease`
type:	`owl:Class`
subClassOf:	`ipo:Issue e event:Event`

Como será exemplificado em seguida, para que máquina consiga fornecer o diagnóstico automaticamente, ou seja, classificar uma ocorrência como membro de uma subclasse `Doença-X` de `Disease`, o projetista da ontologia deve adotar as seguintes diretrizes:

1. Criar a classe `Doença-X`;
2. Relacionar a classe `Doença-X` com a classe `Disease`, por meio do axioma `rdfs:subClassOf`;
3. Para cada sintoma da `Doença-X`, criar uma instância da classe `ipo:Symptom`, descrevendo suas propriedades (título, descrição).
4. Para cada tratamento da `Doença-X`, criar uma instância da classe `ipo:Procedure`, detalhando o *workflow* (passos/ações e transições) correspondente.
5. Na classe `Doença-X`, definir o conjunto de condições necessárias e suficientes para ser classificado na mesma. Para tal, relacioná-la por meio de `owl:equivalentClass` com os sintomas anteriores, usando a propriedade `ipo:indicatedBy` com a restrição `owl:hasValue` e operadores lógicos;
6. Na classe `Doença-X`, descrever as características (condições necessárias) da `Doença-X` (título, descrição, agente causador, tratamento, etc.), usando, para cada característica, `rdfs:subclassOf` e a restrição `owl:hasValue` na propriedade correspondente.

Para não sobrecarregar os exemplos de classes de doença a seguir, é assumido que os sintomas e tratamentos já foram devidamente definidos como instâncias diretas de `ipo:Symptom` e `ipo:Procedure`, respectivamente.

Classe: *Flu*

Essa classe representa a doença Gripe Pretende-se que toda ocorrência desta doença seja classificada como uma instância desta classe.

Sobre a classe *Flu*:

URI:	`http://purl.org/ipo/medicine#Flu`
type:	`owl:Class`
equivalentClass:	`((ipo:indicatedBy value ipo-m:Sintoma_Calafrios) or (ipo:indicatedBy value ipo-m:Sintoma_Dor_de_Cabeca) or (ipo:indicatedBy value ipo-m:Sintoma_Dores_do_Corpo) or (ipo:indicatedBy value ipo-m:Sintoma_Falta_de_Energia) or (ipo:indicatedBy value ipo-m:Sintoma_Congestao_Nasal) or (ipo:indicatedBy value ipo-m:Sintoma_Tosse_Seca)) and (ipo:indicatedBy value ipo-m:Sintoma_Febre)`
subClassOf:	`ipo-m:Disease, ipo:solvedBy value ipo-m:Tratamento_Gripe, ipo:hasCausativeAsset value ipo-m:influenza,`

	ipo:title value "Gripe"^^string, ipo:description value "A gripe ou *influenza* é uma infecção contagiosa que ataca o sistema respiratório, como nariz, garganta e pulmões causada pelo vírus *influenza*."^^string

Classe: *Pneumonia*

Essa classe representa a doença Pneumonia. Pretende-se que toda ocorrência desta doença seja classificada como uma instância desta classe.

Sobre a classe *Pneumonia*:

URI:	ipo-m:Pneumonia
type:	owl:Class
equivalentClass:	((ipo:indicatedBy value ipo-m:Sintoma_Febre) and (ipo:indicatedBy value ipo-m:Sintoma_Tosse)) and ((ipo:indicatedBy value ipo-m:Sintoma_Deficiencia_Respiratoria) or (ipo:indicatedBy value ipo-m:Sintoma_Calafrios) or (ipo:indicatedBy value ipo-m:Sintoma_Dor_no_Peito) or (ipo:indicatedBy value ipo-m:Sintoma_Fadiga) or

	(ipo:indicatedBy value ipo-m:Sintoma_Nausea) or (ipo:indicatedBy value ipo-m:Sintoma_Vomito) or (ipo:indicatedBy value ipo-m:Sintoma_Dor_de_Cabeca))
subClassOf:	ipo-m:Disease, ipo:solvedBy value ipo-m:Tratamento_Pneumonia, ipo:solvedBy value ipo-m:Tratamento_Pneumonia_Internacao, ipo:hasCausativeAsset value ipo-m:Streptococcus_pneumoniae, ipo:title value "Pneumonia"^^string, ipo:description value "A pneumonia é uma infecção que inflama os alvéolos em um ou ambos os pulmões. Os alvéolos pode encher de líquido ou pus, causando tosse com catarro ou pus, febre, calafrios e dificuldade em respirar. Uma variedade de organismos, incluindo bactérias, vírus e fungos, podem provocar uma pneumonia."^^string

Classe: *PulmonaryHypertension*

Essa classe representa a doença Hipertensão Pulmonar, pretende-se que toda ocorrência desta doença seja classificada como uma instância desta classe.

Sobre a classe *PulmonaryHypertension*:

URI:	ipo-m:PulmonaryHypertension
type:	owl:Class
equivalentClass:	(ipo:indicatedBy value ipo-m:Sintoma_Deficiencia_Respiratoria) and (ipo:indicatedBy value ipo-m:Sintoma_Freq_Cardiaca_Acelerada) and ((ipo:indicatedBy value ipo-m:Sintoma_Fadiga) or (ipo:indicatedBy value ipo-m:Sintoma_Tontura) or ((ipo:indicatedBy value ipo-m:Sintoma_Dor_no_Peito) or (ipo:indicatedBy value ipo-m:Sintoma_Inchaco) or (ipo:indicatedBy value ipo-m:Sintoma_Cianose))
subClassOf:	ipo-m:Disease, ipo:solvedBy value ipo-m:Tratamento_HP, ipo:title value "Hipertensão Pulmonar"^^string, ipo:description value " A hipertensão pulmonar é um tipo de pressão arterial elevada, que afeta as artérias nos pulmões e no lado direito do seu coração."^^string

3.6.4 Representação dos Tratamentos

Como pode ser visto na Figura 32 e na Figura 33, os tratamentos podem ser representados através de algoritmos ou fluxogramas. A ontologia IPO fornece meios para representar, semanticamente, fluxogramas através da classe `ipo:Procedure`.

Um `ipo:Procedure` possui um objetivo (`ipo:goal`) a ser alcançado no final de sua execução. Além disso, um `ipo:Procedure` possui (`ipo:hasStep`) uma ou mais etapas, ou passos, a serem executadas (`ipo:Step`), cada `ipo:Step` ativa (`ipo:activates`) uma ação (`ipo:Action`) que deve ser realizada.

Uma `ipo:Action` é uma superclasse de `ipo:Task`, que representa uma tarefa simples e indivisível, e de `ipo:CompoundAction`, que representa uma ação composta por diversas outras ações, simples ou composta.

Ao realizar a `ipo:Action` de uma determinada `ipo:Step`, deve-se passar para próxima `ipo:Step`, ocorrendo assim, uma transição entre etapas (`ipo:Transition`). Cada `ipo:Transition`, porém, possui uma condição de guarda (`ipo:hasGuardCondition`) que consiste em uma expressão lógica (`ipo:BooleanExpression`) que deve ser atendida para que a `ipo:Transition` possa ocorrer. Essas condições de guarda possibilitam que estruturas mais complexas de fluxograma, como condicionais e repetições, sejam construídas.

Para exemplificar uma instanciação de um fluxograma, a Figura 35 traz o algoritmo de tratamento de pneumonia, exposto na Figura 32, instanciado com a ontologia IPO. É importante notar que o `ipo-m:Step5` ativa um novo procedimento, ou seja, um `ipo:Procedure` pode fazer uso de outro `ipo:Procedure`, o que é bastante útil para o reuso de procedimentos ou tratamentos.

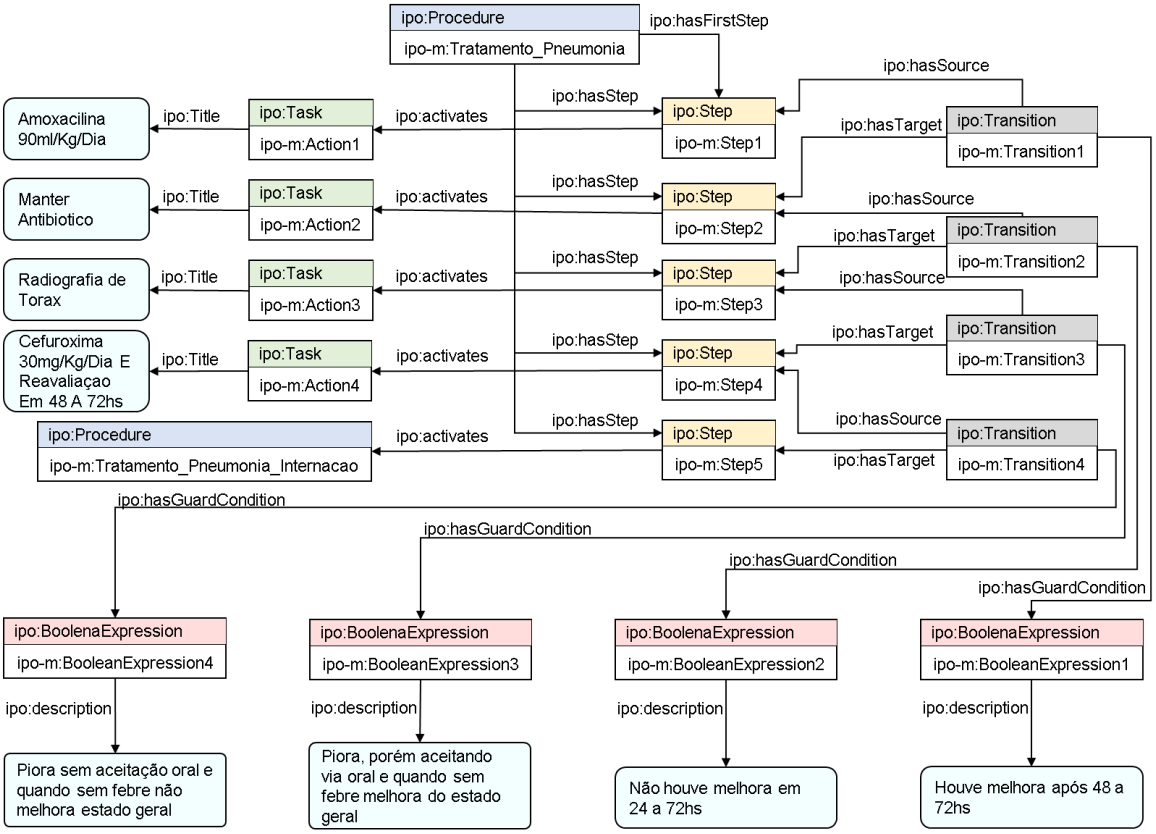

Figura 35 - Instanciação do algoritmo de tratamento da Pneumonia
Fonte: O autor

3.6.5 Avaliação da *Issue Procedure Ontology – Medicine*

A avaliação é uma etapa importante no processo de criação de uma ontologia. Para realizar essa validação, se faz necessário verificar se a ontologia é expressiva o suficiente para prover respostas para as questões de competência levantadas anteriormente e verificar se ela consegue representar todos os dados pertinentes ao domínio.

A seguir, para cada cenário abordado acima, foi efetuada uma instanciação, em Turtle, no formato básico em triplas, para representá-los. Os prefixos utilizados nesta seção são:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ipo: <https://purl.org/ipo/core#>.
@prefixe ipo-m: <https://purl.org/ipo/core/medicine#>
@prefix event: <http://purl.org/NET/c4dm/event.owl#>.
@prefix time: <http://www.w3.org/2006/time#>.
```

Cenário 1

```
ipo-m:Ocorrencia1  ipo:indicatedBy ipo-m:Sintoma_Calafrios.
ipo-m:Ocorrencia1  ipo:indicatedBy ipo-m:Sintoma_Febre.
ipo-m:Ocorrencia1  ipo:hasMaker ipo-m:Medico1.
ipo-m:Ocorrencia1  ipo:hasHostAsset ipo-m:Paciente1.
ipo-m:Ocorrencia1  event:time ipo-m:ProperInterval1.

ipo-m:Medico1 ipo:name “José de Oliveira”^^string.
ipo-m:Paciente1 ipo:name “Carlos da Silva”^^string.

ipo-m:ProperInterval1 time:hasBeginning ipo-m:Instant1
ipo-m:ProperInterval1 time:hasEnd ipo-m:Instant2
ipo-m:Instant1 time:inXSDDateTime “2015-01-
11T12:30:23”^^datetime
ipo-m:Instant1 time:inXSDDateTime “2015-01-
11T13:15:22”^^datetime
```

Cenário 2

```
ipo-m:Ocorrencia2  ipo:indicatedBy ipo-m:Sintoma_Tosse.
ipo-m:Ocorrencia2  ipo:indicatedBy ipo-
m:Sintoma_Deficiencia_Repiratoria.
ipo-m:Ocorrencia2  ipo:indicatedBy ipo-m:Sintoma_Febre.
ipo-m:Ocorrencia2  ipo:hasMaker ipo-m:Medico1.
ipo-m:Ocorrencia2  ipo:hasHostAsset ipo-m:Paciente2.
ipo-m:Ocorrencia2  event:time ipo-m:ProperInterval2.

ipo-m:Paciente1 ipo:name “José Viana”^^string.

ipo-m:ProperInterval2 time:hasBeginning ipo-m:Instant3
ipo-m:ProperInterval2 time:hasEnd ipo-m:Instant4
ipo-m:Instant3 time:inXSDDateTime “2015-01-
12T08:40:13”^^datetime
```

```
ipo-m:Instant4 time:inXSDDateTime “2015-01-
12T09:16:52”^^datetime
```

Cenário 3

```
ipo-m:Ocorrencia3  ipo:indicatedBy ipo-
m:Sintoma_Dor_no_Peito.
ipo-m:Ocorrencia3  ipo:indicatedBy
ipo-m:Sintoma_Freq_Cardiaca_Acelerada.
ipo-m:Ocorrencia3  ipo:indicatedBy ipo-m:Sintoma_
Deficiencia_Respiratoria.
ipo-m:Ocorrencia3  ipo:hasMaker ipo-m:Medico2.
ipo-m:Ocorrencia3  ipo:hasHostAsset ipo-m:Paciente2.
ipo-m:Ocorrencia3  event:time ipo-m:ProperInterval3.

ipo-m:Medico2 ipo:name “Lucas dos Santos”^^string.
ipo-m:Paciente2 ipo:name “José Viana”^^string.

ipo-m:ProperInterval3 time:hasBeginning ipo-m:Instant5
ipo-m:ProperInterval3 time:hasEnd ipo-m:Instant6
ipo-m:Instant5 time:inXSDDateTime “2015-01-
18T20:10:43”^^datetime
ipo-m:Instant6 time:inXSDDateTime “2015-01-
18T21:45:32”^^datetime
```

Considerando a instanciação acima, para cada questão de competência, será feita uma consulta SPARQL, objetivando obter a resposta e assim verificar a expressividade da ontologia.

QC1. Para uma determinada doença, quais seus causadores?

Consulta SPARQL:

```
SELECT ?causador
WHERE {
    ipo-m:Ocorrencia2 ipo:hasCausativeAsset ?causador.
}
```

A consulta acima, aplicada à instanciação dos cenários, a princípio, não retornaria nenhum resultado, pois a ocorrência `ipo-m:Ocorrencia2` não está explicitamente relacionada com nenhum `ipo:Asset` por meio da propriedade `ipo:hasCausativeAsset`. Contudo, ao se aplicar um raciocinador para inferir novas triplas a partir das declarações semânticas contidas na ontologia IPO-M, a ocorrência `ipo-m:Ocorrencia2` será classificada como *Pneumonia*, pois a classe `ipo-m:Pneumonia` possui como condições necessárias e suficientes (ou classe equivalente) a seguinte restrição:

```
((ipo:indicatedBy value ipo-m:Sintoma_Febre)
and (ipo:indicatedBy value ipo-m:Sintoma_Tosse))
and ((ipo:indicatedBy value
ipo-m:Sintoma_Deficiencia_Respiratoria)
or (ipo:indicatedBy value ipo-m:Sintoma_Calafrios)
or (ipo:indicatedBy value ipo-m:Sintoma_Dor_no_Peito)
or (ipo:indicatedBy value ipo-m:Sintoma_Fadiga)
or (ipo:indicatedBy value ipo-m:Sintoma_Nausea)
or (ipo:indicatedBy value ipo-m:Sintoma_Vomito)
or (ipo:indicatedBy value ipo-m:Sintoma_Dor_de_Cabeca))
```

Assim, como a classe `ipo-m:Pneumonia` é subclasse de uma classe anônima formada pela restrição `ipo:hasCausativeAsset value ipo-m:Streptococcus_pneumoniae`, o resultado da consulta será o vírus `ipo-m:Streptococcus_pneumoniae`.

QC2. Quais as ocorrências de uma doença atendidas por um determinado médico?

Consulta SPARQL:

```
SELECT ?ocorrencia
WHERE {
    ipo-m:Medico1 ipo:makerOf ?ocorrencia.
    ?ocorrencia rdf:type ipo-m:Pneumonia.
}
```

Sem o uso de uma raciocinador não haveria resposta para essa questão: (i) não hás triplas com a propriedade `ipo:makerOf` e (ii) não há nenhuma

classificação das doenças. Porém, ao se fazer uso do raciocinador, a ocorrência ipo-m:Ocorrencia2 será classificada como *Pneumonia* e essa mesma ocorrência tem um relacionamento com o médico ipo-m:Medico1 através da propriedade ipo:hasMaker que é inversa da propriedade ipo:makerOf. Assim, a resposta para que essa consulta será a ocorrência ipo-m:Ocorrencia2. Vale ressaltar que o médico ipo-m:Medico1 atendeu também a ocorrência ipo-m:Ocorrencia1, porém está não é classificada como *Pneumonia*.

QC3. Quais ocorrências registradas para um determinado paciente?

Consulta SPARQL:

```
SELECT ?ocorrencia
WHERE {
    ipo-m:Paciente2 ipo:hostAssetOf ?ocorrencia.
}
```

Nessa consulta se deseja saber quais a ocorrências registradas para o paciente ipo-m:Paciente2. Este paciente está associado às ocorrências ipo-m:Ocorrencia2 e ipo-m:Ocorrencia3 pela propriedade ipo:hasHostAsset. Como a propriedade ipo:hasHostAsset possui como propriedade inversa a ipo:hostAssetOf, com o uso do raciocinador, a resposta para esta consulta serão as ocorrências ipo-m:Ocorrencia2 e ipo-m:Ocorrencia3.

QC4. Qual a data e hora de uma ocorrência?

Consulta SPARQL:

```
SELECT ?inicio, ?fim
WHERE {
    ipo-m:Ocorrencia1  event:time ?properInterval.
    ?properInterval time:hasBeginning ?instant1.
    ?properInterval time:hasEnd ?instant2.
    ?instant1 time:inXSDDateTime ?inicio.
    ?instant2 time:inXSDDateTime ?fim.
}
```

Para se obter um resultado não há necessidade do uso de um raciocinador, os relacionamentos já estão declarados na instanciação, o resultado obtido nessa consulta é a data de início e fim da ocorrência `ipo-m:Ocorrencia1`:

Início: `"2015-01-11T12:30:23"^^datetime`

Fim: `"2015-01-11T13:15:22"^^datetime`

QC5. Dados os sintomas, quais são as possíveis doenças?

Consulta SPARQL:

```
SELECT ?doenca
WHERE {
    ipo-m:Ocorrencia1 rdf:type ?doenca.
}
```

Como podemos ver na instanciação, a ocorrência `ipo-m:Ocorrencia1` possui alguns sintomas relacionados pela propriedade `ipo:indicatedBy`, os sintomas são `ipo-m:Sintoma_Calafrios` e `ipo-m:Sintoma_Febre`. Até o momento, não se sabe qual a doença, porém ao ativar um raciocinador, podemos perceber que a ocorrência `ipo-m:Ocorrencia1` será classificada automaticamente como Gripe (`ipo-m:Flu`), isso se deve ao fato da classe `ipo-m:Flu` ser equivalente à uma classe anônima descrita por esta restrição:

```
((ipo:indicatedBy value ipo-m:Sintoma_Calafrios)
or (ipo:indicatedBy value ipo-m:Sintoma_Dor_de_Cabeca)
or (ipo:indicatedBy value ipo-m:Sintoma_Dores_do_Corpo)
or (ipo:indicatedBy value ipo-m:Sintoma_Falta_de_Energia)
or (ipo:indicatedBy value ipo-m:Sintoma_Congestao_Nasal)
or (ipo:indicatedBy value ipo-m:Sintoma_Tosse_Seca))
and (ipo:indicatedBy value ipo-m:Sintoma_Febre)
```

Essa restrição indica que se alguma “coisa” (uma ocorrência) é indicada pelo sintoma de febre e por mais algum sintoma de gripe, então essa “coisa” é uma Gripe.

Diante disso, a resposta para essa consulta será a classe `ipo-m:Flu`, indicando a doença da ocorrência `ipo-m:Ocorrencia1`.

Nada impede de uma ocorrência ser classificada em mais de uma classe, pois o intento desta ontologia é sugerir possíveis doenças para um determinado quadro de sintomas, auxiliando, assim, o médico na sua tomada de decisão.

QC6. Dada a doença, quais seus tratamentos?

Consulta SPARQL:

```
SELECT ?tratamento
WHERE {
    ipo-m:Ocorrencia3  ipo:solvedBy ?tratamento.
}
```

A ocorrência `ipo-m:Ocorrência3` possui como sintomas `ipo-m:Sintoma_Deficiencia_Respiratoria` e `ipo-m:Sintoma_Freq_Cardiaca_Acelerada` indicado pela propriedade `ipo:indicatdBy`. Com o uso do raciocinador, essa ocorrência será classificada como *Hipertensão Pulmonar*, pois a classe `ipo-m:PulmonaryHypertension` possui como condições necessárias e suficientes a seguinte restrição:

```
(ipo:indicatedBy value
ipo-m:Sintoma_Deficiencia_Respiratoria)
and (ipo:indicatedBy value
ipo-m:Sintoma_Freq_Cardiaca_Acelerada)
and ((ipo:indicatedBy value ipo-m:Sintoma_Fadiga)
or (ipo:indicatedBy value ipo-m:Sintoma_Tontura)
or ((ipo:indicatedBy value ipo-m:Sintoma_Dor_no_Peito)
or (ipo:indicatedBy value ipo-m:Sintoma_Inchaco)
or (ipo:indicatedBy value ipo-m:Sintoma_Cianose))
```

Por essa ocorrência ser classificada como `ipo-m:PulmonaryHypertension`, e a classe `ipo-m:PulmonaryHypertension` ser subclasse de uma classe anônima formada pela restrição `ipo:solvedBy value ipo-m:Tratamento_HP`, a seguinte tripla é será inferida: `ipo-m:Ocorrencia3 ipo:solvedBy ipo-m:Tratamento_HP`. Assim, como resposta a essa consulta, será retornado o tratamento `ipo-m:Tratamento_HP`.

4 TRABALHOS RELACIONADOS

Durante a revisão da literatura e nas pesquisas realizadas por ontologias que se propusessem a descrever o domínio do conhecimento proposto nessa dissertação, algumas ontologias se destacaram. São elas:

DOLCE+DnS Ultralite Ontology (DUL)[54]: A *DUL Ontology* é uma combinação da ontologia DOLCE (*Descriptive Ontology for Linguistic and Cognitive Engineering*) e DOLCE DnS (*Descriptions and Situations*) em uma versão mais simples e intuitiva.

GANGEMI *et al*. (2002) ressalta que a DOLCE é uma ontologia de particulares. A distinção ontológica fundamental entre universais e particulares pode ser compreendida tomando em relação a questão da instanciação, onde particulares são entidades que não possuem instâncias e universais possuem.

Diversas extensões da ontologia DUL estão sendo projetadas, dentre elas a PlansLite[55] que aborda os principais conceitos para descrição de *workflows*.

A descrição de *workflow*, abordado pela PlansLite, possui um alto nível de detalhamento e, devido a isso, se apresenta bastante complexa.

P-PLAN Ontology (P-PLAN)[56]: A ontologia P-PLAN tem como objetivo descrever *workflows* científicos, onde se deseja fortemente relacionar seus passos com entidades (produtos, objetos, documentos, etc.) produzidas ou utilizadas pelas etapas deste *workflow* (GARIJO; GIL, 2012).

[54] http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite
[55] http://www.ontologydesignpatterns.org/ont/dul/PlansLite.owl
[56] http://www.opmw.org/model/p-plan/

Publishing Workflow Ontology (PWO)[57]: A ontologia PWO é uma ontologia simples escrita em OWL 2 DL para a caracterização das principais etapas de *workflow* associadas com a publicação de um documento (por exemplo, um documento que está sendo escrito ou sob revisão, captura de XML, design de páginas, a publicação na Web).

A ontologia PWO permite criar *workflows* como uma sequência de passos que possuem um período de duração para ocorrer.

The Wfdesc Ontology (WFDESC)[58]: Esta ontologia ("wfdesc") descreve uma estrutura abstrata para *workflows*, permitindo a descrição de um *workflow* científico como um grafo acíclico direto, ou um *dataflow*. Essa ontologia se apresenta como uma ontologia de topo para *workflow*s específicos. A WFDESC faz parte do projeto WF4Ever[59], que desenvolveu modelos e infra-estrutura com a finalidade de preservação de *workflow*s científicos e seu material de apoio. Esse projeto ainda mantém a ontologia WFPROV[60], onde se permite vincular essas descrições de *workflow* a um rastro de proveniência (BELHAJJAME *et al*., 2012).

HelpDesk Support Ontology (HDO)[61]: A ontologia HDO se propõe a registrar demandas de suporte de TI (*ItSupportTicket*) relacionando-as com as ações que foram executas para solucionar essas demandas (*ItSupportTask*). Para tal, a ontologia HDO faz uso de outras ontologias, como: W3C ORG[62] e REGORG[63] para representação de organizações e unidades de negócios, b) DUL *ontology* para descrição das ações realizadas, e c) *GoodRelations[64] ontology* para descrever os ativos relacionados nessas demandas.

Como pode ser verificado nos trabalhos mencionados nessa seção, existem várias ontologias para descrição de *workflows* e ontologias para resolução de problemas específicos. Não obstante, não foi encontrada uma ontologia que réuna a tríade sintomas-problemas-soluções, unindo problemas e *workflow*s (para

[57] http://www.essepuntato.it/lode/http://purl.org/spar/pwo
[58] http://purl.org/wf4ever/wfdesc
[59] http://www.wf4ever-project.org/
[60] http://wf4ever.github.io/ro/#wfprov
[61] http://www.samos.gr/ontologies/helpdeskOnto/index.html
[62] http://www.w3.org/TR/vocab-org/
[63] http://www.w3.org/TR/vocab-regorg/
[64] http://www.heppnetz.de/projects/goodrelations/

representação das soluções), que possa ser reusada e/ou estendida (*core ontology*) para quaisquer tipos de problemas.

5 CONCLUSÃO

Esse trabalho teve por objetivo desenvolver uma ontologia OWL extensível (*core ontology*) que representasse, de forma genérica, o domínio de sintomas, problemas e soluções. Uma ontologia que viabilizasse a criação de sistemas semânticos para subsidiar o processo de tomada de decisão, no qual a partir de sintomas, agentes de software, com base na ontologia, pudessem identificar os problemas e sugerir possíveis soluções de forma autônoma.

Para tal, foi elaborado um estudo bibliográfico em torno da *Web* Semântica, com ênfase em ontologias, de modo a permitir um maior embasamento para o desenvolvimento do trabalho. Foi elucidado como, em diversos contextos, são tratados os problemas e suas relações com sintomas, bem como as representações das soluções.

Com base nesse estudo, seguiu-se para o desenvolvimento da ontologia em si. Esta foi denominada de *Issue Procedure Ontology* (IPO) e compreende 16 classes, 50 propriedades e 2 duas instâncias, além de diversas assertivas lógicas que dão ao agente de software um expressivo poder de raciocínio.

Para validação e avaliação da ontologia IPO, foram elaborados exemplos de instanciação, bem como consultas SPARQL, visando responder as questões de competência elencadas no início de seu desenvolvimento. Foi demonstrado, então, que a ontologia provê a semântica necessária para que todas as questões de competência sejam respondidas e permite ao computador raciocinar e inferir novos conhecimentos.

Por fim, foi elaborado um estudo de caso realista, onde foi demonstrado como especializar a ontologia IPO para um domínio de problemas específico. Assim, a IPO foi especializada para o domínio de diagnóstico de doenças médicas,

seguindo os mesmos passos metodológicos. Com esse estudo, foi possível ilustrar as diretrizes de como a ontologia IPO pode ser especializada, de modo a prover a base central de conhecimento para sistemas semânticos especialistas, onde, dentre outras facetas, foi claramente demonstrada a classificação automática de ocorrências em tipos de doença a partir de um conjunto de sintomas, bem com posterior sugestão de possíveis tratamentos. Tudo isso de forma autônoma, sem intervenção humana. O único trabalho humano foi informar os sintomas.

5.1 CONTRIBUIÇÕES

A principal contribuição desse trabalho é, obviamente, a ontologia IPO desenvolvida. Esta ontologia já se encontra disponível na *Web* para uso, sob o URI: *http://purl.org/ipo/core*. Para isso, foi implementado negociação de conteúdo HTTP, onde o documento retornado ao acessar esse URI é diferente de acordo com a requisição feita, ou seja, uma pessoa ao acessar esse URI, via navegador, obtém uma página HTML descrevendo a ontologia, porém caso a máquina requisite esse mesmo URI solicitando um arquivo RDF, este será retornado. Assim, em um mesmo URI, podem ser retornadas duas representações da ontologia - em HTML ou em RDF - dependendo da requisição HTTP submetida.

É importante salientar ainda que a ontologia IPO foi indexada no catálogo *on-line* de ontologias *Linked Data* LOV, que é atualmente um dos principais indexadores de ontologias existentes, onde se pode encontrar as principais ontologias para reuso. Isso facilita o acesso à ontologia IPO por parte de pesquisadores e desenvolvedores que trabalham com *Web* Semântica.

Outra contribuição importante é o estudo de caso realista realizado nessa dissertação, pois a partir dele, foi possível demonstrar, sob o ponto de vista prático, o uso da ontologia, adaptando-a para um domínio específico. Por ser uma *core ontology*, a IPO pode ser utilizada como está ou especializada para um domínio mais restrito, aumentando, assim, seu poder de expressividade.

5.2 TRABALHOS FUTUROS

Como trabalho futuro seria interessante desenvolver um estudo de caso real, onde a ontologia IPO pudesse ser utilizada por um período considerável de tempo para que seja paulatinamente ajustada às necessidades dos usuários. Um processo contínuo de aperfeiçoamento peculiar à construção de ontologias.

Poderia ser desenvolvido também um sistema semântico configurável, onde o especialista poderia configurá-lo para que atendesse as suas necessidades. Nesse sistema a ontologia IPO seria automaticamente especializada pelo sistema de acordo com as configurações estabelecida pelo especialista. Assim, um único sistema serviria para diversos domínios, ou seja, um sistema altamente adaptável, tal como a ontologia IPO.

6 REFERÊNCIAS

ABBURU, S. A Survey on Ontology Reasoners and Comparison. **International Journal of Computer Applications**, v. 57, n. 17, p. 33–39, 2012. Disponível em: < http://www.ijcaonline.org/archives/volume57/number17/9208-3748>. Acesso em: 20 mar. 2015.

ALLEMANG, D.; HENDLER, J. A. **Semantic Web for the Working Ontologist: effective modeling in RDFS and OWL**. 2nd ed. Waltham, MA: Morgan Kaufmann/Elsevier, 2011.

ALMEIDA, M. B.; BAX, M. P. Uma Visão Geral sobre Ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. **Ciência da Informação, Brasília**, v. 32, n. 3, p. 7–20, 2003. Disponível em: <http://www.scielo.br/pdf/ci/v32n3/19019.pdf>. Acesso em: 20 mar. 2015.

ANDRADE, P. DE. Sistemas Especialistas de Apoio ao Diagnóstico em Medicina. Relações com o Teorema de Bayes e com a Lógica do Raciocínio Diagnóstico. **Arq Bras Cardiol**, v. 73, n. 6, p. 537–544, 1999. Disponível em: <http://publicacoes.cardiol.br/abc/1999/7306/73060008.pdf>. Acesso em: 29 jan. 2015.

BELHAJJAME, K. et al. Workflow-centric research objects: First class citizens in scholarly discourse. In: Proceeding of SePublica2012, p. 1–12, 2012. Disponível em: <http://oa.upm.es/20401/>. Acesso em: 30 abr. 2015.

BERNERS-LEE, T. **Linked data-design issues**. 2006. Disponível em: <http://www.w3.org/DesignIssues/LinkedData.html>. Acesso em: 11 fev. 2015.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. **Scientific American**, v. 284, n. 5, p. 28–37, 2001.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked data-the story so far. **International Journal on Semantic Web and Information Systems**, v. 5, n. 3, p. 1-22, 2009. Disponível em: <http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf>. Acesso em: 11 fev. 2015.

BRATT, S. **Semantic Web, and Other Technologies to Watch**. 2007. Disponível em: <http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb>. Acesso em: 12 fev. 2015.

BRESLIN, J.; DAVIS, M.; NOVA, S. What is the Evolution of the Internet to 2020?. **Semantic Wave 2008 Report: Industry Roadmap To Web 3.0 & Multibillion Dollar Market Opportunities**. Project10X's. 2008. Disponível em: <http://www.eurolibnet.eu/files/REPOSITORY/20090507165103_SemanticWaveReport2008.pdf>. Acesso em: 13 fev. 2015.

BRICKLEY, D.; GUHA, R. V. **RDF vocabulary description language 1.0: RDF schema**. W3C Recommendation, 2004. Disponível em: <http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>. Acesso em: 14 fev. 2015.

BRICKLEY, D.; GUHA, R. V. **RDF Schema 1.1**. W3C Recommendation, 2014. Disponível em: < http://www.w3.org/TR/rdf-schema/>. Acesso em: 02 mar. 2015.

BRICKLEY, D.; MILLER, L. **FOAF Vocabulary Specification 0.99**. Disponível em: <http://xmlns.com/foaf/spec/>. Acesso em: 11 maio 2015.

CARTLIDGE, A. et al. An Introductory Overview of ITIL® V3. **The UK Chapter of the itSMF**, 2007.

CHAUI, M. **Convite à Filosofia**. São Paulo: Editora Ática, 1995.

FALBO, R. DE A. **Integração de Conhecimento em um Ambiente de Engenharia de Software**. 1998. Tese (Doutorado em Engenharia de Sistemas e Computação) — Universidade Federal do Rio de Janeiro, Rio de Janeiro. 1998. Disponível em: <http://www.cos.ufrj.br/index.php?option=com_publicacao&task=visualizar&id=736>. Acesso em: 25 fev. 2015.

FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A.; JURISTO, N. Methontology: from ontological art towards ontological engineering. **AAAI-97 Spring Symposium Series**, 1997. Disponível em: <http://oa.upm.es/5484/>. Acesso em: 19 mar. 2015.

FONTES, C. A.; DE CARVALHO MOURA, A. M.; CAVALCANTI, M. C. Anotação Semântica em Documentos. In: IX Workshop de Teses e Dissertações em Banco de Dados, n. 9ª. **Anais**. 2010. Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=7888>. Acesso em: 11 fev. 2015.

GANGEMI, A. et al. Sweetening ontologies with DOLCE. In: Knowledge engineering and knowledge management: Ontologies and the semantic Web. Springer, 2002. p. 166–181. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.6038>. Acesso em: 20 mar. 2015.

GARIJO, D.; GIL, Y. Augmenting prov with plans in p-plan: scientific processes as linked data. In: Second International Workshop On Linked Science: Tackling Big Data (LISC), In Conjunction With The International Semantic Web Conference (ISWC). Boston, MA: 2012. Disponível em: <http://oa.upm.es/19478/1/INVE_MEM_2012_13915.pdf>. Acesso em: 15 mar. 2015.

GERBER, A.; VAN DER MERWE, A.; BARNARD, A. A Functional Semantic Web Architecture. In: Proceedings of the European Semantic Web Conference. **Anais**. 2008.

GÓMEZ-PÉREZ, A.; FERNÁNDEZ, M.; VICENTE, A. DE. Towards a Method to Conceptualize Domain Ontologies. In: 12th European Conference on Artificial Intelligence (ECAI'96). **Anais**. 1996. Disponível em: <http://oa.upm.es/7228/>. Acesso em: 19 mar. 2015.

GREENBERG, J.; SUTTON, S.; CAMPBELL, D. G. Metadata: A Fundamental Component of the Semantic Web. **Bulletin of the American Society for Information Science and Technology**, v. 29, n. 4, p. 16–18, 2003. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/bult.282/full>. Acesso em: 12 fev. 2015.

GRUBER, T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing?. **International Journal of Human-Computer Studies**, v. 43, n. 5, p. 907–928, 1995.

GRÜNINGER, M.; FOX, M. S. Methodology for the Design and Evaluation of Ontologies. In: Workshop On Basic Ontological Issues In Knowledge Sharing. **Anais**. 1995. Disponível em: <http://stl.mie.utoronto.ca/publications/gruninger-ijcai95.pdf>. Acesso em: 25 mar. 2015.

GUIMARÃES, J. I.; LOPES, A. A. Diagnóstico, Avaliação e Terapêutica da Hipertensão Pulmonar. **Diretrizes da Sociedade Brasileira de Cardiologia**, 2005. Disponível em: <http://publicacoes.cardiol.br/consenso/2005/039.asp>. Acesso em: 29 jan. 2015.

HAAV, H.-M.; LUBI, T.-L. A Survey of Concept-Based Information Retrieval Tools on the Web. In: Proceedings of the 5th East-European Conference ADBIS. **Anais**. 2001. Disponível em: <http://www.mii.vu.lt/ADBIS/local2/haav.pdf>. Acesso em: 20 mar. 2015.

HARRIS, S.; SEABORNE, A.; PRUD'HOMMEAUX, E. **SPARQL 1.1 Query Language**. W3C Recommendation, 2013. Disponível em: <http://www.w3.org/TR/sparql11-query/>. Acesso em: 16 mar. 2015.

HEATH, T.; BIZER, C. **Linked data: Evolving the web into a global data space**. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 2011.

HEFLIN, J. **OWL Web Ontology Language-Use Cases and Requirements**. W3C Recommendation, 2004. Disponível em: <http://www.w3.org/TR/webont-req/>. Acesso em: 01 abr. 2015.

HENDLER, J. Agents and the semantic web. **IEEE Intelligent systems**, v. 16, n. 2, p. 30–37, 2001.

HINZ, V. T. **Proposta de Criação de uma Ontologia de Ontologias**. 2006. Trabalho Individual I, Programa de Pós-Graduação em Informática, Universidade Católica de Pelotas, Pelotas, 2006. Disponível em: <http://ppginf.ucpel.tche.br/TI-arquivos/2006/VerlaniHinz/PPGINF-UCPel-TI-2006-2-10.pdf>. Acesso em: 24 fev. 2015.

HITZLER, P. et al. **OWL 2 Web Ontology Language Primer (Second Edition)**. W3C Recommendation, 2012. Disponível em: <http://www.w3.org/TR/owl2-primer/>. Acesso em: 25 fev. 2015.

HOBBS, J. R.; PAN, F. **Time Ontology in OWL**. W3C Working Draft, 2006. Disponível em: <http://www.w3.org/TR/owl-time/>. Acesso em: 25 fev. 2015.

ISAAC, A.; SUMMERS, E. **SKOS Simple Knowledge Organization System Primer**. W3C Group Note, 2009. Disponível em: <http://www.w3.org/TR/skos-primer/>. Acesso em: 23 fev. 2015.

JACYNTHO, M. D. DE A. **Um Modelo de Bloqueio Multigranular para RDF**. 2012. Tese (Doutorado em Informática) — Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática, Rio de Janeiro, 2012. Disponível em: <http://www.maxwell.vrac.puc-rio.br/Busca_etds.php?strSecao=resultado&nrSeq=20236@1>. Acesso em: 15 mar. 2015.

JARDIM, A. D. **Aplicações de Modelos Semânticos em Redes Sociais**. 2010. Dissertação (Mestrado em Ciência da Computação) — Centro Politécnico da Universidade Católica de Pelotas, Pelotas, 2010. Disponível em:<http://ppginf.ucpel.tche.br/DM-Arquivos/2010/PPGINF-UCPel-DM-2010-1-001.pdf>. Acesso em: 10 mar. 2015.

JASPER, R.; USCHOLD, M. A Framework for Understanding and Classifying Ontology Applications. In: Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW. **Anais**.1999. Disponível em: <http://ceur-ws.org/Vol-18/11-uschold.pdf>. Acesso em: 20 mar. 2015.

KAYSER, C. Hipertensão Pulmonar nas Doenças Reumáticas Autoimunes - Atualização. **SINOPSE DE REUMATOLOGIA**, n. Out 09 A 11 N 2, p. 57 à 72, 2009. Disponível em: <http://www.moreirajr.com.br/revistas.asp?fase=r003&id_materia=4161>. Acesso em: 17 mar. 2015.

KIFER, M.; BOLEY, H. **RIF Overview**. W3C Working Group Note, 2010. Disponível em: <http://www.w3.org/TR/2010/NOTE-rif-overview-20100622/>. Acesso em: 13 fev. 2015.

LASSILA, O.; MCGUINNESS, D. The Role of Frame-Based Representation on the Semantic Web. **Linköping Electronic Articles in Computer and Information Science**, v. 6, n. 5, 2001. Disponível em: <http://www.ep.liu.se/ea/cis/2001/005/>. Acesso em: 20 fev. 2015.

LICHTNOW, D.; DE OLIVEIRA, J. P. M. Relato e Considerações sobre o Desenvolvimento de uma Ontologia para Avaliação de Sites da Área de Saúde.

Cadernos de Informática, v. 4, n. 1, p. 7–46, 2009. Disponível em: <http://seer.ufrgs.br/cadernosdeinformatica/article/view/5v4n1p7-46>. Acesso em 12 mar. 2015.

MAGALHÃES, I. L.; PINHEIRO, W. B. **Gerenciamento de Serviços de TI na Prática**. **São Paulo:** Novatec, 2007.

MANOLA, F. et al. **RDF Primer**. W3C recommendation, 2004. Disponível em: < http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>. Acesso em: 09 mar. 2015.

MAXIMIANO, A. C. A. **Introdução à administração**. São Paulo: Editora Atlas, 2004.

MAYO CLINIC STAFF. **Pulmonary Hypertension**. Diseases and Conditions: Comprehensive guides on hundreds of conditions, 2013. Disponível em: <http://www.mayoclinic.org/diseases-conditions/pulmonary-hypertension/basics/definition/con-20030959>. Acesso em: 10 jan. 2015.

MAYO CLINIC STAFF. **Pneumonia**. Diseases and Conditions: Comprehensive guides on hundreds of conditions, 2015. Disponível em: <http://www.mayoclinic.org/diseases-conditions/pneumonia/basics/definition/con-20020032>. Acesso em: 10 jan. 2015.

MAYO CLINIC STAFF. **Influenza (flu)**. Diseases and Conditions: Comprehensive guides on hundreds of conditions, 2014. Disponível em: <http://www.mayoclinic.org/diseases-conditions/flu/basics/definition/con-20035101>. Acesso em: 10 jan. 2015.

MCGUINNESS, D. L.; VAN HARMELEN, F. **OWL Web Ontology Language Overview**. W3C Recommendation, 2004. Disponível em: <http://www.w3.org/TR/owl-features/>. Acesso em: 15 mar. 2015.

MICHAELIS, H. **Dicionário Online - Dicionários Michaelis - UOL**. Disponível em: <http://michaelis.uol.com.br/>. Acesso em: 30 mar. 2015.

MIZOGUCHI, R.; VANWELKENHUYSEN, J.; IKEDA, M. Task Ontology for Reuse of Problem Solving Knowledge. **Towards Very Large Knowledge Bases: Knowledge Building & Knowledge Sharing**, p. 46–59. [s.l.]. IOS Press, 1995.

MORGENSTERN, L. et al. **RIF Primer (Second Edition)**. W3C Working Group Note, 2013. Disponível em: <http://www.w3.org/TR/2013/NOTE-rif-primer-20130205/>. Acesso em: 24 mar. 2015.

NOY, N. F.; MCGUINNESS, D. L. Ontology development 101: A guide to creating your first ontology. **Stantford Medical Informatics Technical Report, SMI-2001-0880**, 2001.Disponível em: <http://www.ksl.stanford.edu/KSL_Abstracts/KSL-01-05.html>. Acesso em: 30 mar. 2015.

OREN, E. et al. What are semantic annotations. **Relatório técnico. DERI Galway**, 2006.Disponível em: <http://www.siegfried-handschuh.net/pub/2006/whatissemannot2006.pdf>. Acesso em: 05 abr. 2015.

PARSIA, B.; SIRIN, E. Pellet: An OWL DL Reasoner. Third International Semantic Web Conference-Poster. **Anais**. 2004. Disponível em: <http://pellet.owldl.com/papers/sirin05pellet.pdf>. Acesso em: 20 mar. 2015.

PEREIRA, J. R.; DE SOUZA, M. A.; DA COSTA, H. R. Gerenciamento de Problema: Uma Abordagem com Base na ITIL. **Pensar Tecnologia**, v. vol. 1, No. 2, 2012. Disponível em: <http://revistapensar.com.br/tecnologia/artigo/no=a12.pdf>. Acesso em: 10 mar. 2015.

PICKLER, M. E. V. Web Semântica: ontologias como ferramentas de representação do conhecimento. **Perspectivas em Ciência da Informação**, v. 12, n. 1, p. 65–83, 2007. Disponível em: <http://portaldeperiodicos.eci.ufmg.br/index.php/pci/article/view/251>. Acesso em: 06 mar. 2015.

POLLOCK, J. T. **Semantic Web for Dummies**. Hoboken, N.J.: Wiley, 2009.

PORTAL IG. **Gripe - Enciclopédia da Saúde - iG**. Portal. Disponível em: <http://saude.ig.com.br/minhasaude/enciclopedia/gripe/ref1238131621451.html>. Acesso em: 7 dez. 2014.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. W3C Recommendation, 2008. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Acesso em: 10 fev. 2015.

RAIMOND, Y.; ABDALLAH, S. **The Event Ontology**. Centre for Digital Music, 2007. Disponível em: <http://motools. sourceforge. net/event>. Acesso em: 23 abr. 2015.

REZENDE, D. A. **Tecnologia da Informação Integrada à Inteligência Empresarial**. São Paulo: Editora Atlas, 2002.

RÜHLE, S.; BAKER, T.; JOHNSTON, P. **User Guide - DCMI_MediaWiki**. Wiki. Disponível em: <http://wiki.dublincore.org/index.php/User_Guide>. Acesso em: 22 abr. 2015.

SOUZA, R. R.; ALVARENGA, L. A Web Semântica e suas contribuições para a ciência da informação. **Ciência da Informação, Brasília**, v. 33, n. 1, p. 132–141, 2004. Disponível em: <http://revista.ibict.br/cienciadainformacao/index.php/ciinf/article/view/50>. Acesso em: 15 mar. 2015.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge Engineering: principles and methods. **Data & Knowledge Engineering**, v. 25, n. 1, p. 161–197, 1998.

USCHOLD, M.; GRUNINGER, M. Ontologies: Principles, methods and applications. **Knowledge Engineering Review**, v. 11, n. 2, p. 93–136, 1996.

USCHOLD, M.; KING, M. Towards a Methodology for Building Ontologies. In: Workshop on Basic Ontological Issues in Knowledge Sharing held in conjunction with IJCAI-95, 1995. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.5357>. Acesso em: 10 abr. 2015.

VAN HEIJST, G.; SCHREIBER, A. T.; WIELINGA, B. J. Using Explicit Ontologies in KBS Development. **International Journal of Human-Computer Studies**, v. 46, n. 2, p. 183–292, 1997.

VIINIKKALA, M. **Ontology in Information Systems**. 2004. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.201.2639&rep=rep1&type=pdf>. Acesso em: 11 mar. 2015.

W3C - OWL Working Group. **OWL 2 Web Ontology Language: Document Overview (Second Edition)**. W3C Recommendation, 2012. Disponível em: <http://www.w3.org/TR/owl2-overview/>. Acesso em: 29 mar. 2015.

WEINSTEIN, P. C. Ontology-Based Metadata: Transforming the MARC Legacy. Proceedings of the Third ACM Conference on Digital Libraries. **Anais**. ACM, 1998.

WELTY, C.; MCGUINNESS, D. L.; SMITH, M. K. **OWL Web Ontology Language Guide**. W3C Recommendation, 2004. Disponível em: <http://www.w3.org/TR/owl-guide/>. Acesso em: 09 abr. 2015.

WOOLDRIDGE, M. **An Introduction to Multiagent Systems**. 2nd ed. Chichester, U.K: Wiley & Sons, 2009.

YU, L. **A developer's guide to the semantic Web**. Springer Science & Business Media, 2011. ISBN: 978-3-642-15969-5.