Universidade Federal de Alagoas

Instituto de Computação

Coordenação de Pós-Graduação em Informtica

Uma Abordagem para Descoberta, Composição e Invocação Automática de Serviços Semânticos Aplicada ao Domínio de Mineração de Dados

Michel de Sousa Miranda

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Alagoas - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Informtica.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Computação Visual e Inteligente

Dr. Evandro de Barros Costa

(Orientador)

Maceio, Alagoas, Brasil

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecário Responsável Janis Christine Angelina Cavalcante

M672a Miranda Michel de Sousa.
Uma abordagem para descoberta, composição e invocação automática de serviços semânticos aplicada ao domínio de mineração dados / Michel de Sousa Miranda – 2018.

111 f. : il. color.

Orientador: Dr. Evandro de Barros Costa.
Dissertação (mestrado em Informática) - Universidade Federal de Alagoas Instituto de Computação. Maceió, 2018.

Bibliografia: f. 87-90.
Anexo: f. 91-111.

1. Framework Semântico para Mineração de Dados. 2. Web semântica.
3. Serviços Web Semânticos 4. Descoberta e Composição Automática de Serviços I. Título.

CDU: 004.4’24

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Informática – PpgI
Instituto de Computação
Campus A. C. Simões BR 104-Norte Km 14 BL 12 Tabuleiro do Martins
Maceió/AL - Brasil CEP: 57.072-970 I Telefone: (082) 3214-1401

Membros da Comissão Julgadora da Dissertação de **Michel de Sousa Miranda**, intitulada: "*Uma Abordagem para Descoberta, Composição e Invocação Automática de Serviços Semânticos Aplicada ao Domínio de Mineração de Dados*", apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas em 27 de abril de 2018, às 10h10, na Sala 15, do Instituto de Computação da UFAL.

COMISSÃO JULGADORA

Prof. Dr. Evandro de Barros Costa
UFAL – Instituto de Computação
Orientador

Prof. Dr. Patrick Henrique da Silva Brito
UFAL – Instituto de Computação
Examinador

Prof. Dr. Rafael Ferreira Leite de Mello
Universidade Federal Rural de Pernambuco

Dedicatória

"Dedico esse trabalho ao meu pai João Donato de Souza Sobrinho (in memorian), com todo o meu amor e gratidão. Eternas saudades".

Agradecimentos

Agradeço, primeiramente, a Deus pela vida, pela saúde, pelas oportunidades e proteção durante toda a caminhada do início até o fim do curso.

A minha mãe Maria Aparecida de Sousa Miranda, por todo apoio e incentivo ao longo dessa caminhada. Por entender que a educação é fundamental para formação do ser humano.

Obrigado aos meus irmãos (Marcos e Mônica) e minha sobrinha (Natally), que nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente!

Agradeço a meu orientador Prof. Dr. Evandro de Barros Costa, por toda a paciência, ensinamentos, críticas e aprendizado durante toda essa caminhada.

Aos membros da banca, os professores Dr. Rafael Mello e ao Dr. Patrick Brito, pela disposição e interesse em contribuir para o melhoramento deste trabalho.

Agradeço ao professor (amigo e irmão) Dr. Társis Marinho, por todo apoio e confiança que depositou em mim.

Ao grupo de pesquisa TIPS (Tecnologias, Inteligentes, Personalizadas e Sociais) e seus integrantes Heitor Barros, Marlos Silva, Jonathas Magalhães, Rubens Pessoa, Caio Barbosa e Sebastião companheiros de trabalhos e irmãos na amizade que fizeram parte da minha formação e que vão continuar presentes em minha vida com certeza. Obrigado por tudo!

Ao Programa de Pós-Graduação em Informática – PPGI/UFAL em especial a minha amiga Flora Teixeira, ao meu amigo (irmão) Vitor Torres e Anderson Cabral. Aos professor do PPGI por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

Um obrigado em especial para minha namorada, Larissa Lopes, pela amizade verdadeira a qual construímos durantes este tempo. Pelos os momentos que passamos juntos, na alegria ou na tristeza, sempre estava ali disponível para me ouvir.

Meus agradecimentos aos amigos (a), Élvis Correia, Igor Rafael, Ezequiel Batista, Jorge Henrique, Vitor Nunes e Bruna Granja, irmãos na amizade que fizeram parte da minha formação e que vão continuar presentes em minha vida com certeza. Obrigado por tudo!

Agradeço a Fundação de Amparo à Pesquisa do Estado de Alagoas (FAPEAL) pelo apoio financeiro para realização da pesquisa.

Por fim, a todos que direto ou indiretamente fizeram parte da minha formação, o meu muito obrigado!

É você quem vai fazer o seu diploma, ninguém vai fazer por você.
Se o professor for ótimo, melhor; Se o professor for ruim, supere-o;
Se a instituição existe, use - a para chegar ao seu objetivo.
(Leandro Karnal)

Resumo

O desenvolvimento de aplicações baseadas em Serviços *Web* Semânticos têm evoluído e ganho uma atenção especial, tanto da academia, quanto da indústria. Essencialmente, eles têm sido utilizados para possibilitar e automatizar tarefas que envolvem aspectos, tais como: descoberta, composição e invocação automática de serviços. Para isso, a comunidade tem se dedicado na criação de modelos semânticos e ferramentas que sejam capazes de explorar os conceitos semânticos dos serviços. Nesse contexto, o trabalho em pauta propõe um *framework* baseado em Serviços *Web* Semânticos aplicados ao domínio de Mineração de Dados Educacionais baseado em um novo modelo semântico que apoie o processo de descoberta, composição e invocação de forma automática e dinâmica. Para tal, os serviços semânticos encapsulam técnicas e algoritmos de mineração, pré-processamento e pós-processamento de dados. A fim de avaliar o *framework* proposto, foi realizado um experimento baseado em cenários que simulam a execução da solução proposta em um ambiente real com a intenção de avaliar os atributos de qualidade desempenho e confiabilidade.

Palavras-chave: *Framework* Semântico para Mineração de Dados, *Web* Semântica, Serviços *Web* Semânticos, Descoberta e Composição Automática de Serviços.

Abstract

The development of applications based on Semantic Web Services has evolved and gained special attention from both academia and industry. Essentially, they have been used to enable and automate tasks that involve aspects such as: discovery, composition, and automatic invocation of services. For this, the community has been dedicated in the creation of semantic models and tools that are able to explore the semantic concepts of the services. In this context, the work in question proposes a Framework based on Semantic Web Services applied to the domain of Educational Data Mining based on a new semantic model that supports the process of discovery, composition and invocation in an automatic and dynamic way. To this end, semantic services encapsulate techniques and algorithms for mining, pre-processing and post-processing of data. In order to evaluate the proposed framework, we performed an experiment based on scenarios that simulate the execution of the proposed solution in a real environment with the intention of evaluating the attributes of quality performance and reliability.

Key-words: Semantic Framework for Data Mining, Semantic Web; Semantic Web Services; Automatic Discovery, Composition and Invocation of Services

Conteúdo

Lista de Símbolos

SWS - *Serviços Web Semânticos*

DM - *Mineração de Dados*

EDM - *Mineração de Dados Educacionais*

SOA - *Arquitetura Orientada a Serviços*

ES - *Engenharia de Software*

IA - *Inteligência Artificial*

IBM - *International Business Machines*

SOAP - *Simple Object Access Protocol*

WSDL - *Web Services Description Language*

UDDI - *Universal Description, Discovery and Integration*

SOAP - *Protocolo Simples de Acesso a Objetos*

OS - *Orientação a Serviços*

W3C - *World Wide Web Consortium*

URI - *Uniform Resource Identie*

XML - *eXtensible Markup Language*

WWW - *World Wide Web*

RDF - *Resource Description Framework*

OWL - *Web Ontology Language*

OWL-S - *Semantic Markup for Web Services*

WSMO - *Web Service Modeling Ontology*

JSON - *JavaScript Object Notation*

REST - *Representational State Transfer*

XML - *eXtensible Markup Language*

ATAM - *Method for architecture evaluation*

SGBD - *Sistema de Gerenciamento de Banco de Dados*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

O presente trabalho situa-se na linha de pesquisa Computação Visual e Inteligente do Programa de Pós-graduação em Informática (PPgI/UFAL), que no contexto dessa dissertação envolve as áreas de Serviços Web Semânticos (SWS), Inteligência Artificial (IA) e Engenharia de Software (ES). Nesta perspectiva, o trabalho aqui descrito é o resultado de um estudo que lida com aspectos do uso de Serviços Web Semânticos aplicados a Mineração de Dados Educacionais. Assim, pretende-se a contribuição de uma aplicação capaz de descobrir, compor e invocar de forma automática e dinâmica serviços Web semânticos que encapsulam técnicas e algoritmos de pré-processamento, mineração e pós-processamento de dados, disponíveis na ferramenta Weka.

Neste capítulo apresenta-se um panorama geral o trabalho em questão, iniciando-se com a contextualização e problematização observados sobre o tema em pauta, conectando-os com os objetivos do trabalho.

1.1 Contextualização e Problemática

É notório que o surgimento da *Web* no início da década 90, idealizada por Tim Berners-Lee, despertou muitas mudanças nas relações entre seres humanos e as estruturas organizacionais. O crescimento e a evolução da *Web* trouxeram grandes benefícios à sociedade, tais como a possibilidade de usuários e/ou organizações acessarem e compartilharem recursos em qualquer hora ou momento através de aplicações baseadas na *Web*. A *Web*, desse modo, tornou-se o principal e mais importante meio de comunicação e troca de informação entre

usuários e organizações [25]. Por consequência, a *Web* tem apresentado um alto crescimento, produzindo uma demanda cada vez maior no desenvolvimento de aplicações/soluções com o objetivo de atender a novas necessidades de usuários e organizações. Os serviços *Web*, por exemplo, surgiram a fim de atender parte dessas necessidades [5]. Para Sheng et al. [37], os serviços *Web* são compreendidos como uma nova geração de aplicativos *Web* que podem fornecer funcionalidades e interoperabilidade que vão desde o básico até os mais complexos processos requisitados por aplicações de diferentes sistemas, independente de plataforma.

Neste contexto, Barros [1] afirma que encontrar e utilizar recursos disponíveis na Web tornou-se um processo complexo, pois a tarefa de buscar e localizar recursos úteis fica totalmente a cargo dos usuários. Dessa maneira, com o objetivo de auxiliar os usuários nesse processo, a comunidade tem feito uso de metadados para representar e manipular informações de recursos da *Web*. Metadados são definidos como dados que descrevem atributos de um recurso [16]. O uso do metadados podem estar associados a utilização de ontologias, que fornecem uma descrição semântica para as relações dos recursos em um domínio específico. Assim, surge o conceito de Web Semântica.

Para Ikematu [16], o principal objetivo da *Web* Semântica é desenvolver Serviços *Web* que recebam descrições semânticas para que possam solucionar alguns dos problemas encontrados em outras tecnologias, como *Web Services Description Language* (WSDL), *Simple Object Access Protoco* (SOAP), *Universal Description, Discovery and Integratio* (UDDI), por exemplo: a falta de informação semântica do recurso, que torna o processo de descoberta e invocação de serviços *Web* de forma manual. Além disso, McIlraith et al. [23] destaca que na perspectiva do uso de Serviços *Web* Semânticos, é tornar possível a automatização de algumas tarefas realizadas por agentes de *softwares*, tais como:

- **Descoberta automática de serviços** - é um processo automatizado para localização de serviços da *Web* que possam atender uma determinada necessidade, baseada em restrições especificadas pelo usuário.

- **Invocação automática de serviços** - a invocação automática de um serviço *Web* por um programa de computador ou agente de *software*, dada apenas uma descrição declarativa deste serviço, diferente do caso em que os agentes são pré-programados para invocar um serviço em particular.

- **Composição automática de serviços e interação** - esta tarefa envolve a seleção, composição e interação de serviços *Web* para realizar uma tarefa complexa, a partir de uma descrição de alto nível de um objetivo.

Os Serviços *Web* Semânticos têm sido adotados como soluções em diversos domínios. No domínio da Mineração de Dados Educacional (EDM), por exemplo, soluções que adotam serviços *Web* semânticos (SWS) têm sido propostas a fim de solucionar diversos desafios da área, tais como [31], [32] e [33]:

- **Integração** - As ferramentas de mineração de dados precisam ser integradas aos ambientes educacionais de forma simples. Todas as tarefas de mineração de dados devem ser realizadas em uma única aplicação, para que os *feedbacks* e resultados obtidos possam ser aplicado diretamente dentro dos ambientes;

- **Padronização dos métodos e dados**- Ferramentas atuais de mineração de dados são úteis apenas aos seus desenvolvedores. Não existem ferramentas que possam facilmente ser reutilizadas ou adaptadas para diferentes sistemas educacionais.

Entre as propostas encontradas na literaturas que fazem uso de SWS e visam atender os desafios acima mencionados, destacam-se: Zorilla et al. [45], Barros [1] e Souza [39]. No entanto, essas soluções possuem diversas limitações, muitas vezes, inerentes aos modelos semânticos adotados na construção dos SWS. Essas limitações incluem: modelos semânticos complexos; altos custos para criação e manipulação de novos serviços *Web* semânticos; processo de adoção/extensão dessas soluções é demasiadamente complexo e pouco claros; baixo desempenho; APIs de manipulação dos modelos semânticos adotados pouco maduras, complexas e com baixo desempenho; dificuldade em representar semanticamente tipos abstratos de dados durante construção dos SWS etc.

Diante dessas questões, o presente trabalho busca contribuir com o desenvolvimento de aplicações baseadas em serviços *Web* semânticos através de um *Framework* que apoie o processo de descoberta, composição e invocação de forma automática e dinâmica de serviços com base em um novo modelo semântico e uma abordagem centrada em dados aplicadas ao domínio da mineração de dados. Desse modo, a solução proposta fornece serviços *Web* semântico que encapsulam técnicas e algoritmos de mineração, pré-processamento e pós-processamento de dados.

1.2 Objetivo Geral

O objetivo geral deste trabalho é propor uma abordagem baseada em Serviços Web Semânticos aplicada a mineração de dados, visando realizar a descoberta, composição e invocação automática de serviços semânticos que encapsulam técnicas e algoritmos de pré-processamento, mineração e pós-processamento de dados.

1.2.1 Objetivos Específicos

- Possibilitar a adição, adaptação e/ou remoção de algoritmos/ferramentas de mineração de maneira dinâmica;
- Simplificar o processo de construção de serviços *Web* semânticos;
- Simplificar e promover o desempenho do processo de descoberta, composição e invocação de serviços *Web* semânticos de forma automática e dinâmica;
- Possibilitar a integração de algoritmos e/ou ferramentas de mineração de dados através de serviços;
- Promover o reuso de soluções de mineração de dados aplicadas a diferentes domínios;

1.3 Estrutura do Trabalho

Este trabalho está organizado como se segue. No Capítulo 2 - encontra-se a fundamentação teórica deste trabalho. São apresentados conceitos relacionados às noções de *Service Oriented Architecture* (Arquitetura Orientada a Serviços), Serviços *Web*, *Web* Semântica, Serviços *Web* Semânticos e *Frameworks* conceituais para descrição de Serviços *Web* Semânticos. No Capítulo 3 - é apresenta um levantamento dos trabalhos relacionados a esta proposta, descrevendo suas características e limitações. Além disso, apresenta uma comparação entre as propostas relacionadas, destacando os requisitos atendidos por cada uma. No Capítulo 4 - é apresentada uma visão geral da abordagem proposta neste trabalho, mostrando detalhes da sua arquitetura e aspectos de implementação. No Capítulo 5 - é apresentado o experimento realizado para avaliar o comportamento da abordagem proposta em descobrir, compor e invocar de forma automática e dinâmica serviços *Web* de mineração de dados em diferentes

cenários de aplicação. Além disso, é apresentada a descrição do comportamento da solução em cada cenário de execução. Por fim, no Capítulo 6 - encontra-se as considerações finais e conclusões sumarizadas deste trabalho. Além disso, este capítulo aponta as limitações encontradas neste trabalho e o direcionamento para a realização de trabalhos futuros.

Capítulo 2

Referencial Teórico

"Em todas as coisas, o sucesso depende de preparação prévia."

- Confúcio

Este capítulo tem por objetivo apresentar os principais tópicos que formam a base teórica para o entendimento do trabalho aqui proposto. Inicialmente, são discutidos conceitos de Arquitetura Orientada a Serviços e Serviços *Web*. Em seguida, são apresentados os conceitos da *Web* Semântica e suas camadas. E, por fim, os Serviços *Web* Semânticos (SWS) são discutidos, mostrando seus conceitos básicos e os principais modelos conceituais utlizados para descrição de serviços.

2.1 Arquitetura Orientada a Serviços

De acordo com Linthicum [21], a Orientação a Serviços (OS) [1] é considerada a evolução dos modelos desenvolvidos no passado. Por exemplo, nos anos 80, vimos a introdução dos modelos Orientador a Objetos; em seguida, os modelos de desenvolvimento baseado em componentes nos anos 90; e agora temos Orientação de Serviços (SO). A mesma mantém os benefícios baseado em componentes como autodescrição, encapsulamento, descoberta dinâmica e carregamento, mas realiza mudanças no paradigma de invocação remota de métodos em objetos, passando mensagens entre serviços. As mensagens enviadas entre os serviços obedecem a uma estrutura padrão definida em um esquema. O esquema além de

[1] do Inglês *(Service Orientation)*

definir a estrutura das mensagens, force os modelos comportamentais para definir padrões aceitáveis de troca de mensagens e políticas para definir a estrutura semântica do serviço [21].

Neste contexto, para Narock et al. [28] o paradigma de Arquitetura Orienta a Serviços (SOA) do inglês *(Service Oriented Architecture)* define um conjunto de princípios e metodologias para a concepção e desenvolvimento de *softwares* sob a forma e serviços interoperáveis. Assim, estes serviços, são referidos formalmente como serviços *Web*. Ou seja, são aplicações que podem ser utilizadas automaticamente por aplicações e sistemas computacionais. Frequentemente, esses serviços *web* são incorporados a aplicações para o desenvolvimento de sistemas confiáveis.

Barry [2] define SOA como uma coleção de serviços que se comunicam-se entre si. Desta forma, a comunicação pode envolver uma simples passagem de dados para um determinado serviço, desta mesma forma, também pode acontecer que a comunicação envolva dois ou mais serviços coordenando uma determinada atividade.

Para Barros [1] no geral, SOA é compreendida como uma evolução da computação distribuída na qual as lógicas de negócio são encapsuladas na forma de serviços de maneira modular, de forma que possam ser invocados por clientes. A seguir, será apresenta algumas das principais características da arquitetura SOA apresentadas em Huhns et al. [14], Singh [38] et al. :

- Fraco acoplamento: em SOA cada serviços é compreendido como um elemento independente, da mesma forma que um elemento é independe em Arquitetura Orientada a Componentes.

- Implementação: serviços desenvolvidos em diferentes tecnologias podem se comunicar com outros serviços, visto que em SOA, a interface é o que importa. Não podemos depender dos detalhes das implementações dos componentes, ou seja, a abordagem não precisa ser especificada para um conjunto de linguagens de programação.

- Flexibilidade de configuração: em SOA sistemas são frequentemente configurados de forma tardia e flexível, o que significa que diferentes componentes estão ligados entre si no fim de execução de processos. Assim, a configuração pode mudar dinamicamente conforme necessário e sem perda de correção.

- Persistência: em SOA os serviços não precisam de um logo tempo de vida, devem existir o tempo suficiente para detectar quaisquer exceções relvantes.

- Granularidade: em SOA os serviços devem ser tomados como elementos de baixa granularidade, em vez de modelar ações e interações a um nível detalhado, seria melhor ter uma visão mais abstrata quanto possível, ou seja, ocultando detalhes da modelagem de ações e interações.

Entretanto, apesar das características apresentadas em Huhns et al. [14] e Singh [38] et al. para aplicações baseadas no paradigma SOA , um outro mecanismo importante que sistemas devem garantir é a confiabilidade. Em Pan [29] a confiabilidade de *software* é definida como a probabilidade de um software operar sem falhas em um ambiente específico por um período de tempo. Alguns dos requisitos de qualidade de confiabilidade em sistemas baseados em serviços *Web* são definidos em Zhang [44]. Esses requisitos são divididos entre dos grupos, ou requisitos funcionais e os não-funcionais. Os requisitos funcionais são compostos por requisitos de correção, tolerância a falhas e testabilidade. Enquanto os não-funcionais são disponibilidade, desempenho e interoperabilidade. A seguir, são apresentados em maiores detalhes:

- Correção: implica que o serviço *Web* gera uma saída razoável partir da entrada do domínio do problema;

- Tolerância a falhas: implica que o serviço *Web* é capaz de produzir resultados aceitáveis, embora esteja com defeito;

- Testabilidade: implica que o serviço *Web* permite que as falhas existentes a ser detectado no momento do teste;

- Disponibilidade: implica que o serviço *Web* está disponível no momento de invocação;

- Desempenho: implica que o serviço *Web* oferece a uma velocidade aceitável no momento de invocação;

- Interoperabilidade: implica que o serviço *Web* coexiste com outros componentes do sistema no contexto de uma ambiente específico.

2.2 Serviços Web

Na computação, os serviços *Web* ou *Web* services define um novo paradigma de computação distribuída. Neste contexto, os serviços Web são vistos como uma solução utilizada para prover integração e comunicação entre diferentes aplicações. Com a inclusão dos serviços *Web* é possível que aplicações sejam capazes de interagir com outras aplicações independentemente de plataforma que foram desenvolvidas. Os serviços *Web* são componentes de *softwares* que permitem as aplicações enviar e receber dados que em grande parte, estão distribuídos geograficamente na *Internet* [22].

Hoje em dia, o termo serviço *Web* é usado com bastante frequência, embora nem sempre com o mesmo significado. A W3C[2] define um serviço *Web* como um sistema de *softwares* identificado por uma URI[3] e com especificações definidas em um documento obedecendo o padrão da linguagem XML[4]. A descrição em XML permite que outros sistemas sejam capazes de identificar e interagir com os serviços [41]. Em Fensel et al. [10] um serviço *Web* é definido como objetos de *softwares* que podem ser agregados pela a *Internet* usando protocolos padrão para executar funções ou executar processos de negócios. Para a IBM[5] um serviço *Web* é compreendido como um conjunto de padrões emergentes que permitem a integração de processos e interoperabilidade entre sistemas. De acordo com a IBM [15] os serviços *Web* são como uma geração de aplicativos *Web* que podem fornecer funcionalidades e interoperabilidade que vão desde o básico para os mais complexos processos requisitados por aplicações e diferentes sistemas independente de plataforma.

Desta forma, com base nas definições apresentadas anteriormente, um serviços Web pode ser compreendido como:

- Um método de comunicação entre aplicações que estão conectadas na *Internet* independente da plataforma;
- Um sistema de *softwares* que possibilita a comunicação e interoperabilidade entre aplicações e sistemas diferentes;

[2] do inglês World Wide Web Consortium
[3] *do inglês Uniform Resource Identifie*
[4] *do inglês eXtensible Markup Language*
[5] *do inglês International Business Machines*

- Um *Softwares* que possui uma coleção de padrões ou protocolos para trocar de informações entre aplicações, por exemplo: troca de dados, entre outros;

- Um *software* que fornece operações e funcionalidades que podem ser utilizadas por outras aplicações.

Por exemplo, a Figura 2.1 ilustra os conceitos apresentados anteriormente. Dado que uma aplicação necessite utilizar recursos desenvolvidos em outras linguagens de programação como Java, .Net, PHP, entre outras. Assim, neste caso, com a introdução dos serviços *Web* basta apenas a aplicação fazer uso de serviços para se comunicar com aplicações conectadas na Internet. Assim, na Figura 2.1, por exemplo, aplicações desenvolvidas na linguagem Java podem interagir com outras aplicações como Java, .Net, PHP, entre outras. Neste contexto, o serviço *Web* é uma forma de comunicação independente de linguagem [17].

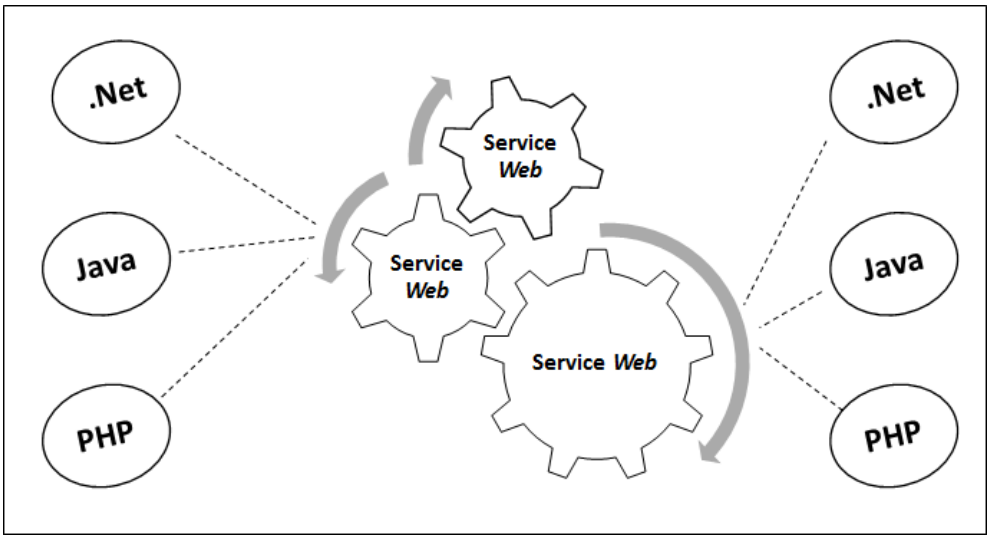

Figura 2.1: Serviços Web prover comunicação entre aplicações independente da linguagem de programação. Fonte: autor da pesquisa

A próxima seção, apresenta um breve resumo sobre a *Web* Semântica. Seguida, pela pilha de camadas de tecnologias definidas por Tim Berners-Lee et al. [4] para da suporta ao desenvolvimento de aplicações baseadas em conceitos semânticos.

2.3 Web Semântica

A *Web* semântica pode ser compreendida como uma nova extensão da atual *World Wide Web* (WWW) com uma estrutura comum para criação e representação de conteúdos disponíveis em páginas *Web* com mais significado. Assim, a *Web* semântica proporciona a criação de um ambiente cooperativo possibilitando que agentes de *softwares* e agentes humanos trabalhem em cooperação. Desta forma, a mesma, utiliza conceitos semânticos para dá significados aos dados publicados na *Internet*, de modo que seja perceptível por qualquer que seja o agente.

A seguir, na seção 2.3.1 é mostrado a Figura 2.2 em maiores detalhes que representa a arquitetura da *Web* semântica definida por Tim Berners-Lee et al. [4]. Nesta Figura 2.2, estão definidas as tecnologias da *Web* semântica que define uma nova abordagem para o gerenciamento de informações e processos, cujo o princípio fundamental é a criação e o uso de metadados semânticos.

2.3.1 Pilha de Camadas da *Web* Semântica

A pilha de camadas que ilustra a arquitetura da *Web* semântica apresentada na Figura 2.2 foram propostas inicialmente por Tim Berners-Lee et al.[4]. Este modelo de arquitetura exibe a distribuição e relação entre as tecnologias que fazem parte da *Web* semântica. Estas tecnologias da *Web* semântica oferecem uma nova abordagem para o gerenciamento de informações e processos, cujo o principal objetivo fundamenta na criação e uso de metadados semânticos. A seguir, será apresenta a descrição das camadas da Figura 2.2 em maiores detalhes. Os conceitos apresentados para as camadas da Figura 2.2 foram baseados em Harb et al. [12].

URI e *Unicode*: uma das principais características da Web é a flexibilidade para os usuários atribuir coisas e recursos. Um recurso pode ser qualquer coisa, como um livro, documento, vídeo, entre outros. Esta flexibilidade é provida pela URI (Identificador Universal de Recursos ou do Inglês *Unified Resource Identifiers*) e *Unicode*. Um identificador de recursos universal (URI) é uma sequência de caracteres formada que serve como um meio de identificar um recursos único na Web. Ou seja, cada URI pertece apenas a um único recurso. Um URI pode ser classificado como o localizador, um nome ou ambos. Além disso, através da URI se torna possível identificar e localizar recursos disponíveis na *Web* para dis-

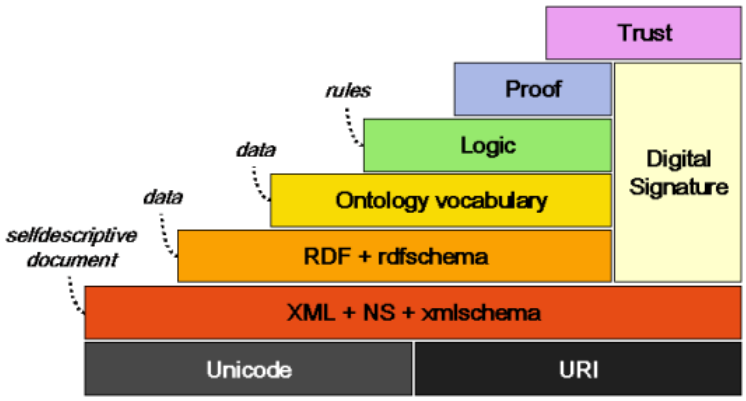

Figura 2.2: Distribuição da Pilha de Camadas da *Web* semântica criadas por Berners-Lee et al. (2001)

ponibilizar em funcionalidades providas por *softwares*. O *Unicode* tem a responsabilidade em fornecer um número exclusivo para cada caractere, independentemente da plataforma, programa ou idioma.

XML e XML Schema: estes estão acima da camada URI e Unicode. O XML[6] (Linguagem Extensível de Marcação ou do inglês *eXtensible markup language*) é uma linguagem de marcação recomendada pela a W3C[7][8] para criação de documentos com dados organizados hierarquicamente. O XML *Schema*[9] foi aprovado como recomendação da W3C para definir a estrutura, conteúdo e semântica de documentos XML.

RDF e RDF Schema: este estão acima das camadas XML e XML *Schema*. O RDF *(Resource Description Framework)* é uma estrutura para representar informações sobre recursos em um formulário gráfico. RDF é baseado em uma tripla (O-A-V) que formam um gráfico de dados com relações entre objeto (um recurso), um atributo (uma propriedade) e um valor (um recurso). O RDF Schema (RDFS) define o vocabulário do modelo RDF. Ele fornece um mecanismo para descrever propriedades específicas do domínio e classes de recursos aos quais essas propriedades podem ser aplicadas, usando um conjunto de primitivas básicas de modelagem (classe, subclasse de propriedade, sub propriedade de domínio, intervalo, tipo).

[6]https://www.w3.org/TR/REC-xml/sec-intro

[7]do inglês *(World Wide Web Consortium)* é uma organização internacional de padrões que desenvolve os pilares de tecnologias Web. O grupo da W3C é liderado pelo inventor da *Web* Tim Berners-Lee e CEO Jeffrey Jaffe, a missão do W3C é levar a *Web* ao seu pleno potencial.

[8]https://www.w3.org/Consortium/

[9]https://www.w3.org/XML/Schema

No entanto, RDFS é bastante simples e ainda não fornece semântica exata de um domínio.

Ontology vocabular: está camada é localizada acima das camadas RDF e RDF Schema. A ontologia compreende um conjunto de termos de conhecimento, incluindo o vocabulário, a interconexões, regras simples de inferência e lógica para algum tópico particular. As ontologias facilitam o compartilhamento de conhecimento e fornecem conteúdo *Web* reutilizável por serviços *Web* e Aplicações.

Logic: está camada permite a criação de regras. Além de possibilitar a criação de regras, a camada logica é responsável por indicar qualquer regra e permitir que um agente de *software* ou uma máquina faça inferências úteis sobre a regra.

Proof: está camada é responsável em realizar a verificação dos dados transmitidos, assim analisando se os dados são válidos.

Digital Signature: está é composta por blocos criptografados de código que verificam se as informações que estão sento transmitidas provêm de uma fonte confiável e através do uso de verificação garantindo que os dados não foram alterados.

Trust: a camada de confiança tem a responsabilidade de garantir que os dados são confiáveis, verificando a assinatura digital para ver quem será o responsável para preparação dos dados.

2.3.2 OWL

A OWL[10] *(do inglês Web Ontology Language)* é uma linguagem recomendada pela W3C[11] para definir ontologias *Web* . Uma ontologia OWL possibilita descrever um domínio em termos de classes, propriedades e indivíduos com descrições e características dos objetos [24]. A OWL é subdividida em três sub-linguagens de acordo com sua expressividade:

- OWL Lite: é a sub-linguagem sintaticamente mais simples da OWL, dado que seu poder de expressividade é relativamente inferior comparado com as outras sub-linguagens da OWL. A OWL Lite destina-se a situações em que apenas são necessarias restrições e uma hierraquia de classes simples.

- OWL DL: está linguagem é mais expressiva do que a OWL Lite, mantendo a inte-

[10] https://www.w3.org/OWL/

[11] *World Wide Web Consortium (W3C)*

gridade computacional. A OWL DL inclui todas as construções da linguagem OWL, podendo ser usadas sob determinadas restrições. Está linguagem é baseada em lógica descritiva.

- OWL Full: representa a mais expressiva das sub-linguagem da OWL. A OWL Full destina-se a situações onde a alta expressividade é mais importante do garantir a decibilidade ou completude da linguagem.

2.3.3 Serviços *Web* Semânticos

Os Serviços *Web* Semânticos foram inicialmente propostos por McIlraith et al. [25] como uma extensão dos serviços *Web* com descrições semânticas afim de fornecer definições declarativas formais de suas interfaces, bem como suas propriedades e capacidade de captura declarativa que os serviços fazem.

Para Martin et al. [23] os serviços Web semânticos são compreendidos como serviços *Web* enriquecidos de anotações Semânticas. A representação semântica do serviço é realizar por meio de ontologias com a com a finalidade de proporcionar um cenário onde os agentes de *software* sejam capazes de explorar as informações contidas nessas descrições semânticas e executar tarefas em um domínio específico.

Segundo Martin et al. [22] a *Web* Semântica deve possibilitar que o acesso não seja restrito apenas aos conteúdos presentes na *Web*, mas também para os serviços na *Web*. Dessa forma, a introdução de semântica aos serviços visa a automação dos processos de descoberta e composição de serviços. Assim, possibilitando que usuários e agentes de *softwares* sejam capazes de descobrir, compor e invocar serviços de forma automática e dinâmica.

Neste contexto, conforme McIlraith et al. [23] na perspectiva do uso de Serviços *Web* Semânticos, pode-se alcançar as seguintes funcionalidades descoberta automática de serviços, invocação automática de serviços e composição automática de serviços e interação.

Descoberta automática de serviços - é um processo automatizado para localização de serviços da *Web* que possam atender uma determinada necessidade, baseada em restrições especificadas pelas o usuário. Por exemplo, o usuário pode querer encontrar um serviço que venda bilhetes de avião entre duas cidades e aceita um determinado cartão de crédito. Atualmente, para que essa tarefa seja executada, um usuário deve utilizar uma ferramenta de

busca para encontrar o serviço, ler a páginas *Web* deste serviço, e executar o serviço manualmente, para determinar se ele satisfaz suas restrições. Com Serviços *Web* Semânticos, a informação necessária para descoberta do serviço poderia ser especificada através de marcações semânticas, interpretáveis por computador, presentes na página do serviço, e uma ferramenta de busca baseada em ontologias poderia ser usada para localizar os serviços automaticamente.

Invocação automática de serviços - a invocação automática de um serviço *Web* por um programa de computador ou agente de *software*, dada apenas uma descrição declarativa deste serviço, diferente do caso em que os agentes são pré-programados para invocar um serviço em particular. Por exemplo, isto é necessário, para que um usuário possa fazer uma compra, em um site encontrado por este usuário, de uma passagem aérea de um voo particular. Serviços *Web* Semânticos possuem descrições semânticas dos parâmetros usados na execução desse serviço, bem como dos resultados dessa execução, tendo ela obtido sucesso ou falha. Um agente de *software* poderá interpretar estas descrições semânticas dos parâmetros de entrada necessários para invocar o serviço, bem como da informação que será retornada, desta forma sendo possível invocar este serviço de forma transparente.

Composição automática de serviços e interação - esta tarefa envolve a seleção, composição e interação de serviços *Web* para realizar uma tarefa complexa, a partir de uma descrição de alto nível de um objetivo. Por exemplo, o usuário pode querer fazer todos os arranjos envolvidos na realização de uma viagem a uma conferência. Na maneira tradicional, o usuário deve selecionar os serviços, especificar a composição manualmente, e ter certeza de que qualquer *software* necessário para a interação entre serviços, que deverão trocar informações, e está bem configurado. Com os Serviços *Web* Semânticos, as informações necessárias para selecionar e compor serviços pode ser encontrada nas páginas dos provedores de serviços. Um programa ou um agente de*software* pode ser construído para manipular estas informações, junto com as especificações dos objetivos da tarefa, para executar a tarefa automaticamente.

Em Domingue et al. [6] são identificadas as principais tarefas e conceitos tipicamente utilizados e mencionados no campo de estudo de Serviços *Web* Semânticos (SWS), tais como: *Crawling*, Descoberta, *Matching*, Classificação, Seleção, Composição, Orquestração, Coreografia, *Mediation*, Invocação, *Grounding*, *Lifting* e *Lowering*. As definições aqui exposta

estão de acordo com os conceitos apresentados em Domingue et al. [6].

Crawling - está tarefa é encarregada de navegar na *Web* para localizar os serviços existentes. Essa tarefa é essencialmente idêntica a qualquer outro *Web Crawling* com a diferença de que as informações solicitadas não são páginas da Web, mas sim arquivos WSDL ou, mais recentemente, páginas HTML que descrevem APIs da *Web*.

Descoberta - envolve a localização de serviços que são capazes de cumprir determinados requisições do usuário. Essa tarefa envolve, a partir de definições abstratas das necessidades do usuário interpretar estas exigências de tal que possam ser usados para identificar os serviços da Web que podem posteriormente ser utilizados para fornecer as necessidades do usuário.

Matching - o *Matching*, também conhecido como *Matchmaking*, é a tarefa que, dado um pedido para algum tipo de serviço, ele tenta identificar os serviços *Web* que correspondem a um certo grau de semelhança com o pedido. A pesquisa em Serviços *Web* Semânticos *Matchmaking* tem dedicado esforços substancias para formalizar a funcionalidades de serviços *Web* usando raciocinadores. Em geral, a funcionalidade do serviço da *Web* é especificado em termos de entradas, saídas, pré-condições e efeitos.

Classificação *Ranking* - está tarefa que, dado um conjunto de serviços *Web* obtidos a partir do processo de *Matchmaking*, classifica as diferenças *Matchmaking* de acordo com o conjunto de preferências. Essas preferências são normalmente dadas em tempo de invocação do serviço e são especificadas de acordo com as propriedades não-funcionais dos serviços *Web*.

Seleção - é a tarefa em que tendo obtido uma lista com possíveis serviços, um destes é selecionado para ser invocado. Esta atividade é frequentemente realizada por humanos mas existem sistemas que realizam este passo automaticamente baseando-se na classificação realizada anteriormente.

Composição - é a tarefa responsável de combinar serviços *Web*, a fim de realizar uma tarefa complexa. Normalmente, essa tarefa é acionada sempre que o sistema não consegue encontrar um serviço da *Web* que cumpra todos os requisitos. O resultado de uma tarefa de composição é uma Orquestração de serviços que, dadas algumas condições iniciais e um conjunto de services *Web*, atenderia os requisitos do usuário quando executado. A maioria dos trabalhos na composição automatizada de Services *Web* Semânticos tem sido abordada

como uma tarefa de planejamento, que acontece a partir da especificação formal de entradas, saídas, pré-condições e efeitos de services *Web* para gerar orquestrações adequadas.

Orquestração - define a sequência e as condições para a execução de serviços Web, a fim de realizar um objetivo complexo através da combinação das funcionalidades fornecidas pelos serviços *Web* existentes. Orquestrações incluem definições de fluxo de dados (isto é, como os dados são propagados e utilizados ao longo do processo) e fluxo de controlo (isto é, quando uma determinada atividade deve ser executada).

Coreografia - descreve as interações dos serviços com seus usuários, segundo a qualquer cliente de um serviço da *Web*, pode ser uma máquina ou humano, é considerado um usuário. Uma coreografia define o comportamento esperado de um serviço da *Web*, ou seja, as mensagens trocadas na sua execução, do ponto de vista de um cliente. A interpretação coreográfica leva a uma invocação bem-sucedida de um serviço da *Web*independentemente de como a execução do serviço *Web* é realizada internamente.

Mediação *Mediation* - a mediação é necessária em ambientes onde a presença de componentes heterogêneos e frequente. Heterogeneidade é uma das principais características da *Web* e, portanto, qualquer solução orientada para a *Web* deve enfrentar esta questão de uma forma ou de outra. A mediação é um princípio pelo qual um elemento intermediário, um mediador, é introduzido entre dois elementos para resolver suas heterogeneidades sem ter que adaptar um ou outro.

Invocação - diz respeito a chamada real de uma operação de um serviço *Web*. A invocação é, portanto, intimamente relacionada com as coreografias que especificam uma ordem para a execução de um conjunto de invocações.

Grounding - especifica como determinadas atividades são mapeadas em operações em nível com o serviço. A necessidade de especificação do *Grounding* vem do fato de que os Serviços *Web* Semânticos são essencialmente manipulados no nível semânticos onde os detalhes de invocação são frequentemente ignorados. Na maioria das abordagens, as definições de *Grounding* são basicamente ponteiros para definições de operação existentes.

Lifting - refere-se à transformação de informações de sua representação no nível sintático usado pelo serviço da *Web* em sua contraparte semântica. A *Lifting* é expressa geralmente declarativa de modo que possa diretamente ser interpretada por algum motor a fim transformar os dados em alguma representação semântica.

Lowering - refere-se à tarefa que toma as informações representadas semanticamente e as transforma em alguma representação sintática que pode ser usada para se comunicar com o serviço da *Web*. Esta tarefa é o inverso de *Lifting*, e da mesma forma é frequentemente abordado com XSLT (*eXtensible Stylesheet Language for Transformation*).

2.3.4 Modelos Conceituais Semânticos

O foco de trabalhos em Serviços *Web* Semânticos (SWS) são as descrições semânticas de serviços que oferecem suporte a automatização de tarefas, tais como: descoberta automática de serviços, descoberta e composição de serviços, invocação de composição de serviços e invocação de serviços. Neste contexto, ao longo dos anos a comunidade da área de SWS têm investido na criação de modelos conceituais para prover suporte a descrição de SWS. Assim, está seção tem por objetivo apresentar os detalhes das principais abordagens proposta pela a comunidade para descrição de SWS.

OWL-S

A *Semantic Markup for Web Services* (OWL-S) é uma ontologia de serviços *Web* construída pelo programa DARPA *Agent Markup Language* (DAML) baseada na OWL *(Web Ontology Language)*. A OWL-S (anterimento DAML-S) tem por objetivo fornecer um conjunto de conceitos para descrever as propriedades e os recursos de serviços Web de forma inequívoca. De com Martin et al. [22] ontologia de topo OWL-S foi projetado com o propósito de automatizar tarefas que são consideradas complexas na área de serviços *Web* Semânticos, tais como: descoberta automática de serviços, descoberta e composição de serviços, composição e monitoramento de execução e invocação de serviços.

Entretanto, por ser uma submissão da W3C a ontologia de serviços OWL-S tem despertado forte interesse da comunidade de Serviços Web Semânticos (SWS). Nesta perspectiva, a comunidade vem contribuindo com o desenvolvimento de ferramentas e algoritmos que prover a descoberta, composição e invocação automática de serviços. Desta forma, diferentes de outras abordagens propostas para descrição de serviços a OWL-S prover efetividade na descrição de serviços semanticamente, assim garantindo mais interoperabilidade. De acordo com Martin et al. [22] a estruturação de uma ontologia de serviços é motivada pela a neces-

sidade de três tipos essenciais de conhecimento básico sobre um serviço, o que poderia ser caracterizado pela a resposta a três perguntas, são:

- O que o serviço faz?
- Como o serviço trabalha?
- Como acessar o serviço?

Ainda nesta perspectiva, as respostas para às três perguntas definidas por Martin et al. [22] habilita com que um agente de *software* seja capaz de identificar: qual a interface e objetivo do serviço, quais são as etapas realizadas em sua execução e quais são os protocolos necessários para sua invocação. Neste contexto, conforme é exibido na Figura 2.3, a *Semantic Markup for Web Services* (OWL-S) foi construída baseada em três componentes principais, são:

- Um perfil *(Profile)* apresenta qual é o objetivo do serviço, qual a funionalidade do serviço, o que serviços provê;
- Um modelo de serviço *(ServiceModel)* apresenta um maior detalhamento do funcionamento do serviço e;
- Um *grounding* fornece detalhes específicos de como um agente de *software* pode invocar um serviço.

Por outro lado, como pode ser visto graficamente na Figura 2.3, os relacionamentos entre todas as entidades definidas na OWL-S representa um modelo abstrato de um Serviço *Web* Semântico. Este modelo abstrato contem as informações dos serviços e detalhamentos de suas funcionalidades.

Percebe-se que a *Semantic Markup for Web Services* (OWL-S) é vista pela comunidade como um linguagem que possui um vocabulário para representação de serviços semanticamente. Além disso, a OWL-S force conceitos que têm por objetivo fornecer uma descrição semântica que possibilitem que agentes de softwares façam inferências e sejam capazes de tomar decisões em ambientes automatizados.

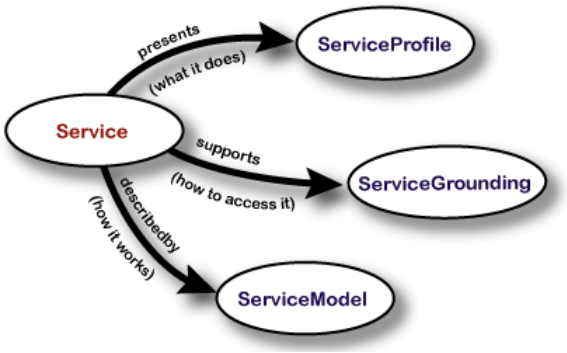

Figura 2.3: Estrutura proposta de um serviço semântico definida em OWL-S imagem extraída de Martin et al. (2004)

WSMO

A *Web Service Modeling Ontology* (WSMO) é uma submissão da W3C de uma ontologia que tem por objetivo fornecer uma estrutura conceitual e uma linguagem formal para descrever todos os aspectos relevantes dos serviços, a fim de prover a automatização ”total ou parcial”de tarefas de descoberta, composição e execução de acompanhamento dos serviços *Web*, proposta por Roman et al. [30]. A ontologia WSMO é baseada no WSMF (Web Service Modeling Framework) proposto em Fensel et al. [10].

Desta forma, seguindo os principais elementos identificados na WSMF, a WSMO consistem em quatro elementos de alto nível com os principais conceitos, são: ontologias (*Ontologies*) , serviços web (*Web services*), objetivos (*Goals*) e mediadores (*Mediators*) como pode ser visto na Figura 2.4.

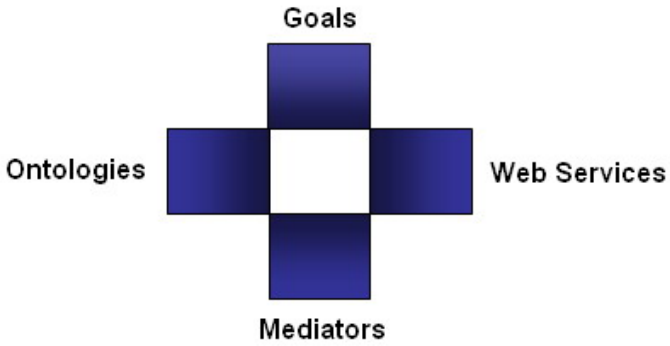

Figura 2.4: Os elementos de alto nível do WSMO imagem estraída de Roman et al. (2005)

Ontologias - fornecer a terminologia usada por outros elementos do WSMO para des-

crever os aspectos relevantes dos domínios em questão.

Objetivos - Representa a perspectiva dos clientes, para o qual o cumprimento dessas perspectivas é através de execução de funcionalidades oferecida por serviços. Essas perspectivas podem ser descrições de serviços que potencialmente satisfariam os desejos do usuário.

Serviços *Web* - descreve a entidade computacional que permite acesso a serviços que proveem algum valor ao domínio. Essas descrições incluem os recursos, interfaces e funcionamento interno do serviço da *Web*. Todos esses aspectos de um service são descritos usando a terminologias definidas pelas ontologias.

Mediadores - descrevem elementos que lidam com problemas de interoperabilidade entre diferentes elementos da WSMO. Os mediadores são conceitos centralizados para resolver incompatibilidade com os níveis de dados, processos de protocolos, ou seja, para resolver os desajustes entre diferentes terminologias. Por exemplo: comunicação entre serviços a nível de combinação de serviços.

Portanto, a WSMO é uma ontologia que utiliza uma linguagem formal para descrever os serviços. Desta forma, percebe-se que os conceitos semânticos apresentados nessa ontologia atende parcialmente a descrição semântica de processos que visam automatizar a descoberta, composição e invocação automática de Serviços *Web* Semânticos.

Modelo proposto por Barros [7]

Na Figura 2.5 é apresenta a representação gráfica da ontologia de serviços adaptada por Barros [7]. Este modelo de representação semântica de serviços é contribuição da tese de doutorado de Barros [7]. Esta ontologia de serviços foi criada a partir da adaptação da linguagem *OWL-S* visando a representação semântica de serviços. Para construção dessa ontologia de serviços, foi considerado o subconjunto das descrições presentes na *OWL-S* com a finalidade de prover compatibilidade com a mesma e adicionando um suporte a especificidades dos serviços (suporte a troca de mensagens *JSON do ingles JavaScript Object Notation*).

A Figura 2.5 exibe as entidades que fazem parte da ontologia de serviços, com seus respectivos atributos e relacionamentos. A classe *Service* representa os serviços *Web*. Está classe tem como atributos um rótulo (*label*), descrição (*descrition*) e localização (*location*). Para cada serviço, é definido seu conjunto de parâmetro (*Parameters*), que podem ser de entrada (*Input*) ou de saída (*Output*).

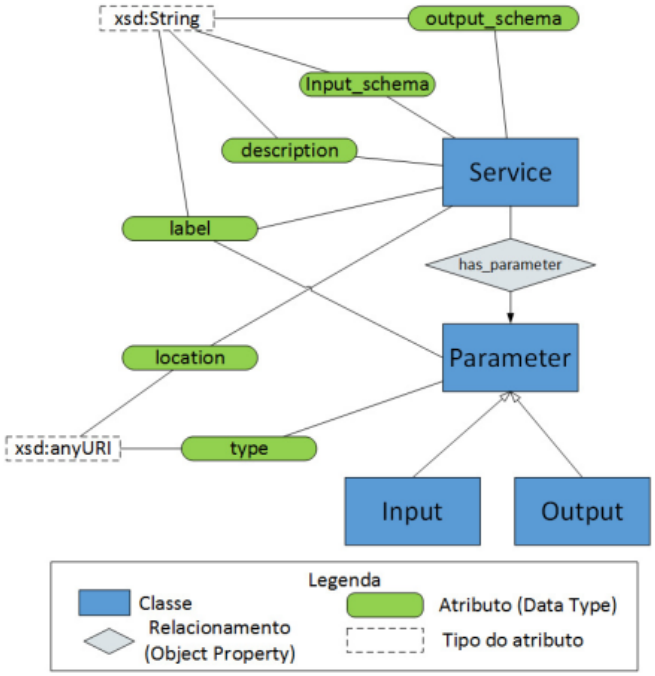

Figura 2.5: Representação gráfica da Ontologia de serviços adaptada por Barros 2016

Nesta ontologia de serviços é definido um *schema* para os parâmetros de entrada e outro para os parâmetros de saída do serviço. Estes *schemas* definem o formato que os parâmetros de entrada devem ser enviados aos serviços e como os parâmetros de saída dos serviços serão retornados. Como pode ser visto, estes *schemas* seguem uma definição baseada no *JSOM Schema*.

Discussão

Uma das principais decisões no desenvolvimento de aplicações baseadas em Serviços *Web* Semânticos é a definição do modelo semântico adotado para realizar a descrição semânticas dos serviços. Neste contexto, como apresentado nas seções anteriores, existem diferentes abordagens para prover descrição semântica de serviços. No entanto, no desenvolvimento do *framework* proposto neste trabalho, foi adotada a abordagem semântica proposta por Barros [7] devido fornecer algumas vantagens frente as demais:

- Baixa complexidade para criação de serviços;
- Baseado no paradigma *REST* permitindo que os serviços se comuniquem por mensagens do *JSON*;

- Flexibilização de parâmetros de entrada e saída, afim de permitir a passagem de vários tipos de arquivos, como: *JSON*, dentre outros;

- Possibilitar a construção de serviços *Web*, com anotações semânticas que atendam a representação de diferentes tipos parâmetros, sendo tipos simples ou complexos, de entrada e saída dos algoritmos de Mineração de Dados;

- Provê maior desempenho no processo de descoberta, composição e invocação automática dos serviços a fim de permitir que aplicações possam realizar o processo em tempo de execução;

Baixa complexidade para descrever serviços semanticamente. Com base nas abordagens apresentadas anteriormente, a *OWL-S* é a principal linguagem utilizada para descrição de serviço, no entanto, fazer a representação semântica de serviços torna-se um processo complexo, pois a ontologia de serviços é dividida em três sub-ontologias (*ServiceProfile, ServiceGrounding e ServiceModel*) e cada sub-ontologia representa uma parte das informações do serviço. Neste contexto, a proposta de Barros [7] atende as propriedades da ontologia *OWL-S* e possibilita com mais facilidade que o usuário faça a descrição semântica dos parâmetro de entrada e de saída de serviços sem a necessidade de realizar a especificação desses parâmetros nas classes serviços definia em cada uma das sub-ontologias da *OWL-S*.

Suporte a troca de mensagens no formato *JSOM*. Com base nas abordagens apresentadas anteriormente, este suporte de trocas de mensagens no formato *JSON* não é implementado em nenhuma das abordagens. Neste contexto, este recurso é relativamente importante para o desenvolvimento de aplicações que fazem uso de serviços Web semânticos baseado no paradigma *REST*. Assim, este mecanismo de suporte a mensagens do tipo *JSON* possibilita que serviços *Web* semânticos utilizem mensagens no formato *JSON* para se comunicar com outros serviços *Web*. Por outro lado, a manipulação de arquivos do tipo *JSON* é relativamente simples, além disso, mensagens no formato *JSON* são consideradas mais leves, ou seja, podendo ser rapidamente geradas e analisadas por aplicações que fazem uso de serviços *Web* semânticos.

Desempenho Prezando pelo o desempenho em tempo de requisição na execução do processo de descoberta, composição e invocação de serviços *Web* semânticos na abordagem proposta neste trabalho. Verificou-se nas abordagens apresentas anteriormente, que os

serviços *Web* semânticos utilizam mensagens no formato *XML* para se comunicar com outros serviços, ou seja, todas as requisições enviadas por aplicações clientes utilizam dados no padrão *XML*, assim, a desvantagem no *XML* é a sobrecarga na transmissão dos dados e o tempo de processamento desses dados pelos os serviços que foram selecionados. Neste contexto, o uso dessas abordagens no processo de descoberta e composição de serviços requer custo no tempo de resposta para o usuário. Assim, nesta perspectiva, visando resposta para o usuário em tempo hábil, os serviços *Web* semânticos descritos com abordagem adaptada de Barros [7] utilizam mensagens do tipo *JSON*, ou seja, o tempo de resposta para uma requisição do usuário é relativamente inferior em relação aos serviços que utilizam mensagens no padrão *XML*.

2.4 Mineração de Dados

Para Fayyad et al. [8] a Mineração de Dados do inglês "Data Mining" é um processo de Descoberta de Conhecimento em Base de Dados (KDD) do inglês "knowledge-discovery in databases", que consiste na aplicação da análise de dados e algoritmos de descoberta, produzindo uma enumeração de padrões ou modelos sobre os dados.

Entretanto, conforme Kurgan et al. [20] o termo "Mineração de Dados" tornou-se comum e vem sendo muito utilizado pela comunidade. Para muitos pesquisadores, o termo "Mineração de Dados" é usado como um sinônimo para Descoberta de Conhecimento, além de está sendo utilizado para descrever um dos processos de Descoberta de Conhecimento.

Segundo Fayyad et al. [9] termo "Mineração de Dados" também é conhecido por muitos outros nomes, tais como: including knowledge extraction, information discovery, information harvesting, data archeology e data pattern processing.

Para este trabalho, será utilizado o termo "Mineração de Dados" para se referir ao processo de extração de conhecimento, além de também e a etapa de mineração do referido processo por ser um termo comum e utilizado pelos pesquisadores.

2.4.1 Processo de Descoberta de Conhecimento

O processo de extração de conhecimento requer a execução de diversas etapas. No entanto, não existem um consenso sobre o número de etapas e suas nomenclaturas. Neste contexto,

Kurgan et al. [20] realizam um levantamento histórico e apontam os principais modelos de processos e o modelo escolhido e utilizado neste trabalho será o modelo proposto por Fayyad et al. [9] exibido na Figura 2.6.

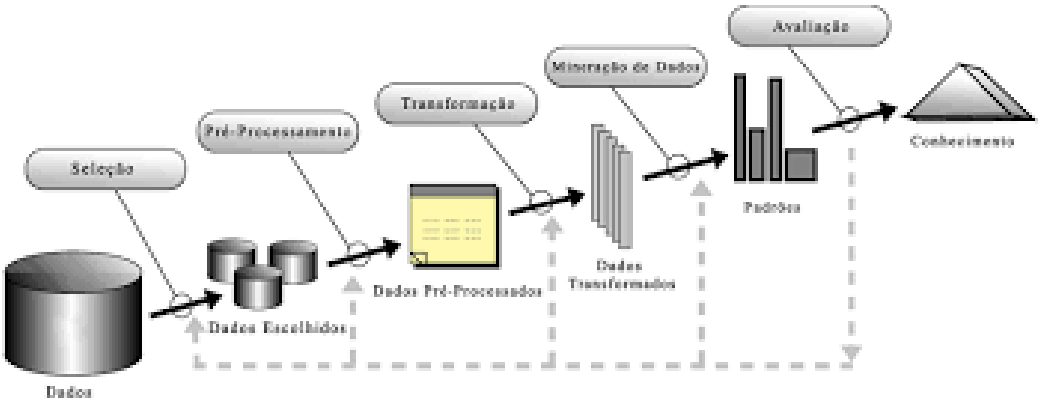

Figura 2.6: Arquitetura de serviços de Mineração de Dados. Fonte Fayyad et al. [9].

O processo consiste nos seguintes passos: seleção, pré-processamento, transformação, mineração de dados e avaliação. A etapa de seleção envolve a seleção de dados relevantes para a análise compreendida. A etapa de pré-processamento é realizada a limpeza dos dados para remover os ruídos e inconsistências dos dados. A etapa de transformação os dados para formato apropriado para que os algoritmos de mineração sejam aplicados. A etapa de mineração de dados é considerada a principal etapa do processo de descoberta de conhecimento, onde os dados são analisados em busca de padrões relevantes. E por fim, a etapa de avaliação, nesta etapa são interpretados os padrões e avaliações dos padrões encontrados.

Capítulo 3

Trabalhos Relacionados

"O primeiro passo para conseguirmos o que queremos na vida é decidirmos o que queremos."

- Ben Stein

É apresentado neste Capítulo os principais trabalhos relacionados encontrados na literatura a fim de posicionar a solução proposta frente ao estado da arte. Desse modo, são apresentadas cada uma das soluções propostas, incluindo suas principais características e limitações. Por fim, são sumarizadas as características consideradas primordiais a fim de anteder as necessidades e desafios levantados inicialmente neste trabalho e sua relação entre os trabalhos relacionados aqui apresentados e a solução proposta.

3.1 A service oriented architecture to provide data mining services fornon-expert data miners

O trabalho de Zorrilla et al. [45] propõe uma arquitetura orientada a serviços para o domínio de mineração de dados com a finalidade de atender as necessidades de usuários e/ou mineradores de dados não especialistas em mineração. Para atender as necessidades desses usuários com prévio conhecimento técnico na área de mineração de dados, os autores, Zorrilla et al. [45], fornecem quatro serviços *Web* que encapsulam alguns dos algoritmos de mineração SimplesKmeans, Xmeans, EM e Apriori fornecidos na ferramenta Weka:. A descrição dos dados e informações sobre os serviços é feita com a WSDL (*Web Services Description Lan-*

guage). A fim de gerenciar tarefas que envolvem execução e invocação de serviços, Zorrilla et al. [45], adotaram o processo de coreografia de serviços. Neste trabalho, Zorrilla et al. [45] definiram um conjunto de modelos para os usuários trabalhar com os serviços de mineração. Os modelos contêm as definições dos atributos e algoritmos de mineração que são adequados para obter melhores padrões de mineração. Assim, para o usuário utilizar os serviços de mineração é necessário fornecer os dados requisitados como parâmetros de entrada nos modelos que estão vinculados aos serviços. Em relação ao formato de dados definido como parâmetro de entrada dos algoritmos de mineração do Weka, Zorrilla et al. [45] ressalta que independente do tipo de arquivo, a solução oferece funções para fazer a transformação dos dados em formatos que sejam válidos.

A Figura 3.1, exibe a arquitetura definida por Zorrilla et al. [45] para este trabalho, assim como, suas principais camadas: *presentation*, *business process choreography*, *services*, *enterprise components*, e *data layer*. A seguir, são apresentadas as descrições das camadas conforme os conceitos definidos em [45].

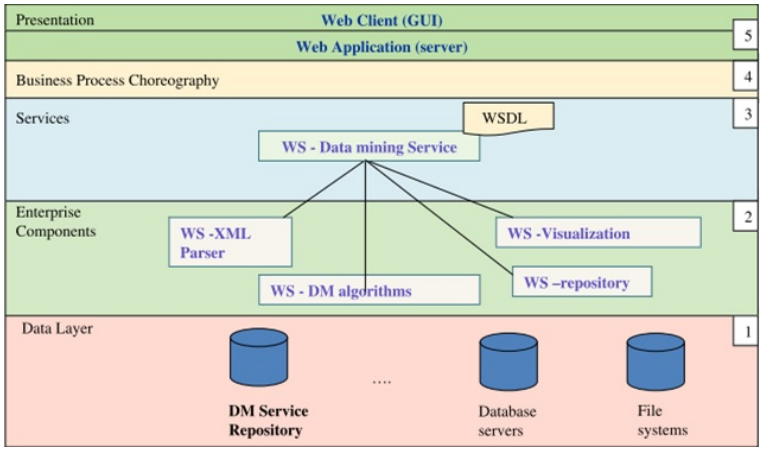

Figura 3.1: Arquitetura de serviços de Mineração de Dados.

Presentation: fornece uma interface que permite a conexão de aplicações com os serviços.

Business Process Choreography: processo de negociação dos serviços fornecidos na camada *Services*.

Services - esta camada fornece acesso aos serviços que podem ser usados por outras aplicações. Os serviços estão descritos em WSDL (*"Web Services Description Language"*).

Enterprise Components - nesta camada encontra-se os serviços que fornecem as funci-

onalidades para a camada *Services*: *WS-XML Parser*, *WS-DM algoritms*, *WS-Repository*, e *WS-Visualization*.

- **WS-XML *parser*** - este serviço realiza a validação de dados enviado no formato XML *"eXtensible Markup Language"* e os transforma em formatos aceitos pelos os algoritmos de mineração.
- **WS-DM *algoritms*** - este serviço fornece o encapsulamento dos algoritmos de mineração SimpleKmeans, Xmeans, EM e Apriori.
- **WS-*repository*** - este serviço permite o acesso ao repositório de dados.
- **WS-*visualization*** - fornece gráficos para exibir os resultados.

Data Layer - nesta camada estão definidos o repositório de serviços, e também, os repositórios de que armazenam os dados que serão processados pelos serviços.

Diferente da abordagem proposta neste trabalho, o trabalho de Zorilla et al. [45] possui algumas limitações: (i) Por trata-se de uma aplicação desenvolvida para usuário não especialistas em mineração de dados a proposta de certa forma exige conhecimento técnico do usuário devido fornecer modelos para os usuários fornecer informações que serão processadas pelos serviços. Por exemplo, como o usuário com pouco conhecimento ou até mesmo, nenhum conhecimento em mineração, pode identificar quais atributos da base de dados pode contribuir para geração de resultados mais precisos; (ii) A solução fornece apenas quatro serviços de mineração que encapsulam os algoritmos SimpleKmeans, Xmeans, EM e o Apriori da ferramenta Weka. Por exemplo, dado que os usuários precisam utilizar outros serviços de mineração que encapsulam as funcionalidades de classificadores de algoritmos *associations*, *clusteres*, *trees*, *rules*, *music*, *lazy*, *funtions* e *bayes* etc, não é possível, pois o trabalho limita-se apenas aos serviços que foram implementados; (iii) Composição de serviços é um processo onde a aplicação busca, seleciona, combina e executa serviços para atender a requisições de usuários que necessitam de funcionalidades mais complexas para um domínio específico, entretanto, este trabalho não realiza composição de serviços de mineração. Além disso, foi adotada a técnica de coreografia de serviços. A coreografia de serviços em relação a técnica de orquestração, não fornece flexibilidade para adicionar novos serviços de mineração, ou seja, incluir novos serviços torna-se uma tarefa complexa, pois

existe a necessidade do usuário conhecer as especificações dos parâmetros de entrada e saída dos serviços, e a função desempenhada, para que seja possível adicionar novos serviços. Por outro lado, dado que um serviço de mineração foi removido e/ou modificado, e o mesmo, fornece funcionalidades que são importantes para o funcionamento dos demais serviços, isto é, na coreografia de serviços existe a dependência de serviço e de cordo com Saudate et al. [34], então realizar este tipo de operação pode gerar problemas que possivelmente, refletirá nas funcionalidades disponibilizadas pelos demais. Por fim, outra característica do processo de coreografia é a descentralização de serviços da aplicação, isso do ponto de vista de escalabilidade é considerado ganho, entretanto, do ponto de vista de manutenibilidade em relação coreografia é entendido como desvantagem Saudate et al. [34]. Por exemplo, dado que um determinado serviço está com problemas e/ou produzindo resultados inconsistentes, o custo para corrigir as falhas deste não será trivial devido encontra-se descentralizado; (iv) Baixo desempenho devido os serviços utilizar o protocolo SOAP (Simple Object Access Protocol), pois este protocolo usa o XML (eXtensible Markup Language) para troca de mensagens. Conforme Senagi et al. [35], Voigt et al. [40], Minaei et al. [26], Mumbaikar et al. [27] a desvantagem do XML está relacionada ao tempo total que um serviço leva para realizar o processamento de dados contidos em arquivos com essa extensão.

3.2 Um *framework* para mineração de dados educacionais baseado em serviços Web semânticos

Marinho [39] propõe um *framework* baseado em Serviços *Web* Semânticos para prover mineração de dados com enfoque educacional. A principal contribuição deste trabalho está direcionada a flexibilidade de integração do *framework* em diferentes ambientes *e-Learning* através do uso de Serviços *Web* Semânticos. Este *framework* é composto por Serviços que encapsulam algoritmos de pré-processamento, algoritmos de mineração e pós-processamento de dados. O processo de composição de serviços é realizado de forma automática, assim possibilitando aos seus usuários facilidades no uso de algoritmos e técnicas de mineração de dados com fins educacionais.

Alguns benefícios proporcionados por esta solução:

- **Técnicas de mineração**: possui técnicas de mineração implementadas, com isso os ambientes não necessitarão mais implementá-las para realizar mineração de dados (e.g. Regras de associação);

- **Serviços educacionais**: disponibilizará serviços educacionais aos ambientes, provendo a essas ambientes funcionalidades da mineração de dados com enfoque educacional (e.g. Serviço de recomendação de materiais);

- **Automatização dos processos**: irá prover a composição, descoberta e invocação de serviços de forma dinâmica e automática.

A Figura 3.2 mostra a visão lógica da arquitetura desenvolvida por Souza [39], incluindo seus principais componentes. Como pode ser visto, o projeto adota o modelo de Arquitetura em Camadas, assim separando o controle da aplicação e os dados armazenados. Nota-se que a camada de controle interna do *framework* é baseia-se no modelo de Arquitetura orientada a Serviços. A arquitetura possuí os seguintes elementos: *Mining Algorithms*, *I/O Adapter*, *Service Adapter*, *BD Adapter*, *Log Adapter*, *Web Services*, *Preprocessing Services*, *Educational Services*, *Service Manager* e *Ontology*. A seguir, cada um dos elementos presentes na arquitetura é apresentado em maiores detalhes.

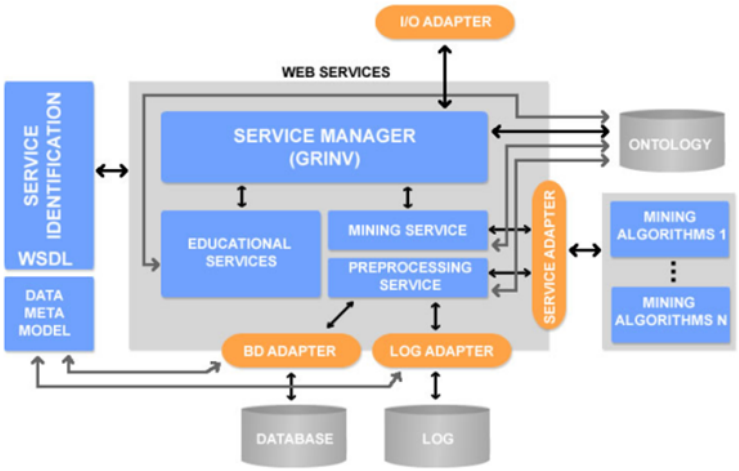

Figura 3.2: Visão lógica

Mining Algorithms - Esta camada é composta por vários algoritmos e técnicas de mineração de dados que serão utilizados para atingir os objetivos da mineração com enfoque educacional. Os algoritmos e técnicas de mineração são encapsulados em Serviços Web,

assim possibilitando que suas funcionalidades sejam utilizadas pelos os demais camadas da arquitetura.

Adapter - A arquitetura é composta por quatro adaptadores que executam funcionalidades distintas entre si.

- ***I/O Adapter*** - Define o padrão para comunicação com os ambientes *E-Learning*. A entrada aceita para comunicação com o *framework* é um par (chave, valor), que representa a URI da ontologia e o seu valor.

- ***Service Adapter*** - Fornece uma abstração do encapsulamento de algoritmos de mineração disponibilizados pelas as ferramentas de mineração. Assim, cada serviço encapsula um algoritmo especifico de mineração, possibilitando que outras camadas possam descobri, compor e invocar os serviços de mineração de forma automática de dinâmica, realizando a combinação de técnicas e algoritmos de mineração de acordo com as necessidades da aplicação.

- ***BD Adapter*** - Possibilita que desenvolvedores de ambientes *E-Learning* utilizem o *framework* independentemente de como suas fontes de dados foram modeladas. Para isso, o usuário do *framework* deve implementar a interface definida pelo *Data Meta Model*. Esta interface define os formatos de dados aceitos pelo o *framework*.

- ***Log Adapter*** - Apresenta função semelhante ao *DB Adapter*, permite que desenvolvedores de ambientes *E-Learning* utilizem o*framework* para minerar os *logs* gerados pelos os seus ambientes em busca de informações educacionais relevantes.

Web Services - Esta camada é composta pelos os serviços Web que encapsulam as funcionalidades disponibilizadas pelo o framework, tais como: gerenciador de serviços, algoritmos de pré-processamento e pós-processamento.

Preprocessing Services - Fornece serviços que encapsulam algoritmos de pré-processamento disponibilizados pela camada *mining algorithms*. Esta camada fornece funcionalidades para preparação e tratamentos dos dados que serão utilizados para descoberta de conhecimento. Alguns serviços disponíveis nesta camada: ListToARFF, StringToRule entre outros.

Mining Services - Esta camada fornece uma abstração dos requisitos implementados nas ferramentas de mineração em serviços Web. Através desta camada, os ambientes *E-Learning* não precisam implementar os algoritmos de mineração, bastando apenas utilizar os serviços disponíveis no *framework*. Alguns serviços disponíveis nesta camada: Apriori, K-Means, Naive Bayes entre outros.

Educational Services - Esta camada é composta por serviços Web semânticos educacionais, que proveem as funcionalidades educacionais presentes no*framework*. Através desta camada, torna-se possível atingir os objetivos da mineração de dados com enfoque educacional, tais como: recomendação de conteúdo, classificação de alunos, recomendação de link baseado *Web* baseado em regras entre outras. Para tal, prover a descoberta de conhecimento com enfoque educacional, os serviços educacionais processam os resultados obtidos dos serviços de mineração que compõe a camada *Mining Services*.

Service Manager - Está camada é composta apenas por um único serviço *Web*. Este serviço prover a comunicação entre a aplicação cliente e os serviços fornecidos pelo *framework*. Assim, provendo que aplicações tenham acesso a todas as funcionalidades disponibilizadas pelo *framework*. Para construção deste serviço, foi utilizado o *middleware* grinv. Este *middleware* disponibiliza algoritmos e técnicas que permite realizar a descoberta, composição e invocação automática dos serviços de mineração de dados fornecidos pelas camadas do *framework*.

Ontology - Descreve as tarefas e técnicas de mineração de dados, especificando os parâmetros de entrada e saída dos algoritmos de mineração.

Para a abordagem proposta neste trabalho, poderíamos adotar o trabalho de [39] devido à semelhança da nossa abordagem com as funcionalidades providas pelo mesmo. Entretanto, podemos destacar algumas limitações: (i) Complexidade de criar de novos serviços que encapsulam técnicas e algoritmos de mineração de outras ferramentas. Pois, existe a necessidade de utilizar a ontologia OWL-S para descrever o serviço, bem como, as três subclasses *ServiceProfile*, *ServiceModel* e *ServiceGrounding*. Entretanto, essas três classes funcionam de forma diferente, por exemplo, na classe *ServiceProfile* são definidas as funcionalidades do serviço; a classe *ServiceModel* exibe um detalhamento de como o usuário pode usar o serviço e como ele funciona; e no *ServiceGrounding* fornece os detalhes de como o agente de *software* podem invocar. Ainda dentro desse contexto, [39] afirma que a principal difi-

culdade para adicionar novos serviços no *framework* esta relacionada ao mapeamento que precisa ser feito entre os serviços e as ontologias. (ii) Uso de outra ferramenta para realizar o processo de descoberta, composição e invocação de serviços, o *middleware* GRINV proposto em [1]. Pois, como destacado anteriormente na seção **??**, algumas das limitações do GRINV, estão relacionadas com a complexidade para criação de serviços, adicionar serviços baseados em novos modelos de descrição semântica de serviços, restrito com serviços desenvolvido seguindo o REST (*"Representational State Transfer"*), e custo no tempo de resposta relacionado ao processo de descoberta e composição de serviços. Apesar do trabalho de [39] apresentar uma forte contribuição conforme a literatura, o mesmo, pode sofre impactos devido não suportar a integração de tecnologias , por exemplo, REST etc.

3.3 Um Middleware adaptável para descoberta, composição e invocação automática de Serviços Web Semântico

Barros [1] propõe um *middleware* para prover o uso de Serviços Web Semânticos para aplicações, através dos processos de descoberta, composição e invocação automática de serviços, chamado de *GRINV*. O *Grinv* tem como principal vantagem a possibilidade de extensão e adaptação da solução proposta de acordo com as necessidades de seus usuários/desenvolvedores. Os principais pontos de variabilidade do *Grinv* são apresentados abaixo [1]:

- O usuário pode adicionar técnicas para realizar os processos de descoberta e composição de serviços de acordo com as características e necessidades da aplicação;
- O uso de um conjunto de técnicas para descoberta e composição de serviços no atendimento cada requisição, aumentando assim a possibilidade de se obter sucesso na busca por um serviço;
- Criação de novas estruturas para descrição de controladores para execução de serviços compostos;

- Desenvolvimento de mecanismos para realizar a invocação dos serviços que forneçam informações ou transformações necessárias para a aplicação;
- Adição de Restrições com o objetivo de verificar os resultados fornecidos pelos serviços que a aplicação faz uso;
- Uso de diferentes abordagens para descrição de Planejamento estático de composições de serviços.

Como apresentado na Figura 3.3, A arquitetura conceitual da solução desenvolvida por Barros [1] é composta por cinco módulos: *Request Manager*, *Discovery Controller*, *Inovocation Controller*, *Repository Manager* e *Configuration*. A seguir, os módulos que compõem a arquiteutra são apresentados em maiores detalhes.

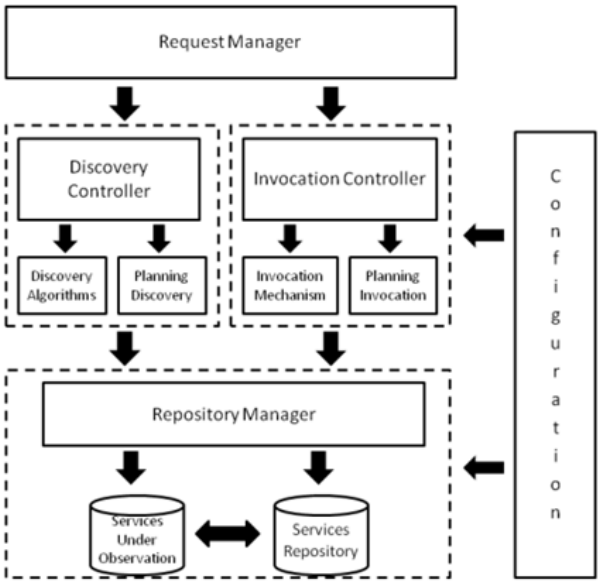

Figura 3.3: Projeto Arquitetural do Middleware.

Request Manager - É o núcleo do *middleware*. Ele é responsável por indicar o fluxo dos processos que devem ser realizados para atender à requisição do usuário.

Discovery Controller - Este módulo é responsável por encontrar o serviço presente no repositório que é mais similar às descrições enviadas pelo usuário. Este módulo implementa um conjunto de técnicas para descoberta de serviços que são utilizados para realizar a atividade de descoberta. Este módulo também é responsável por encontrar composições

de serviços de acordo com a necessidade da aplicação, através de planejamento estático e dinâmico.

Inovocation Controller - Este módulo é responsável pela execução dos serviços, sejam eles providos pela aplicação ou descobertos pelo *Discovery Controller*. Este módulo deve ser capaz de invocar serviços simples (um único serviço) e processos compostos de uma maneira transparente à aplicação.

Repository Manager - Este módulo tem a função de armazenar as descrições dos serviços que serão utilizados pelos demais módulos do *middleware*. Além disso, ele deve oferecer suporte a diferentes tecnologias para armazenamento das descrições dos serviços. O Gerenciador de Repositório tem acesso a dois tipos de repositórios: *Services Repository* (SR) e *Services Under Observation* (SUO). SR é responsável por armazenar os serviços que serão utilizados pelo sistema, enquanto o SUO armazena os serviços que apresentam problemas durante o processo de descoberta ou invocação.

Configuration - Este módulo é responsável por ajustar o sistema de acordo com as configurações feitas pelo desenvolvedor da aplicação. Assim, este módulo carrega os arquivos de configuração que indicam as técnicas para descoberta de serviços que serão utilizadas, o mecanismo de invocação de serviços e as políticas do repositório de serviços.

Para a abordagem proposta neste trabalho, poderíamos adotar o *middleware Grinv* devido à semelhança da nossa abordagem com as funcionalidades providas no mesmo. Entretanto, o mesmo apresenta algumas limitações: (i) adicionar serviços *Web* com descrições semânticas feitas com novos modelos semânticos. Este problema é causado devido a implementação do *middleware* ser totalmente amarrada a linguagem OWL-S. Por exemplo, dado que o usuário tenha desenvolvido um novo modelo para representação semântica de serviços e precise instanciar o *Grinv* para utilizar as funcionalidades para descobrir, compor e invocar de forma automática serviços semânticos baseado nas informações do modelo não será possível, pois a implementação das funcionalidades do *Grinv* estão presas as propriedades definidas na OWL-S. (ii) Complexidade de criação de serviços semânticos com OWL-S, isto é, a descrição de serviços de forma semântica com a OWL-S não é uma tarefa trivial. Pois, nesta ontologia, temos a classe *Service* que define mais outros três subclasses *ServiceProfile*, *ServiceModel* e *ServiceGrounding*. Então, para criação de novos serviços é necessário fazer a descrição e relações entre as classes. Entretanto, essas três classes funcionam de forma

diferente, por exemplo, na classe *ServiceProfile* são definidas as funcionalidades do serviço; a classe *ServiceModel* exibe um detalhamento de como o usuário pode usar o serviço e como ele funciona; e no *ServiceGrounding* fornece os detalhes de como os agentes de *software* podem invocar. (iii) Restrição com serviços REST, isto é, a definição do *Grounding* na OWL-S é feita para serviços que são baseados na linguagem XML para troca de dados. Entretanto, os serviços implementados na abordagem proposta adotam o paradigma REST e utilizam JSON para troca de mensagens durante a execução do processo de descoberta, composição e invocação de serviços. (iv) Tempo de resposta no processo de descoberta e composição de serviços, isto é, a OWL-S é uma linguagem baseada em XML. De acordo com Klusch et al. [19], Senagi et al. [35], Voigt et al. [40], Minaei et al. [26], Mumbaikar et al. [27] a desvantagem do XML está relacionada ao tempo de resposta que os serviços levam para realizar o processamento de dados e interagir com os demais serviços.

3.4 Comparação entre as soluções

Nesta seção é realizada uma análise comparativa entre as soluções relacionadas apresentadas nas seções 3.1, 3.2 e 3.3 e a abordagem proposta neste trabalho. Como destacado anteriormente, para o desenvolvimento da solução proposta foram selecionados os três trabalhos que possuíam maior relevância frente ao estado da arte com a aplicação proposta:

1. *A service oriented architecture to provide data mining services fornon-expert data miners.* Zorrilla et al. [45]
2. *Um framework para Mineração de Dados Educacionais Baseado em serviços Web Semânticos.* Souza [39]
3. *Um Middleware adaptável para descoberta, composição e invocação automática de Serviços Web Semântico.* Barros [1]

A fim de fazer uma análise comparativa das principais caraterísticas fornecidas nos trabalhos relacionados com a abordagem proposta foram definidos 8 critérios: *(i) - Altos Custos para Criação de SWS; (ii) - Processo Complexo para Manipulação de SWS; (iii) - Descoberta e Composição e Invocação Automática de Serviços; (iv) - Desempenho no Processo de Descoberta e Composição; (v) - Processo Complexo de Extensão da Aplicação; (vi) - APIs*

de manipulação dos modelos semânticos adotados pouco maduras, complexas e com baixo desempenho; (vii) - Dificuldade em representar semanticamente tipos abstratos de dados durante construção dos SWS; (viii) - Modelos Semânticos Complexos. As características foram classificadas em: **C** - Complexo, **F** - Fácil, **A** - Atende, **N** - Não atende, **A*** - Atende parcialmente, **A+** - Alto e **B** - Baixo. A Tabela 3.1 exibe o resultado da avaliação da abordagem proposta com os trabalhos.

Tabela 3.1: Comparação entre os trabalhos relacionados e a solução proposta.

Características	**1**	**2**	**3**	**Proposta**
Custos para criação de SWS	A+	A+	A+	B
Processo complexo para manipulação de SWS	A*	C	C	F
Descoberta e composição e invocação automática de serviços	N	A	A	A
Desempenho no processo de descoberta e composição	A*	B	B	A+
Processo complexo de extensão da aplicação	A*	C-A+	C-A+	F-B
APIs de manipulação	N	N	A-C	A-F
Dificuldade para representar tipo de dados semanticamente	A+	A-A+	A-A+	A-B
Modelos semânticos complexos	N	A-C	A-C	A-F

Capítulo 4

Proposta

"O pessimista queixa-se do vento, o otimista espera que ele mude e o realista ajusta as velas."

Willian George Ward

Este capítulo tem como objetivo apresentar a concepção e o desenvolvimento da abordagem proposta no presente trabalho. Inicialmente, é apresentada uma visão geral da abordagem proposta e o modelo de representação semântica dos serviços adotado . Seguido da visão lógica da arquitetura e mecanismo de tolerância a falhas. Por fim, a visão de processos.

4.1 Visão Geral

A solução proposta neste trabalho consiste em um *framework* para a descoberta, composição e invocação automática de Serviços *Web* Semânticos com enfoque em Mineração de Dados. O *framework* proposto encapsula algoritmos de pré-processamento, mineração e pós-processamento em serviços Web semânticos de modo à oferecer uma estrutura flexível, permitindo que desenvolvedores estendam e adicionem novos algoritmos a fim de realizar todo o processo de descoberta de conhecimento de modo transparente para a aplicação.

Inicialmente, o *framework* proposto implementa algoritmos provenientes da ferramenta *Weka*. O Weka é uma coleção de algoritmos de aprendizagem de máquina para tarefas de mineração de dados. Os algoritmos podem ser aplicados diretamente a um conjunto de dados ou chamados de seu próprio código Java. Weka contém ferramentas para pré-processamento de dados, classificação, regressão, agrupamento, regras de associação e

visualização. Também é adequado para o desenvolvimento de novos esquemas de aprendizado de máquinas [43]. No entanto, como destacado anteriormente, a aplicação desenvolvida, não se limita apenas aos algoritmos de mineração da ferramenta *Weka*, mas permite aos desenvolvedores adicionarem novos serviços que encapsulam diferentes algoritmos e técnicas de mineração de diferentes ferramentas de mineração. Desse modo, os desenvolvedores podem instanciar a solução proposta para o desenvolver de novas aplicações no domínio da mineração de dados.

4.2 Visão Lógica da Arquitetura

Nesta Seção é apresentada a visão lógica da arquitetura da abordagem proposta. A arquitetura adota o estilo arquitetural orientado a serviços com o objetivo de promover a extensão e a integração entre diferentes aplicações. Nesse sentido, o projeto arquitetural apresentado na Figura 4.1 tem como objetivo apresentar em detalhes os componentes que integram a estrutura da aplicação.

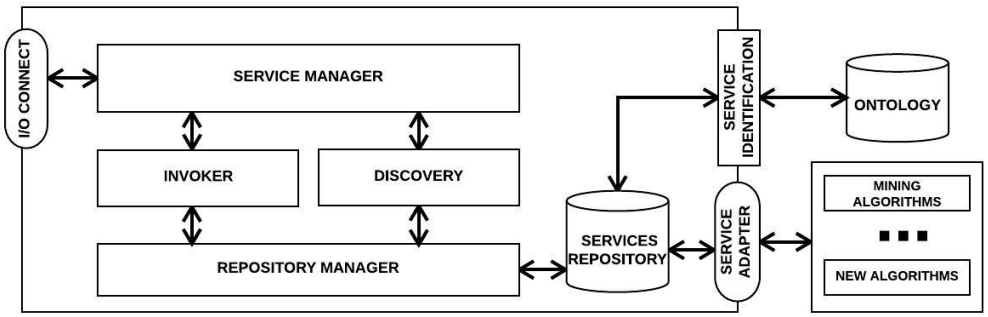

Figura 4.1: Visão Lógica da Arquitetura do Framework

Nas seções a seguir, será apresentado em maiores detalhes a função desempenhada por cada componente definidos no projeto arquitetural 4.1.

I/O Connect

O componente **I/O Connect** contém os padrões de comunicação aceitos na solução proposta e possibilita a integração de outras aplicações *Web*. Além dos padrões de comunicação, este componente permite que outras aplicações clientes façam uso de serviços e/ou composição

de serviços de mineração. Outra função deste é proporcionar a entrada de diferentes tipos de dados que serão usados na mineração. Para assegurar que os dados submetidos por outras aplicações obedeçam aos formatos aceitos pelos algoritmos de mineração, este componente tem acesso a serviços que recebem os dados enviados da aplicação e os transformam em formatos aceitos pelos algoritmos.

Service Adapter

Este componente ***Service Adapter*** representa a abstração do encapsulamento de algoritmos de mineração através dos serviços *Web*. Desse modo, é possível integrar técnicas e algoritmos de mineração disponibilizados por diferentes ferramentas, onde cada serviço encapsula um algoritmo específico de mineração.

Service Manager

O gerenciador de serviços é considerado o núcleo da solução proposta. Ele é responsável por controlar a sequência de processos que devem ser realizados para atender as requisições solicitadas por outras aplicações. Este componente recebe requisições enviadas do ***I/O Connect*** e através das mesmas, invoca os algoritmos para realizar todo o processo de descoberta, composição e inovação de serviços *Web* semânticos encontrados no ***Repository Services***.

Discovery

Este componente de descoberta tem a função de encontrar serviços disponíveis no ***Repository Services*** para atender a requisições enviadas por outras aplicações. O mesmo, recebe a descrição do serviços e começar a buscar serviços e/ou composições que tenha maior similaridade com as informações definidas na requisição. Então quando encontrados serviços e/ou composições que melhor atendam as restrições definidas na requisição, são encaminhados para o ***Service Manager***.

Invoker

Este componente de invocação é responsável pela execução dos serviços encontrados durante a execução do componente ***Discovery***. O mesmo, é capaz de invocar serviços simples e/ou

composições de maneira transparente a aplicações que estejam fazendo uso da abordagem. Então quando o processo de invocação de serviços é finalizado, o resultado obtido é enviado para o componente ***Service Manager***.

Repository Manager

O componente gerenciador de repositório fornece o controle de acesso aos serviços disponíveis no ***Repository Services***, incluindo operações que permitem adicionar, buscar, listar, excluir e atualizar serviços.

Repository Services

Este componente armazena todas as implementações de serviços que encapsulam técnicas, algoritmos de mineração, pré-processados e pós-processamento de dados fornecidos na abordagem proposta.

Service Identification

O componente identificador de serviços realiza o mapeamento dos serviços disponíveis no ***Repository Services*** com a sua respectiva descrição semântica armazenada no repositório de ontologias.

Ontology

Este componente é responsável no armazenamento das ontologias de domínio usadas na abordagem proposta. Nele, temos a Ontologia de Serviços e a Ontologia Mineração.

4.3 Visão de Processos

A Figura 4.2 tem por objetivo exibir o diagrama de atividades, que ilustra os processos envolvidos na execução de requisições e respostas enviadas pela aplicação. Nesta figura é apresentado a execução de todo o processo envolvido, desde a requisição enviada pelo usuário, a procura por serviços, a verificação de composições existentes para a requisição solicitada, a invocação de serviços, a composição de serviços, a verificação de serviços com

problemas, além da preparação de respostas que serão enviadas para requisição solicitada. A seguir, são apresentadas as descrições em maiores detalhes.

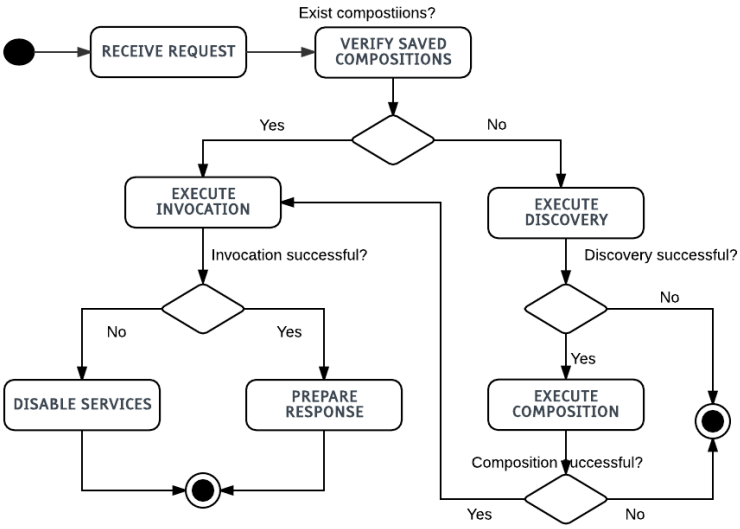

Figura 4.2: Diagrama de Atividades da Solução Proposta

***Receive request*:** nesta etapa, a aplicação recebe a descrição dos serviços que a aplicação deseja executar. Logo em seguida, as descrições serão utilizadas nos processos de descoberta, composição e invocação dos serviços.

***Verify saved compositions*:** nesta etapa, é executada uma busca no repositório de composições salvas, para verificar se existem composições que atendam a requisição informada pelo usuário. Caso existam, e estas sejam compatíveis com a requisição desejada, é ativado o mecanismo de invocação dos serviços, caso contrário, é acionado o mecanismo de descoberta de serviços.

***Execute discovery*:** nesta etapa, é iniciada a busca por serviços ou conjunto de serviços que melhor se adéquam as especificações informadas na requisição solicitada. Caso existam serviços compatíveis, o mecanismo de composição é ativado, ao contrário, se mecanismo de descoberta falhar na tentativa de encontrar serviços que ofereçam as funcionalidades requisitadas, é interrompido.

***Execute composition*:** nesta etapa, é realizada a composição dos serviços encontrados na fase anterior. O processo de composição baseia-se no cálculo do cosseno de similaridade entre os serviços. Assim sendo construída alguma composição, é ativado o mecanismo de invocação, ao contrário, se o mecanismo de composição falhar na tentativa de construir

composições, é interrompido.

***Execute invocation*:** nesta etapa, é executado o processo de invocação dos serviços encontrados na composição realizada anteriormente. Conforme o resultado obtido durante a execução do processo de invocação, apenas um mecanismo é ativado, podendo ser o mecanismo de resposta ou o mecanismo que realiza a desativação do serviço.

***Prepare response*:** nesta etapa, é criada a resposta que será enviada à aplicação.

***Disable services*:** nesta etapa, dado que ocorreu um problema durante a invocação e execução de um serviço, o mecanismo de tolerância a falhas é ativado.

4.4 Mecanismo de Tolerância a Falhas

O *framewokr* proposta neste trabalho provê um mecanismo de tolerância a falhas que está integrada aos mecanismos de invocação de serviços. Nesta fase de invocação dos serviços, os dados passam por um processo de validação dos parâmetros de entrada e saída dos serviços a serem invocados. Este processo de validação, consiste na extração de informações dos dados informados pelo usuário para assegurar que os parâmetros de entrada e saída dos serviços sejam consistentes com os requisitados pelos serviços disponíveis no *Repository Services*.

Após a validação, a aplicação realiza a execução do processo de invocação do (s) serviço (s) com os paramentos consistentes. Assim, dado que ocorreu falhas durante o processo de invocação, neste caso, é disparado o mecanismo de tolerância a falhas, assegurando que serviços com falhas, erros e defeitos sejam desativados automaticamente do *Repository Services*. Evitando assim, que esses venham proporcionar danos futuros a outras aplicações que façam uso da solução proposta.

4.5 Serviços Web

Nesta seção são apresentados detalhes dos serviços *Web* implementados pelo *framework* proposto. Esses serviços encapsulam algoritmos de mineração disponíveis na ferramenta Weka. Subdivididos em três categorias - serviços de pré-processamento, serviços de mineração e pós-processamento - o serviços implementados cobrem cerca de 61% da coleção de algoritmos de mineração disponibilizados pela ferramenta Weka, incluindo algoritmos de árvore de

decisão, agrupamento, regras, entre outros, conforme apresentado na Figura 4.3.

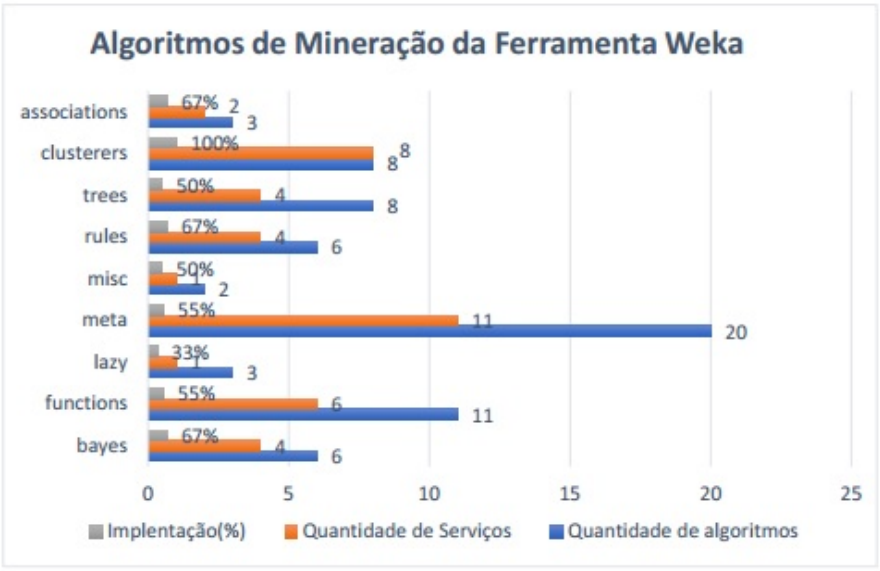

Figura 4.3: Porcentagem de serviços implementados cada grupo de algoritmos do Weka.

4.5.1 Representação Interna de Serviços

O modelo definido para representar os serviços *Web* semânticos é descrito através do diagrama de classes apresentado na Figura 4.4. Composto por três classes, a representação interna dos serviços possui uma classe abstrata *Service* e as subclasses classes *SimpleService* e *CompositeService*.

A classe *AbstractService* é a base para representação interna dos serviços. Essa classe contém o método abstrato *invoke* que tem a função de executar o processo de invocação do serviço seja ele um serviço simples ou um conjunto de serviços compostos. As classes *SimpleService* e a *CompositeService* estendem a classe *AbstractService*. A classe *SimpleService* representa um serviço simples enquanto a classe *CompositeService* corresponde a um conjunto de serviços compostos. Nesse contexto, a fim de possibilitar a representação de estruturas de serviços compostos, foi adotado o padrão de projeto *Composite* [11].

Os serviços disponíveis na aplicação foram implementados utilizando a linguagem Java, assim como os demais módulos da arquitetura apresentados na Figura 4.1. Os serviços seguem o padrão REST (*Representational State Transfer*) e utilizam JSON (*JavaScript Object Notation*) para se comunicar. Um dos fundamentos do REST é o uso dos métodos definidos

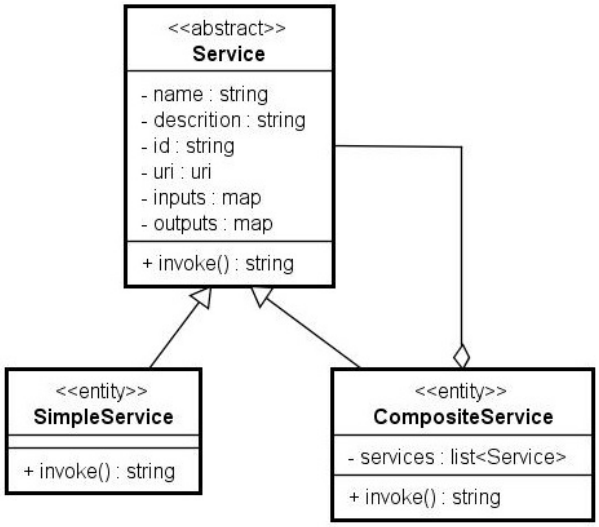

Figura 4.4: Diagrama de classes que representam os serviços da Solução

pelo protocolo HTTP nas funcionalidades de suas aplicações. Esses métodos são: *POST*, *GET*, *PUT*, *PATCH* e *DELETE*. Dentre eles, foram implementados na abordagem proposta as funções que permitem aos usuários e desenvolvedores adicionar, deletar, buscar, atualizar e listar serviços de pré-processamento, mineração e pós-processamento de dados.

4.5.2 Descrição Semântica de Serviços

A etapa seguinte no processo de construção do *framework* proposto é a definição das descrições semânticas de cada serviço implementado a fim de transformá-los em serviços semânticos, permitindo a realização do processo de descoberta, composição e invocação automática desses serviços. Como supracitado, o modelo semântico adotado foi o modelo proposto por Barros [7]. Esse modelo possui como principal vantagem a simplificação do processo de construção e manipulação de serviços semânticos. Apresentada no formato Manchester da OWL, a Listagem 4.1 ilustra a representação semântica do serviço *Web* que encapsula o algoritmo de classificação J48 seguindo esse modelo semântico.

Código Fonte 4.1: Representação semântica do serviço J48 no formato Manchester da OWL

```
1 Individual: <domain.owl#J48_Service>
2   Types:
3     <domain.owl#Service>
4   Facts:
```

```
5    <domain.owl#hasInput><domain.owl#ArffInput>,
6    <domain.owl#hasOutput><domain.owl#J48Model>,
7    <domain.owl#URI> "http://172.20.9.39/framework/j48",
8    <domain.owl#inputSchema> "INPUT_JSON_SCHEMA",
9    <domain.owl#label> "J48 Service",
10   <domian.owl#descrition> "Este servicos retorna
11   o modelo de decisa o em formato de a vore dos
12   dados fornecidos como par metros. ",
13   <domain.owl#outputSchema> "OUTPUT_JSON_SCHEMA"
```

Analisando o código 4.1, na linha 1, é definido o indivíduo *J48Service*, a linha 3, informa que o indivíduo é um serviço. Finamente, nas linhas de 5 a 10, temos a definição das propriedades do serviço. A linha 5, define o *ArffInput* que representa o parâmetro de entrada. Na linha 6, temos definido como saída do serviço o *J48MODEL* que representa o modelo de classificação. A URI para acessar o serviço na linha 7. As linhas 8 e 13, define os *inputSchema* e o *outputShema* que são utilizados na invocação do serviço. O *onputSchema* aponta para o *INPUTJSONSCHEMA* na ontologia de domínio. O *INPUTJSONSCHEMA* possui as propriedades de como os dados devem ser enviados para o serviço. Por sua vez, o *outputSchema* aponta para o *OUTPUTJSONSCHEMA* definido na ontologia de domínio, o mesmo, define as propriedades das saídas do serviço que podem ser usadas como parâmetros de entradas para outros serviços. E, por fim, nas linhas 9 e 10, o *label* representa o nome do serviços e o *descritions* possui a descrição do serviço. Também foi criando um tutorial para auxiliar os usuários na criação de Seviços Web Semânticos A.1.

4.5.3 Ontologia de Domínio

A ontologia de domínio adotada como base para definição das descrições semânticas dos serviços *Web* semânticos implementados foi proposta neste trabalho e descreve tarefas e técnicas de mineração de dados, especificando os parâmetros de entrada e saídas dos algoritmos de mineração da ferramenta Weka. Na Figura 4.5 é apresentada graficamente a ontologia de domínio proposta, destacando a distribuição das classes e suas subclasses.

- ***MiningAlgorithms*:** é a classe mais abstrata que representa as técnicas e algoritmos de mineração de dados. Na ontologia, as técnicas e algoritmos de mineração estão

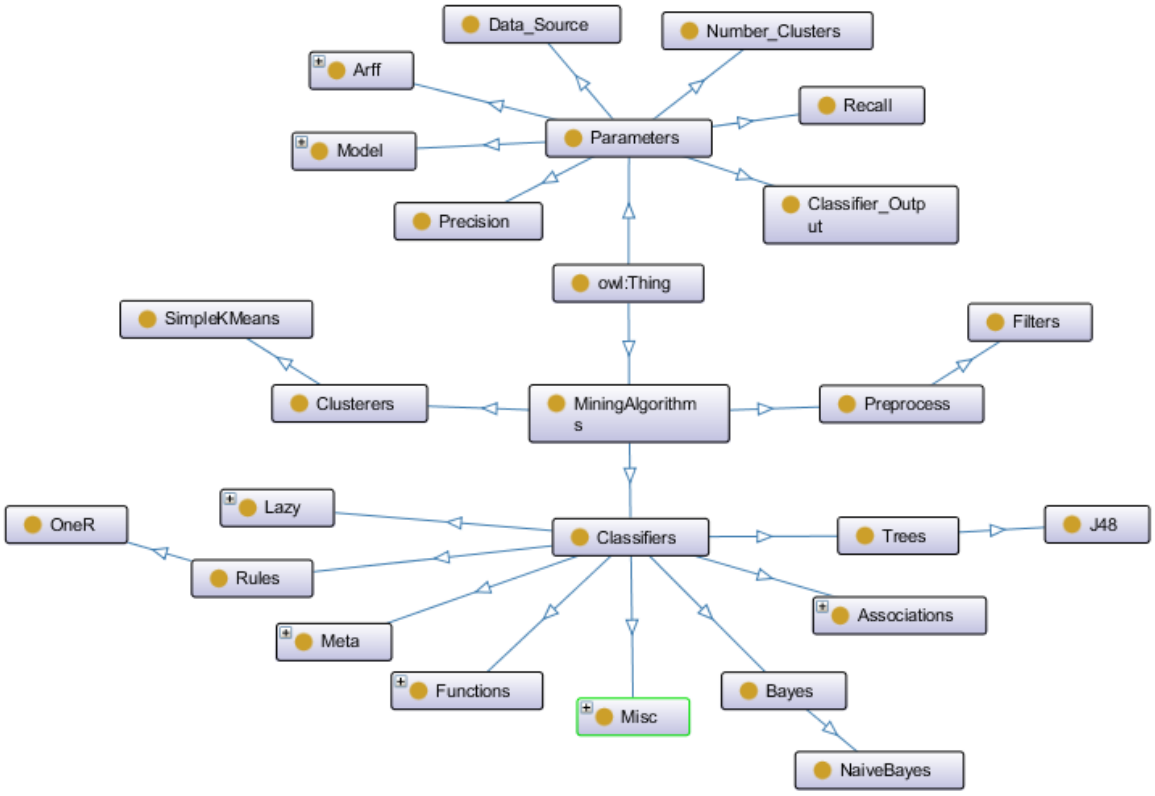

Figura 4.5: Ontologia de mineração implementada na aplicação.

categorizadas em: pré-processamento, classificação, agrupamento e associação.

- ***Preprocess*:** Os algoritmos e técnicas que realizam pré-processamento devem ser subclasses desta classe. Por exemplo: algoritmo *Filters* da ferramenta Weka.
- ***Classifiers*:** é a subclasse ”mãe”de todas as técnicas e algoritmos de classificação. Na ontologia, os algoritmos e técnicas de classificação estão divididos por tipos, como: árvores, funções, regras, associação, bayesiano, entre outros.
  - ***Trees*:** os algoritmos e técnicas que realizam classificação baseada em árvores devem ser subclasses desta classe. Por exemplo: algoritmo J48 da ferramenta Weka.
  - ***Functions*:** os algoritmos e técnicas que lidam com a classificação baseada em funções devem ser subclasses desta classe. Por exemplo: algoritmo *SimpleLogistic* da ferramenta Weka.
  - ***Bayes*:** os algoritmos e técnicas que utilizam classificação baseada em pro-

babilidade Bayesiana devem ser subclasses desta classe. Por exemplo: algoritmo *NaiveBayes* da ferramenta Weka.

∗ ***Rules*:** os algoritmos e técnicas que realizam classificação baseada em regras devem ser subclasses desta classe. Por exemplo: algoritmo *OneR* da ferramenta Weka.

– ***Clusters*:** os algoritmos e técnicas relacionados à clusterização devem ser subclasses desta classe. Por exemplo: algoritmo *SimplesKMeans* da ferramenta Weka.

– ***Associetes*:** os algoritmos e técnicas que realizam associação devem ser subclasses desta classe. Por exemplo: algoritmo *Apriori* da ferramenta Weka.

- ***Parameters*:** representa os parâmetros de entrada e saída dos algoritmos e técnicas de mineração de dados definidos na ontologia.

Se necessário, a ontologia de domínio proposta pode ser estendida a fim de descrever novos algoritmos de mineração e/ou parâmetros de entrada e saída desses algoritmos. Assim, o desenvolvedor pode adaptar os serviços de mineração de acordo com as necessidades exigidas em sua aplicação.

4.6 Service Manager

O gerenciador de serviços é o componente principal do *framework* proposto. Esse componente é responsável por todo o processo descoberta, composição e invocação de serviços *Web* semânticos de forma automática e dinâmica a partir dos parâmetros de entrada e saída especificados pelas requisições recebidas das aplicações clientes. As requisições são recebidas pelo gerenciador através do par *[chave, valor]*, onde a *chave* representa a URI da ontologia do parâmetro de entrada e/ou saída e o *valor* representa o valor do parâmetro em si. Desse modo, são recebidos dois *JSONs*, um com os parâmetros de entrada e outro com os parâmetros de saída.

O diagrama de classes que representa o mecanismo de gerenciamento de serviços é apresentado na Figura 4.6 e detalhado a seguir. Esse mecanismo é composto por 03 classes

principais - *ServiceManager*, *AbstractMatchermaker*, *InvocationEngine* - e suas subclasses, *BCMatchMaker* e *ModelInvocation*.

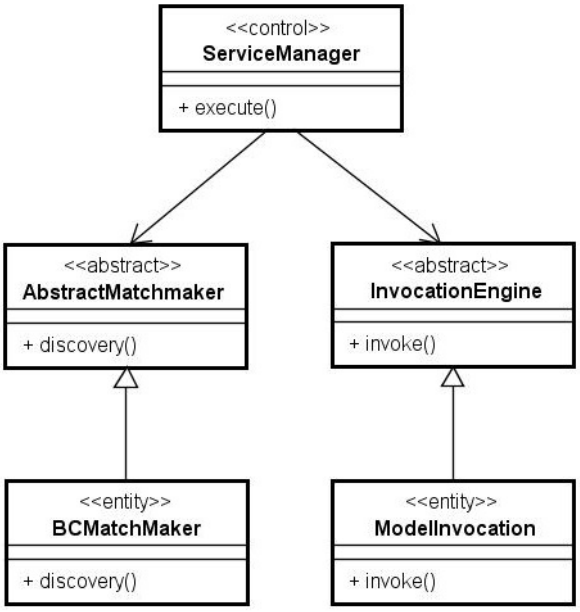

Figura 4.6: Diagrama de classes que representam o Service Manager da Solução.

A classe *ServiceManager* referência as classes abstratas (*AbstractMatchermaker* e *InvocationEngine*) e utiliza suas subclasses para realizar os processo de descoberta e composição e invocação dos serviços *Web* semânticos, respectivamente. No modelo representado na Figura 4.6, são instanciadas a classe *BCMatchMaker* a fim de realizar o processo de descoberta e composição dos serviços e a classe *ModelInvocation* para realizar o processo de invocação dos serviços.

Em resumo, o *ServiceManager* controla o processo de requisições e respostas da aplicação. As requisições são a forma como os usuários especificam as funcionalidades (descobrir, compor e invocar) que desejam durante o uso da solução proposta. Estas funcionalidades contém os parâmetros que serão utilizados durante a sua execução. Estas requisições podem ser do tipo: requisições de descoberta de serviços, requisições de descoberta e invocação de serviços e requisições de invocação de serviços. A seguir, são apresentadas em maiores detalhes os tipos de requisição citados:

- **Requisições de descoberta de serviços:** Na requisição de descoberta, o usuário informa como entrada as descrições dos parâmetros de entrada e saída do serviço que

deseja invocar. Por meio desta requisição, a aplicação busca o serviço que a melhor atende, fazendo composições de serviços se for necessário, e retornando como resposta para o usuário o serviço encontrado ou a composição de serviços para que este possa analisar, se desejar, futuramente fazer a invocação.

- **Requisições de descoberta e invocação de serviços:** o usuário informa as descrições dos parâmetros de entrada e saída do serviço que se deseja invocar. Por meio desta requisição, a aplicação realiza a descoberta de serviços que a melhor atendam, fazendo composições de serviços se for necessário, e realizando a invocação dos serviços de forma automática e dinâmica, retornando assim para o usuário o resultado da invocação.

- **Requisições de invocação de serviços:** o usuário informa como parâmetros de entrada a URI e os dados definidos como entrada do serviço que se deseja invocar. Por meio da URI e os parâmetro definido como entrada do serviço, a solução proposta, realiza de forma automática e dinâmica a invocação do serviço, e retorna para o usuário como resposta o resultado da invocação.

4.6.1 Mecanismo de Descoberta e Composição de Serviços

Nesta Seção são detalhados os aspectos de implementação do mecanismo responsável pela realização do processo de descoberta e composição dos serviços semânticos disponíveis. Ou seja, este mecanismo tem a função de encontrar e selecionar os serviços que melhor atendam às necessidades do usuário especificadas nas requisições recebidas pelo *ServiceManager*. Na Figura 4.7 é apresentado o diagrama de classes que representa este mecanismo. As classes que compõem o mecanismo de descoberta e composição de serviços são: as classes abstratas *AbstractMatchMaker* e as subclasse classe *BCMatchMaker*.

A classe *AbstractMatchMaker* define o método abstrato *discovery()*, que , através dos parâmetros de entrada e saída recebidos, realiza o processo de descoberta e composição de serviços da aplicação. Assim, a classe *AbstractMatchMaker* é estendida apenas pela a classe *BCMatchMaker*, que, por sua vez, implementa o método responsável por controlar o processo de descoberta e composição. A fim de realizar esse processo, a classe *BCMatchMaker* implementa o algoritmo com uma estratégia de encadeamento para trás. Este algoritmo faz

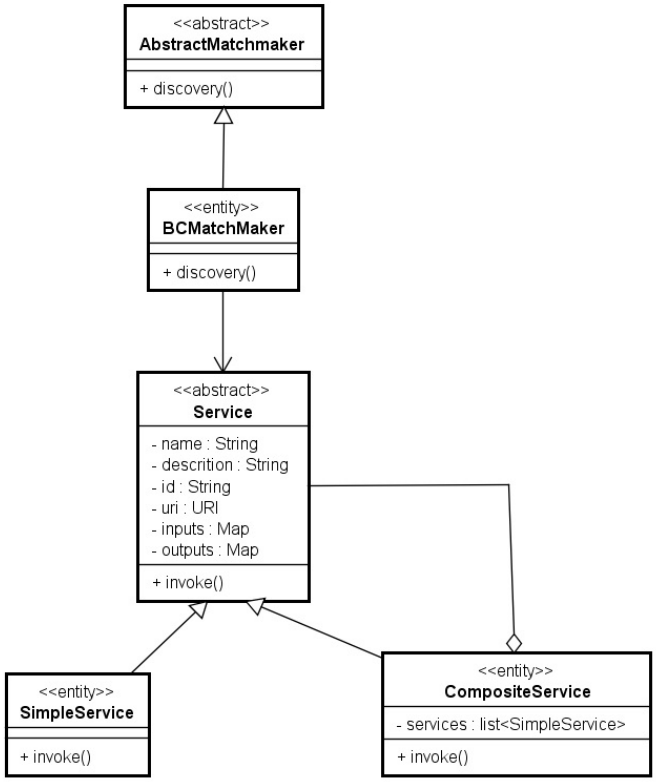

Figura 4.7: Diagrama de classes que representam o Mecanismo de Descoberta e Composição de Serviços da Solução

a análise de similaridade através da métrica de cosseno de similaridade a fim de avaliar a correspondência semântica, *"Semantic Matching"*, que existe entre entidades.

O cálculo da métrica de cosseno de similaridade é uma medida de similaridade entre dois vetores não nulos de um espaço de produto interno que mede o cosseno do ângulo entre eles, isto é, quanto maior for o ângulo, maior é a similaridade. Portanto, a medida do cosseno de similaridade entre dois vetores diferentes de zero, de atributos A e B, pode ser obtida através da fórmula:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

O algoritmo *BCMatchMaker* utiliza o cálculo da métrica do cosseno de similaridade para localizar serviços utilizando os métodos *directMatch* e *inderectMatch*. Assim, dado que o usuário envie uma requisição com a descrição semântica do serviço que se deseja e os parâmetros de entrada e saída, o algoritmo inicializa a busca por serviços utilizando o método *directMatch*. Caso nenhum serviço disponível atenda a requisição recebida, o

método *indirectMatch* é acionado, buscando compor um conjunto de serviços que atenda as demandas recebidas.

O método *directMatch* tem por objetivo encontrar relações diretas (”Direct Matching”) entre os parâmetros de entrada e saída informados na requisição do usuário, retornando para o usuário, o serviço que melhor atenda a requisição desejada.

O método *indirectMatch* tem a função de encontrar relações indiretas (”Indirect Matching”) entre os parâmetros de entrada e saída especificados na requisição do usuário, com os serviços de mineração disponíveis na aplicação. Desse modo, este método é responsável em construir novas composições de serviços de mineração que mais se aproximem da requisição recebida. Para construir uma nova composição de serviços, o *indirectMatch* recebe os parâmetros de entrada e saída definidos na requisição do usuário e, a partir desses parâmetros, o método realiza a busca entre os serviços no repositório de serviços, realizando a comparação através do cálculo de cosseno de similaridade com os conceitos definidos nas ontologias. Dessa maneira, o algoritmo inicia o processo de composição a partir da saída especificada pelo usuário, utilizando o encadeamento para trás até encontrar um serviço que possua uma entrada compatível com a entrada especificada pelo usuário. São considerados compatíveis entradas e saídas com similaridade acima de 80%. Esse valor foi definido após testes preliminares onde percebeu-se que a similaridade entre nós pais e filhos era de 86%. Como nós pais e filhos são compatíveis, é possível, por exemplo, substituir um serviço que tem como parâmetro de saída um nó pai por um outro serviço que possui um nó filho como saída sem prejuízos, pois são serviços semelhantes e compatíveis entre si.

Dicionário Semântico

Inicialmente proposto por Barros et al. [13], o dicionário semântico tem a função de armazenar a similaridade entre os conceitos presentes na ontologia calculados de modo *offline*. Desse modo, os algoritmos de descoberta e composição não necessitam realizar os cálculos de similaridade em tempo de execução, tornando esses processos mais rápidos e eficientes. A adoção do dicionário semânticos mais a proposta de adotar um SGBD baseado em arquivos para armazenar as similaridades calculadas, agrega um diferencial importante ao *framework* proposto em relação aos demais propostas relacionadas.

Este dicionário é utilizado junto ao mecanismo de descoberta e composição de serviços

(maiores detalhes, Seção 4.6.1). Por exemplo, dado que o usuário enviou uma requisição para a aplicação e um serviços simples não atende a requisição deste usuário, é necessário construir uma composição de serviços que atenda a requisição recebida com a maior similaridade possível. Nesse contexto, o algoritmo *BCMatchMaker* (maiores detalhes, na seção 4.6.1) recebe as informações definidas na requisição e com base nos parâmetros especificados como entrada e saída do serviço, o mesmo realiza consultas ao dicionário para verificar se existem *Match* que podem ser *directMatch* e *indirectMatch*. DEsse modo, quando são encontrados *Matchs* entre os parâmetros, o algoritmo seleciona os serviços que apresentam maior similaridade e os utiliza no processo de composição.

Na Listagem 4.2 é apresentado resultado de uma consulta ao dicionário semântico da aplicação. Analisando o código, nas linhas 1 e 2 estão dois termos e na linha 3 o resultado da similaridade calculada a partir do cosseno de similaridade entre os termos.

Código Fonte 4.2: Dicionário de conceitos semânticos.

```
1  term_1:http://172.20.9.1/framework/domian_ontology.owl#Trees
2  term_2:http://172.20.9.1/framework/domian_ontology.owl#J48
3  CossineSimilarity: 0.8660253882408142
4
5  term_1:http://172.20.9.1/Framework/ontology/domian_ontology.owl#
       TrainBayes
6  term_2:http://172.20.9.1/Framework/ontology/domian_ontology.owl#Model
7  CossineSimilarity: 0.8164966106414795
8
9  term_1:http://172.20.9.39/framework/domian_ontology.owl#J48
10 term_2:http://172.20.9.39/framework/domian_ontology.owl#J48
11 CossineSimilarity: 1
12
13 ...
```

4.6.2 Mecanismo de Invocação de Serviços

É representado na Figura 4.8 o diagrama de classes do mecanismo de invocação de serviços da solução proposta. Este mecanismo é responsável por executar o processo de invocação de serviços e/ou composições para atender a requisições das demais camadas. Como destacado

na Figura, este mecanismo de invocação é composto pelas seguintes classes: a classe abstrata *InvocationEngine* e a subclasse *ModelInvocation*. A classe *InvocationEngine* é a base para realizar invocação de serviços. Nesta classe, está definido o método *execute*, responsável por controlar todo processo de invocação. A classe abstrata *InvocationEngine* é estendida pela a classe *ModelInvocation*, desta forma, a partir da classe *InvocationEngine* o desenvolvedor pode implementar outros mecanismos de invocação de acordo com as restrições de suas aplicações.

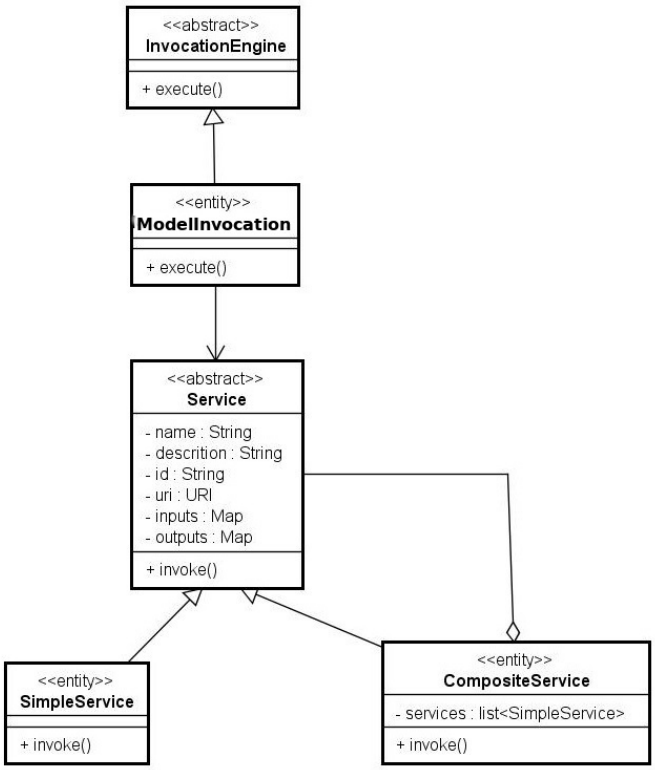

Figura 4.8: Diagrama de classes que representam o Mecanismo de Invocação de Serviços da Solução.

A subclasse *ModelInvocation* gerencia o processo de invocação de serviços que são executados para melhor atender as requisições enviadas pelo componente *ServiceManager* (maiores detalhes na Seção 4.6). Essas requisições podem ser direcionadas para a invocação de um serviço simples ou um conjunto de serviços compostos.

Como destacado anteriormente, tanto o processo de invocação quanto o processo de composição de serviços é realizado a partir dos *JSONs Schemas* de entrada e saída de cada serviço. Os *JSONs Schemas* definem os parâmetros de entrada esperados e o padrão de res-

posta para os parâmetros de saída. As Listagens 4.3 e 4.4 representam, respectivamente, o *JSON Schema* de saída e uma instancia do *JSON Schema* de saída com os dados de resposta de um serviço.

Código Fonte 4.3: JSON Schema do parâmetro de saída de serviço.

```
1  {
2      "@Ontology" : "http://domian_ontology.owl",
3      "type" : "object",
4      "properties" : {
5          "@Ontology#Arff" : {
6              "type" : "object",
7              "properties" : {
8                  "@Ontology#path" : {
9                      "type" : "string"
10                 }
11             },
12             "required" : [
13                 "@Ontology#path"
14             ]
15         }
16     },
17     "required" : [
18         "@Ontology#Arff"
19     ]
20 }
```

A Listagem 4.3 mostra um exemplo de *JSON Schema* que define os parâmetros de saída de um serviço que retorna um arquivo no formato (.arff), formato adotado pela ferramenta Weka. Analisando o Código 4.3, na linha 2, a URI que é definida na ontologia de domínio construída. Na linha 3 é definido o retorno do serviço como um objeto. Na linha 5, é definido o tipo de retorno do objeto, um .arff. Em seguida, na linha 6, é estabelecido o serviço deve retornar apenas uma instância. Da linha 7 até a linha 11 são definidas as propriedades do *arff* que serão retornadas: *Path* e o seu tipo (*String*). Nas linhas 12 à 14, o parâmetro *required* indica que a propriedade (*Path*) é obrigatória no *arff* retornado. Por fim, entre linhas 17 à 19, a definição do parâmetro *required* sinaliza que o tipo *arff* é obrigatório para a resposta.

Na Listagem 4.4 é apresentada o resultado de um serviço no formato *JSON* seguindo

o padrão de saída para definido pelo *JSON Schema* de saída deste serviço, representado na Listagem 4.3.

Código Fonte 4.4: Resposta no formato JSON do serviço que retorna um arquivo no formato (.arff))

```
1 {
2   "@Ontology": "http://domian_ontology.owl",
3   "@Ontology#Arff": {
4     "@Ontology#path": "src/br/tips/framework/data/Notas.arff"
5   }
6 }
```

Como representado na Listagem 4.4, a linha 2 define a URL de acesso para a ontologia de domínio adotada neste contexto. Na linha 3, o serviço retorna o arquivo *arff* resultante. A linha 4 indica o *Path* do *arff* resultante.

Em linhas gerais, o processo de invocação de serviços acontece da seguinte maneira, a classe *ModelInvocation* utiliza o *JSON Schema* (Código 4.3) que descreve os parâmetros de entrada do serviço. A partir do *JSON Schema* de entrada, são recuperadas as informações especificadas na estrutura do *JSON Schema* para construir o *JSON* (Código 4.4) de invocação do serviço. Esse *JSON* contém os dados que serão utilizados para invocar o serviço pretendido. Após construir o *JSON* com os dados de invocação, a classe *ModelInvocation* chama o método de execução, passando como parâmetros a URI do serviço e o *JSON* com os dados necessários a invocação. Após a invocação do serviço, a saída é retornado no formato *JSON* seguindo o padrão definido no *JSON Schema* de saída. Em cenários de serviços compostos, o *JSON* resultante de um serviço será utilizado como *JSON* de entrada do próximo serviço a ser invocado. Vale ressaltar que a saída recebida após a invocação de um determinado serviço pertencente a uma composição pode não ser igual a entrada do serviço subsequente. Desse modo, o mecanismo de invocação cria um novo *JSON* de entrada baseado no *JSON* de saída recebido do último serviço e no *JSON Schema* de entrada do serviço subsequente. Esse processo, por sua vez, é repetido até que todo o conjunto de serviços compostos sejam executados.

Outra característica importante do *framework* proposto é a capacidade de lidar com a ausência de parâmetros durante o processo de invocação. Isso é possível porque o meca-

nismo de invocação sempre realiza o processo de validação dos dados de entrada antes de realizar a invocação de quaisquer serviços. Desse modo, caso o usuário deseje invocar o serviço *SimplesKMeans* para gerar um modelo de agrupamento, a requisição deve especificar como parâmetros de entrada: um arquivo (.arff) e a quantidade de grupos desejada (*Clusters*). No entanto, se o usuário não informar todos os dados necessários para a invocação do serviço, esses estarão incompletos. Nesse caso, quando os dados passarem pelo processo de validação, o usuário será notificado dos parâmetros ausentes e que estes são necessários para a realização do processo de invocação. Este processo de validação tem por objetivo garantir a consistência dos dados de invocação, assegurando, assim, que os dados informados como entrada do serviço não produzam saídas inconsistentes e/ou erros durante o processo de invocação. O mesmo acontece durante o processo de invocação de um conjunto de serviços compostos. Assim, quando o mecanismo detecta que a ausência de algum parâmetro, o processo de invocação é suspenso e o usuário informado da necessidade de definição do parâmetro ausente.

O mecanismo de invocação da solução proposta também possui um mecanismo de tolerância à falhas. Esse mecanismo é capaz de lidar com outros tipos de falhas, como a indisponibilidade de um serviço. Nesse caso, o mecanismo de tolerância a falhas é invocado e o serviço em questão, desativado. Durante sua execução, o mecanismo executa o processo de descoberta e composição novamente em busca de serviços compatíveis a partir dos parâmetros definidos na requisição.

4.7 Repositório de Serviços

O repositório possui a função de gerenciar o acesso aos serviços disponíveis na aplicação. Para tal, o *framework* proposto utiliza dois repositórios. O primeiro, *RepositoryServices*, gerencia todos os serviços. O segundo, repositório *Ontology*, gerencia as descrições semânticas dos serviços. O intuito na adoção desse modelo foi promover o desempenho e eficiência dos processos de descoberta e composição dos serviços semânticos.

Na construção dos repositórios o SGBD adotado foi o MongoDB. O MongoDB [1] é uma aplicação de banco de dados de código aberto, de alta performance, sem esquemas e orien-

[1] https://www.mongodb.com/what-is-mongodb

tado a documentos. Desse modo,com o modelo semântico adotado onde os parâmetros de entrada e saída são descritos através de arquivos no formato *JSON* é possível gerenciar os dados relacionados aos serviços e suas descrições semânticas de modo natural e sob uma estrutura de alta performance. As informações mantidas pelos repositórios são:

- Dados dos serviços - descrição do serviço, URI de acesso ao serviço, dados de entradas e saídas;
- Descrição semânticas - *JSONs Schemas* de entrada e saídas dos serviços;

Nessa perspectiva, o processo de descoberta e composição de serviços é realizado a partir de consultas ao repositório. A Figura 4.9 representa o diagrama de classes que ilustra o módulo de controle e acesso aos dados relacionados aos serviços armazenados no repositório. O módulo controle é composto pelas seguintes classes: *AbstractRepository* e *RepositoryManager*. A seguir, são descritos os papéis desempenhados por cada uma dessas classes no contexto da solução proposta.

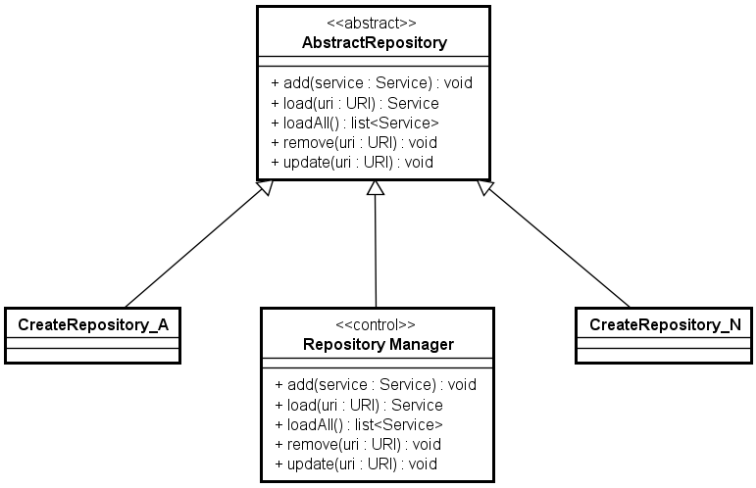

Figura 4.9: Diagrama de classes do Módulo de Repositório.

A classe *AbstractRepository* é a base para o processo de consultas ao repositório. Esta classe inclui os métodos abstratos que permitem adicionar e recuperar dados um serviço específico, além de possibilitar remover, atualizar e carregar dados de todos os serviços.

A classe *RepositoryManager* implementa os métodos abstratos definidos na classe *AbstractRepository*. Esta classe *RepositoryManager* tem a função de garantir a persistência e

manipulação dos dados dos serviços armazenados no repositório, assim como gerenciar as consultas realizados nos repositórios, atendendo as requisições das camadas superiores.

Capítulo 5

Avaliação da Abordagem Proposta

É descrito neste capítulo o processo adotado para avaliar o *framework* desenvolvido neste trabalho. Desse modo, foi realizada uma avaliação baseada em cenários com o objetivo de avaliar a *framework* proposto sob a perspectiva de dois atributos de qualidade, *Desempenho* e *Confiabilidade*.

5.1 Objetivo

O objetivo desse experimento foi definido com base no *Template GQM (Goal, Question, Metric)* [3]. Assim, *o experimento tem como objetivo* a abordagem proposta *com a intenção de* avaliar sua viabilidade *a respeito de* seu desempenho e confiabilidade *do ponto de vista de* desenvolvedores e usuários de algoritmos de mineração de dados *no contexto da* construção de aplicações de mineração de dados. Com base no objetivo, as seguintes questões de pesquisa foram definidas:

- **QP 1:** O *framework* proposto mantém o mesmo desempenho em diferentes cenários de composição?
- **QP 2:** O *framework* proposto reporta falhas/erros durante o processo de composição?

5.2 Design do Experimento

O experimento foi executado no contexto de uma instancia construída a partir do *framework* proposto e possui um caráter comparativo. Foram utilizados um conjunto de serviços de mineração construídos a partir da ferramenta Weka, contendo cerca de 42 serviços semânticos, o que corresponde a 61% dos algoritmos disponibilizados pelo Weka. Nesse contexto, as hipóteses definidas foram:

1. **Desempenho:**

   - Hipótese Nula, H_0: Não há diferença de tempo de resposta no processo de descoberta e composição de serviços configurando o algoritmo *BCMatchMaker* com o dicionário semântico.

   - Hipótese Alternativa, H_1: Há diferença de tempo de resposta no processo de descoberta e composição de serviços configurando o algoritmo *BCMatchMaker* com o dicionário semântico.

2. **Confiabilidade:**

   - Hipótese Nula, H_0: A aplicação não consegue se recuperar em cenários de execução malsucedidas.

   - Hipótese Alternativa, H_1: Há aplicação consegue se recuperar em cenários de execução malsucedidas.

Para avaliar o atributo de *Desempenho*, a métrica adotada foi *tempo de resposta*, em milissegundo (ms). O tempo de resposta compreende as seguintes etapas: i) recebimento da requisição; ii) realização do processo de descoberta e composição de serviços a partir dos parâmetros de entrada e saída definidas; iii) e a resposta ao usuário com o resultado da composição efetuada ou a mensagem informando que não há serviços ou composições correspondentes. Com relação ao atributo de *Confiabilidade*, foi mensurada a capacidade de tolerância a falhas do *framework* proposto a partir do *número de falhas identificadas* durante as execuções dos cenários a serem definidos.

5.2.1 Geração e Execução dos Cenários

Para a realização dos experimentos foram definidos 07 cenários baseado no ATAM (*Method for architecture evaluation*) [18]. Os cenários foram definidos a partir de um *Brainstorming* entre os principais *stakeholders* (desenvolvedores e usuários de algoritmos de mienração de dados) envolvidos. Os cenários definidos foram:

- C1 - Execução de composição mal sucedida
- C2 - Execução de composição simples com *matching* direto entre dois serviços
- C3 - Execução de composição simples com *matching* indireto entre dois serviços
- C4 - Execução de composição com 2 serviços e múltiplas escolhas de serviços com *matching* indireto e mais de uma opção de composição
- C5 - Execução de composição com 2 serviços e múltiplas escolhas de serviços com *matching* indireto e mais de uma opção de composição com a desativação de um serviço
- C6 - Execução em cenário com múltiplas escolhas de serviços com *matching* indireto entre 3 serviços
- C7 - Execução em cenário de recuperação com múltiplas escolhas de serviços com *matching* indireto entre 3 serviços com a desativação de um serviço

Para cada cenário foram realizadas 30 execuções. O número de execuções foi definido a partir do calculo de significância estatística para o tempo de resposta do sistema. A execução dos cenários aconteceu em um ambiente local composto por um *laptop* com processador Intel Core i7, com memória RAM de 8 GB e sistema operacional *Windows* 10.

5.3 Priorização dos Cenários e Execução dos Experimentos

Na sua execução, os cenários foram agrupados dois a dois a fim de compará-los frente aos atributos definidos, Desempenho e Confiabilidade. A priorização na execução dos experimentos foi definido com base na relação entre os cenários, suas características e a possi-

bilidade de permitir avaliar ambos os atributos simultaneamente. Desse modo, os cenários foram agrupados da seguinte maneira:

- Cenários C4 e C5
- Cenários C6 e C7

Para os demais cenários, só seria possível avaliar um dos atributos por vez. Assim, a execução e seus resultados são apresentados separadamente, na Seção 5.3.4. Como destacado, os resultados obtidos foram avaliados sob a perspectiva de um dos atributos de qualidades definidos, Desempenho ou Confiabilidade. Os cenários executados de maneira complementar foram:

- Cenário C1
- Cenários C2 e C3
- Cenários C3 e C4

A seguir são apresentados os resultados do primeiro grupo de experimentos e, posteriormente, os resultados dos experimentos complementares.

5.3.1 Cenários C4 e C5

Na tabela 5.4 são apresentados os valores dos resultados obtidos para métrica tempo de resposta do *framework*. O tempo de resposta médio para a realização do processo de descoberta e composição de serviços foi de *1042 ms* para C4 e *1050 ms* para C5. Os *boxplots* da Figura 5.1 representam a dispersão da métrica avaliada.

Tabela 5.1: Resultado da métrica - Tempo de Resposta C4 e C5.

Fonte: Elaborada pelo autor

	Mínimo	**1º Qt**	**Mediana**	**Média**	**Desvio Padrão**	**3º Qt**	**Máximo**
C4	990.0	1018.0	1030.0	1042.0	40.13	1046.0	1164.0
C5	989.0	1018.0	1029.0	1050.0	51.50	1056.0	1186.0

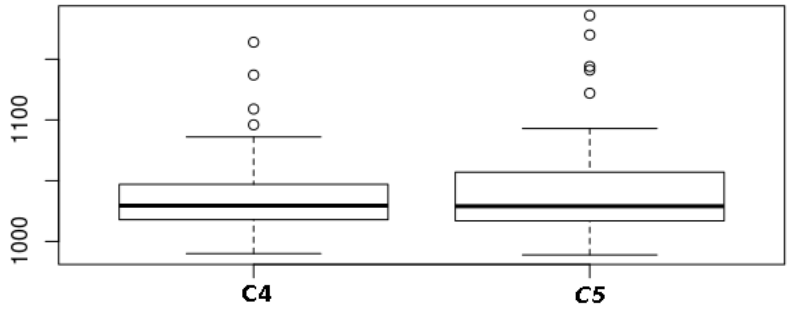

Figura 5.1: Boxplot - Tempo de Resposta dos cenários C4 e C5.

Fonte: Elaborada pelo autor

Na Figura 5.2 é apresentado os histogramas com as métricas *tempo de resposta* obtidas para cada um dos cenário. Como pode ser visto, os dados distribuídos nos histogramas não seguem uma distribuição normal. A fim de confirmar, foi aplicado o teste de *Shapiro-Wilk* [36] com resultados para *p-value* de 0.0002334 para C4 e *p-value* de 2.277e-05 para C5, isto é, ambos os resultados abaixo do limiar de 0.05.

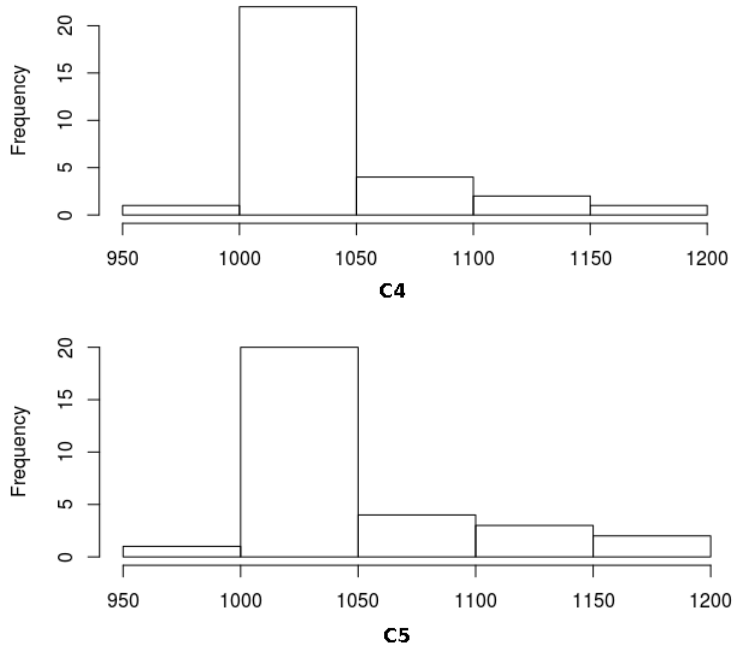

Figura 5.2: Distribuição não normal dos dados do tempo de resposta dos cenários C4 e C5.

Fonte: Elaborada pelo autor

O teste de *Wilcoxon* [42] foi aplicado para avaliar a métrica *tempo de resposta* comparando os resultados obtidos nos cenários. O resultado do teste apontou um *p-value* de 0.7901

superior a limiar de 0.05. Diante disso, não se pode refutar a hipótese nula, ou seja, o *framework* manteve o mesmo desempenho no processo de descoberta e composição de serviços mesmo em cenários que ocorreu a indisponibilidade de serviços, isto é, com base no resultado do *p-value* não há diferença estatisticamente significante para o atributo de Desempenho nesses cenários. Além disso, durante o cenário de recuperação, cenário com a desativação de um serviço que faria parte da composição, nenhuma falha foi reportada e a solução proposta encontrou um novo serviço compatível com o serviço desativado.

5.3.2 Cenário C6 e C7

Os valores para esse grupo foram sumarizados na Tabela 5.3 com resultados obtidos para métrica tempo de resposta para os processos de descoberta e composição de serviços. Para esses cenários o tempo de resposta médio foi de 1099 (ms) para C6 e 1102 (ms) para C7. Os *boxplots* da Figura 5.5 apresenta a dispersão da métrica avaliada.

Tabela 5.2: Resultado da métrica - Tempo de Resposta C6.

Fonte: Elaborada pelo autor

	Mínimo	**1º Qt**	**Mediana**	**Média**	**Desvio Padrão**	**3º Qt**	**Máximo**
C6	1033.0	1062.0	1074.0	1099.0	59.57	1117.0	1226.0
C7	1037.0	1053.0	1075.0	1102.0	65.28	1119.0	1263.0

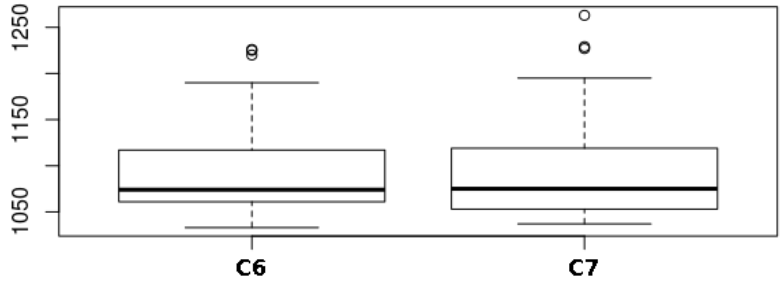

Figura 5.3: Boxplot - Tempo de Resposta do cenário C6 e C7.

Fonte: Elaborada pelo autor

Os histogramas apresentados na Figura 5.6 mostram a distribuição dos resultados. Para métrica *tempo de reposta*, foi aplicado o teste *Shapiro-Wilk* [36] que confirmou uma distribuição não-normal com *p-value* de 0.0001283 para C6 e 0.0001432 para C7.

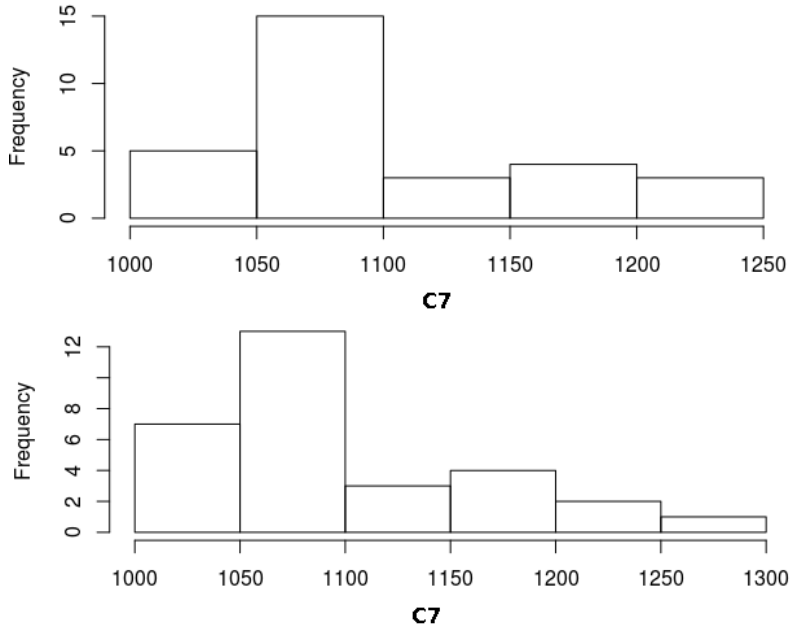

Figura 5.4: Distribuição não normal dos dados do tempo de resposta do cenário C6 e C7.

Fonte: Elaborada pelo autor

Nesse contexto, foi aplicado o teste de *Wilcoxon* [42] a fim de avaliar o atributo de Desempenho, utilizando a métrica de *tempo de resposta*. O resultado do teste apontou que não há diferença estatisticamente significativamente de Desempenho entre os cenários C6 e C7 com *p-value* de 0.836. Assim, os resultados mostram que o *framework* manteve o desempenho no processo de descoberta e composição de serviço mesmo diante de cenários de recuperação, onde falhas foram inseridas.

5.3.3 Cenário C6 e C7

Os valores para esse grupo foram sumarizados na Tabela 5.3 com resultados obtidos para métrica tempo de resposta para os processos de descoberta e composição de serviços. Para esses cenários o tempo de resposta médio foi de 1099 (ms) para C6 e 1102 (ms) para C7. Os *boxplots* da Figura 5.5 apresenta a dispersão da métrica avaliada.

Os histogramas apresentados na Figura 5.6 mostram a distribuição dos resultados. Para métrica *tempo de reposta*, foi aplicado o teste *Shapiro-Wilk* [36] que confirmou uma distribuição não-normal com *p-value* de 0.0001283 para C6 e 0.0001432 para C7.

Nesse contexto, foi aplicado o teste de *Wilcoxon* [42] a fim de avaliar o atributo de De-

Tabela 5.3: Resultado da métrica - Tempo de Resposta C6.

Fonte: Elaborada pelo autor

	Mínimo	**1º Qt**	**Mediana**	**Média**	**Desvio Padrão**	**3º Qt**	**Máximo**
C6	1033.0	1062.0	1074.0	1099.0	59.57	1117.0	1226.0
C7	1037.0	1053.0	1075.0	1102.0	65.28	1119.0	1263.0

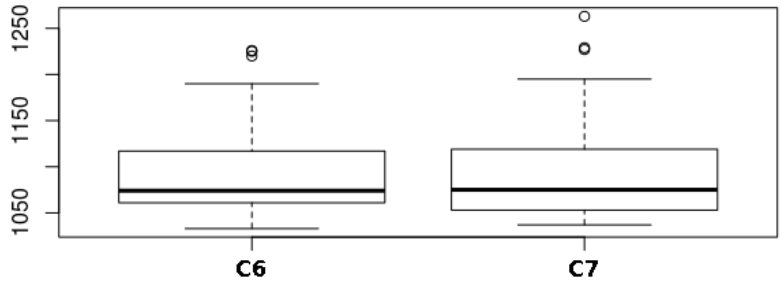

Figura 5.5: Boxplot - Tempo de Resposta do cenário C6 e C7.

Fonte: Elaborada pelo autor

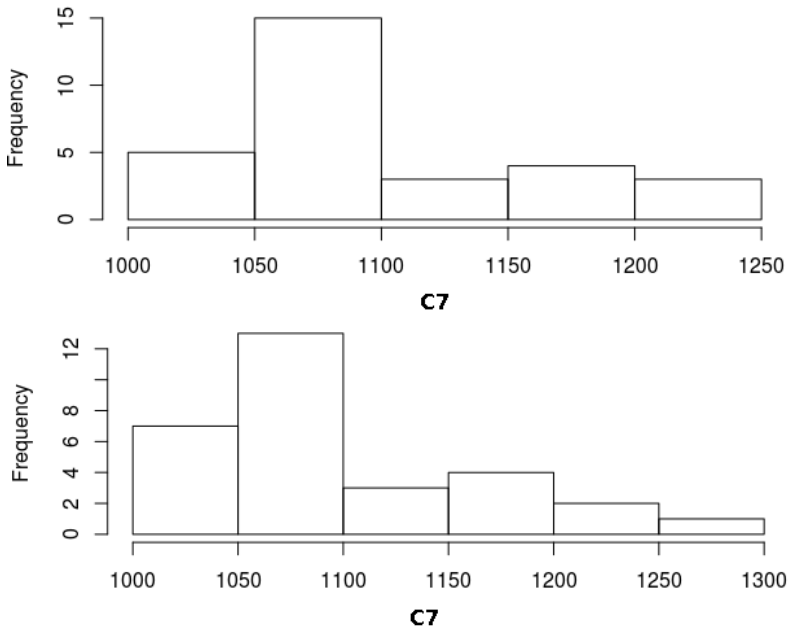

Figura 5.6: Distribuição não normal dos dados do tempo de resposta do cenário C6 e C7.

Fonte: Elaborada pelo autor

sempenho, utilizando a métrica de *tempo de resposta*. O resultado do teste apontou que não há diferença estatisticamente significativamente de Desempenho entre os cenários C6 e

C7 com *p-value* de 0.836. Assim, os resultados mostram que o *framework* manteve o desempenho no processo de descoberta e composição de serviço mesmo diante de cenários de recuperação, onde falhas foram inseridas.

5.3.4 Execuções Complementares

Cenário C1

A tabela 5.4 sumariza os valores de mínimo e máximo, a mediana , média e desvio padrão dos valores obtidos para a métrica *tempo de resposta*. O calculo da média foi feito a partir do somatório de todos os resultados de tempos que foram coletados durante a execução do *framework* e dividido por 30, valor correspondente ao número de repetições definido para cada cenário. O tempo de resposta médio foi de *151.9* milissegundos (ms). O *boxplot* apresentado na Figura 5.7 exibe a dispersão da métrica.

Tabela 5.4: Resultado da métrica - Tempo de Resposta C1

Fonte: Elaborada pelo autor

Mínimo	**1º Qt**	**Mediana**	**Média**	**Desvio Padrão**	**3º Qt**	**Máximo**
130.0	136.2	151.5	150.9	15.12	163.5	182.0

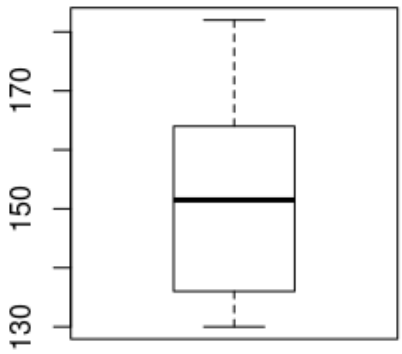

Figura 5.7: Tempo de resposta cenário C1.

Fonte: Elaborada pelo autor

O histograma representado na Figura 5.8 apresenta os resultados obtidos para métrica de tempo de reposta. Para verificar se os dados obedecem ou não uma distribuição normal foi

aplicado o teste de *Shapiro-Wilk* [36]. O resultado do teste confirma um *p-value* de 0.02568, ou seja, com de *p-value* inferior a 0.05 os dados fazem parte de uma distribuição não-normal.

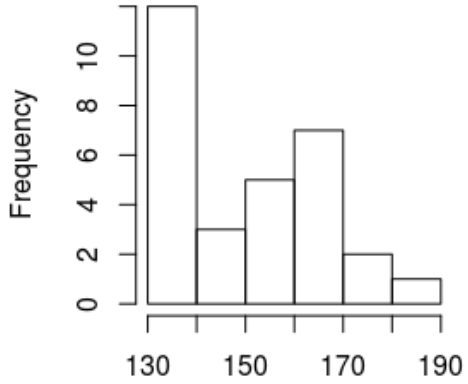

Figura 5.8: Distribuição não normal dos dados do tempo de resposta do C1

Fonte: Elaborada pelo autor

Este cenário simula a execução da aplicação em um ambiente malsucedido como o objetivo de avaliar o comportamento do sistema. Desse modo, foi possível verificar que em cenários onde nenhum serviço ou conjunto de serviços atende as necessidades do usuário o sistema respondeu bem e sem falhas.

Cenários C2 e C3

Os resultados obtidos para este grupo foram sumarizados na Tabela 5.5, onde estão definidos os valores mínimo e máximo, mediana, média e o desvio padrão para a métrica de desempenho *tempo de resposta*. O tempo de resposta médio para a realização do processo de descoberta e composição dos serviços foi de 982.1 ms para C2 e 1078 ms para C3. O *boxplot* apresentado na Figura 5.9 representa a dispersão dos dados.

Tabela 5.5: Resultado da métrica - Tempo de Resposta C2 e C3.

Fonte: Elaborada pelo autor

	Mínimo	**1º Qt**	**Mediana**	**Média**	**Desvio Padrão**	**3º Qt**	**Máximo**
C2	914.0	937.2	952.0	982.1	62.40	1033.5	1119.0
C3	993.0	1021.0	1068.0	1078.0	74.07	1111.0	1310.0

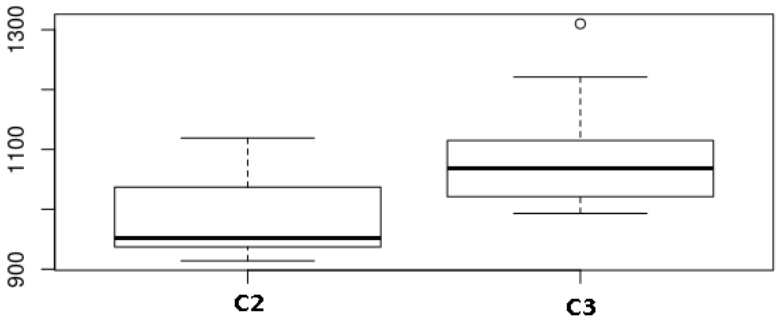

Figura 5.9: Tempo de resposta para os cenários C2 e C3

Fonte: Elaborada pelo autor

Os histogramas apresentados na Figura 5.10 apresentam as distribuições da métrica *tempo de reposta* para os cenários C2 e C3. O resultado do teste de normalidade, *Shapiro-Wilk* [36], confirmou distribuições não-normal para ambos os cenários, com *p-value* de 0.0001023 para C2 e 0.0007964 para C3. Ou seja, com *p-value* abaixo da limiar de 0.05.

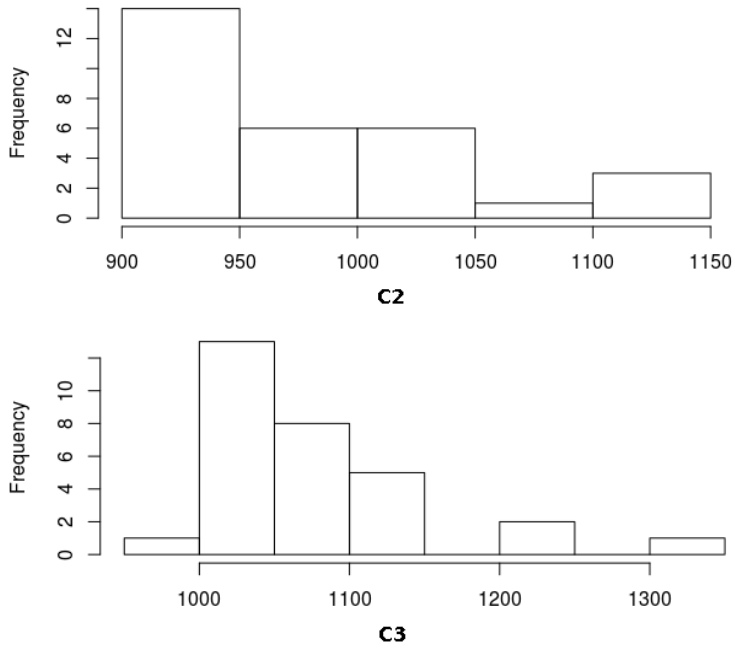

Figura 5.10: Distribuição não normal dos dados do tempo de resposta dos cenários C2 e C3.

Fonte: Elaborada pelo autor

Para avaliar a métrica *tempo de reposta* comparando os resultados obtidos nos cenários foi usado o teste de *Wilcoxon* [42]. O resultado do teste confirmou o *p-value* de 8.873e-06,

isto é, uma valor abaixo da limiar de 0.05. Diante disso, o *framework* não manteve o mesmo desempenho no processo de descoberta e composição de serviços. Por outro lado, isto se justifica, uma vez que para atender o cenário C2 é necessário apenas um serviço, enquanto para o cenário C3, exige que o *framework* realize uma composição de serviços.

Sumarizando os resultados

Na Tabela 5.6, estão sumarizados os resultados para os valores de mínimo e máximo, mediana , média e desvio padrão para a métrica *tempo de respostas* obtidos durante a execução do *framework* nos cenários apresentados na seção 4.6.

Tabela 5.6: Resumo das métricas tempo de resposta para todos os cenários.

Fonte: Elaborada pelo autor

	Mínimo	**1º Qt**	**Mediana**	**Média**	**Desvio Padrão**	**3º Qt**	**Máximo**
C1	130.0	136.2	151.5	150.9	15.12	163.5	182.0
C2	914.0	937.2	952.0	982.1	62.40	1033.5	1119.0
C3	993.0	1021.0	1068.0	1078.0	74.07	1111.0	1310.0
C4	990.0	1018.0	1030.0	1042.0	40.13	1046.0	1164.0
C5	989.0	1018.0	1029.0	1050.0	51.50	1056.0	1186.0
C6	1033.0	1062.0	1074.0	1099.0	59.57	1117.0	1226.0
C7	1037.0	1053.0	1075.0	1102.0	65.28	1119.0	1263.0

5.4 Resultados e Discussão

Nesse experimento foi adotada uma abordagem baseada em cenários que simulam um ambiente de execução real com a finalidade de avaliar os atributos de desempenho e Disponibilidade da solução proposta através da execução dos processo de descoberta e composição de Serviços Web Semânticos.

Para o atributo de *Desempenho*, os testes estatísticos demostram que o desempenho do *framework* proposto se manteve o mesmo durante a execução do processo de descoberta e composição de serviços mesmo em cenários de recuperação. Assim, destaca-se a Disponibilidade e Desempenho da abordagem adotada diante de diferentes cenários de execução.

Vale ressaltar que os grandes diferenciais da solução proposta como o modelo semântico adotado e o modelo de repositórios, que gerenciam os dados relacionados aos serviços e suas descrições semânticas através de um SGBD de alto desempenho orientado a documentos, contribuíram significativamente para o bom desempenho da aplicação. Pois, o processo de descoberta e composição de serviços não necessita manipular ontologias diretamente, o que é um gargalos em grande parte das aplicações atualmente. Além disso, com o modelo semântico adotado, o processo de construção de serviços semânticos foi simplificado, tornando o processo de construção e manutenção de aplicações semânticas mais simples e eficiente.

O outro atributo avaliado foi a Disponibilidade sob a perspectiva de tolerância a falhas. Para tal, foram estabelecidos cenários de recuperação com a inserção de falhas. A falha definida foi a desativação de um serviço e a métrica, o número de erros produzidos pela solução proposta. Durante a execução dos experimentos, nenhuma falha foi reportada, mesmo nos cenários onde serviços foram desativados ou cenários com requisições onde nenhum serviço ou composição atenderia aos parâmetros definidos. Portanto, destaca-se a boa capacidade de adaptabilidade e tolerância a falhas da solução proposta em tempo de execução.

Os resultados dos cenários complementares que buscaram avaliar desempenho foram semelhantes aos resultados dos cenários priorizados. Para os cenários C3 e C4, o desempenho se manteve o mesmo. Nesses cenários, a solução precisaria realizar uma composição entre dois serviços, com a diferença que para o cenário C4 haviam duas opções de serviços para a composição. Apenas na comparação entre os cenários C2 e C3 houve uma diferença significativa de desempenho. Fato esse que pode ser explicado pelo cenário C2 representar uma requisição cuja a resposta é um serviço único com *matching* direto e para o cenário C3, a requisição exigia a composição de dois serviços através de um *matching* indireto.

Por fim, a fim de discutir os objetivos estabelecidos inicialmente neste trabalho, estes serão reapresentados a seguir:

- Possibilitar a integração de algoritmos e/ou ferramentas de mineração de dados através de serviços;
- Promover o reuso de soluções de mineração de dados aplicada a diferentes domínios;
  - Esse objetivos foram alcançados através da adoção de uma abordagem baseada

em serviços para encapsular algoritmos/ferramentas de mineração que favorece o reuso e simplifica o processo de integração entre a solução proposta e os ambientes educacionais, pois agregam vantagens inerentes e bem conhecidas das abordagens orientadas à serviços, tais como: interoperabilidade, reuso, produtividade, entre outros. Desse modo, o processo de integração entre ambientes educacionais e a solução proposta é simplificado, sendo necessário adaptar o ambiente educacional para se comunicar com o *ServiceManager*, que é um serviço *Web* tradicional, e demandar suas requisições. Além disso, é possível adicionar novas funcionalidades/soluções encapsuladas em serviços que poderão ser reutilizadas por outros ambientes e/ou outras aplicações

- Possibilitar a adição, adaptação e/ou remoção de algoritmos/ferramentas de mineração de maneira dinâmica;
  - O processo de adição e/ou manipulação de novos serviços é simplificado através do uso de serviços *Web* semânticos. Através desse tipo de serviço é possível construir novos serviços e adicioná-los a solução proposta sem a necessidade de realizar nenhuma configuração previa. Isso acontece graças a descrição semântica que cada serviço recebe no momento da sua criação. Dessa maneira, o gerenciador de serviços consegue realizar todo o processo de descoberta, composição e invocação de forma automática e dinâmica.
- Simplificar o processo de construção de serviços *Web* semânticos;
- Simplificar e promover o desempenho do processo de descoberta, composição e invocação de serviços *Web* semânticos de forma automática e dinâmica;
  - A simplificação do processo de construção dos serviços semânticos, bem como o ganho no desempenho nos processos de descoberta e composição dos serviços são apoiadas com as opções pela adoção de um novo modelo semântico e do MongoDB para armazenar/gerenciar os dados relacionados aos serviços e as suas descrições semânticas. O novo modelo semântico adotado é mais simples que os demais e faz uso de arquivos do tipo *JSON* para descrever os parâmetros de entrada e saída dos serviço. Esse fato aliado a um SGBD orientado a documentos

de alta performance adotado na solução proposta torna o processo de construção e manipulação dos serviços mais simples e natural, promovendo o desempenho durante os processos de descoberta e composição dos serviços. Pois, esses processos precisam manipular dados relacionados aos serviços e as suas descrições semânticas.

Capítulo 6

Considerações Finais

Neste trabalho foi proposto um *framework* baseado em Serviços *Web* Semânticos aplicado ao domínio da Mineração de Dados. A solução proposta realiza todo processo de descoberta, composição e invocação de serviços de maneira automática e dinâmica. Os serviços implementados na aplicação encapsulam técnicas e algoritmos de mineração, pré-processamento e pós-processamento de dados disponíveis na *api* da ferramenta Weka e pode ser estendido para receber diferentes algoritmos/ferramentas de mineração. A solução proposta oferece vantagens, como: (i) Reutilização de algoritmos/soluções de mineração de dados; (ii) Simplifica o processo de integração entre ferramentas de mineração e ambientes educacionais; (iii) Adição de novos algoritmos de mineração e / ou técnicas de maneira dinâmica; (iv) Simplifica o processo de criação e manipulação de serviços semânticos; (v) Promove o desempenho no processo de descoberta, composição e invocação de serviços.

Entre os principais diferenciais do *framework* proposto, destacam-se a adoção de um novo modelo semântico baseado em REST e de uma abordagem baseada em repositórios que simplificam o processo de construção e manipulação de serviços semânticos. Além disso, tais decisões promovem o desempenho dos processos de descoberta, composição e invocação, e, consequentemente, promovem o construção e manutenção de sistemas baseados em serviços semânticos. Ressalta-se, portanto, que os ganhos citados na abordagem proposta neste trabalho são relevantes para o desenvolvimento de soluções nos domínios de Mineração de Dados Educacional e de serviços *Web* semânticos.

A avaliação do *framework* proposto foi realizada por meio de um experimento baseado em cenários que buscou avaliar os atributos de qualidade de Desempenho e Disponibilidade.

Os resultados demonstraram um bom de desempenho da solução proposta, que chegou a realizar todo o processo de descoberta e composição em aproximadamente 900 ms. Além disso, a solução proposta manteve o bom desempenho em diferentes cenários, incluindo cenários de recuperação onde foram realizadas a inserção de falhas. Diante desses cenários, não foram reportadas falhas, o que ressalta a eficiência do sistema e do seu mecanismo de tolerância a falhas.

6.1 Limitações e Trabalhos Futuros

Como limitações e trabalhos futuros, para o *framework* apresentando nesta dissertação pretende-se:

- Integrar algoritmos de mineração de outras ferramentas, tais como: R, Apache Mahout, entre outros;
- Implementar uma interface Web para que usuários possam gerenciar os serviços;
- Realizar a integração do *framework* em cenário real de utilização a fim de avaliar os custos de integração e o comportamento da aplicação;
- Avaliar outros atributos de qualidade além do Desempenho e Confiabilidade, tais como: Portabilidade, Manutenibilidade, entre outros;

Bibliografia

[1] H. Barros. Um middleware adaptável para descoberta, composição e invocação automática de serviços web semântico. Master's thesis, Universidade Federal de Alagoas, 2011.

[2] Douglas K Barry. *Web services, service-oriented architectures, and cloud computing*. Morgan Kaufmann, 2003.

[3] Victor R Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, 1992.

[4] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[5] ACPL Carvalho, F de Ponce de Leon, et al. Grandes desafios da pesquisa em computação no brasil–2006–2016. *São Paulo: Sociedade Brasileira de Computação*, 2006.

[6] John Domingue, Dieter Fensel, and James A Hendler. *Handbook of semantic web technologies*. Springer Science & Business Media, 2011.

[7] Heitor José dos Santos Barros. *Um Modelo Semântico para Compartilhamento de Recursos Educacionais*. PhD thesis, Universidade Federal de Campina Grande, 2016.

[8] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[9] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34, 1996.

[10] Dieter Fensel and Christoph Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[11] ERICH GAMMA, R Helm, R Johnson, and J Vlissides. Padrões de projeto–soluções reutilizáveis de software orientado a objetos, 2004, ed.

[12] Islam H Harb, Salah Abdel-Mageid, Hassan Farahat, and MS Farag. Enhanced semantic web layered architecture model. In *Proceedings of the 10th WSEAS international conference on applied informatics and communications, and 3rd WSEAS international conference on Biomedical electronics and biomedical informatics*, pages 341–347. World Scientific and Engineering Academy and Society (WSEAS), 2010.

[13] Evandro Costa Jonathas Magalhes Patrick Brito Heitor Barros, Tarsis Marinho. A synonyms dictionary approach in semantic web services composition. In *Proceedings of the IADIS International Conference WWW/Internet 2013*, 2013.

[14] Michael N Huhns and Munindar P Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet computing*, 9(1):75–81, 2005.

[15] IBM. Standards and web services.

[16] Ricardo Shoiti Ikematu. Gestão de metadados: sua evolução na tecnologia da informação. *DataGramaZero-Revista de Ciência da Informação*, 2(6), 2001.

[17] Javatpoint. What is web service.

[18] Rick Kazman, Mark Klein, and Paul Clements. Atam: Method for architecture evaluation. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2000.

[19] Matthias Klusch, Patrick Kapahnke, Stefan Schulte, Freddy Lecue, and Abraham Bernstein. Semantic web service search: a brief survey. *KI-Künstliche Intelligenz*, 30(2):139–147, 2016.

[20] Lukasz A Kurgan and Petr Musilek. A survey of knowledge discovery and data mining process models. *The Knowledge Engineering Review*, 21(1):1–24, 2006.

[21] David Linthicum. Chapter 1: Service oriented architecture (soa) soas are like snowflakes no two are alike. 2008.

[22] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22:2007–04, 2004.

[23] David Martin, Mark Burstein, Drew Mcdermott, Sheila Mcilraith, Massimo Paolucci, Katia Sycara, Deborah L Mcguinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to web services with owl-s. *World Wide Web*, 10(3):243–277, 2007.

[24] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[25] S. A. McIlraith, T. C. Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, Mar 2001.

[26] Behrouz Minaei and Parinaz Saadat. Soap serialization performance enhancement, design and implementation of a middleware. *arXiv preprint arXiv:0911.0488*, 2009.

[27] Snehal Mumbaikar, Puja Padiya, et al. Web services based on soap and rest principles. *International Journal of Scientific and Research Publications*, 3(5):1–4, 2013.

[28] Tom Narock, Victoria Yoon, and Sal March. A provenance-based approach to semantic web service description and discovery. *Decision Support Systems*, 64:90–99, 2014.

[29] Jiantao Pan. Software reliability, 1999.

[30] Dumitru Roman, Uwe Keller, Holger Lausen, Jos De Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied ontology*, 1(1):77–106, 2005.

[31] Cristobal Romero and Sebastian Ventura. Educational data mining: A survey from 1995 to 2005. *Expert systems with applications*, 33(1):135–146, 2007.

[32] Cristóbal Romero and Sebastián Ventura. Educational data mining: a review of the state of the art. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(6):601–618, 2010.

[33] Cristobal Romero and Sebastian Ventura. Data mining in education. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(1):12–27, 2013.

[34] Alexandre Saudate. *SOA aplicado: Integrando com web services e além*. Editora Casa do Código, 2014.

[35] Mutange Kennedy Senagi, Okeyo George, Cheruiyot Wilson, Sati Arthur, and Kalunda Jades. A review of soap performance optimization techniques to improve communication in web services in loosely coupled systems. *International Journal of Computer Science Issues (IJCSI)*, 11(2):142, 2014.

[36] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[37] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. Web services composition: A decades overview. *Information Sciences*, 280:218–238, 2014.

[38] Munindar P Singh and Michael N Huhns. *Service-oriented computing: semantics, processes, agents*. John Wiley & Sons, 2006.

[39] T. Souza. Um framework para mineração de dados educacionais basedo em serviçõs semânticos. Master's thesis, Universidade Federal de Alagoas, 2011.

[40] Ricardo Voigt and Osmar de Oliveira Braz Junior. Análise de desempenho de arquitetura soap e rest para comunicação entre sistemas. *Anais SULCOMP*, 8, 2017.

[41] W3C. Web services architecture requirements, 2004.

[42] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[43] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[44] Jia Zhang and L-J Zhang. Criteria analysis and validation of the reliability of web services-oriented systems. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[45] Marta Zorrilla and Diego García-Saiz. A service oriented architecture to provide data mining services for non-expert data miners. *Decision Support Systems*, 55(1):399–411, 2013.

Apêndice A

Apêndice

A.1 Instruções para criação de Serviços Web Semânticos de Mineração de Dados

Este tutorial tem a finalidade de mostrar passo a passo, o processo que deve ser realizado para criação de Serviços Web Semânticos de Mineração de Dados. O serviço criado esta disponível: https://github.com/michelmiranda/ontology. A seguir são detalhados os passos para a construção de um serviço Web semântico e as ferramentas utilizadas na sua construção.

Para criação do serviço são necessários fazer o download dos arquivos:

- Ferramenta Protégé (https://protege.stanford.edu/)
- Ontologia de Serviços (https://github.com/michelmiranda/ontology)
- Ontologia de Mineração (https://github.com/michelmiranda/ontology)

A.2 Passo 1 - Importando a ontologia do serviço.

As Figuras A.1, A.2, A.3, A.4, A.6 e A.7, a seguir, mostram o detalhamento de todo o processo. Faça o processo seguindo a ordem que as figuras estão apresentadas.

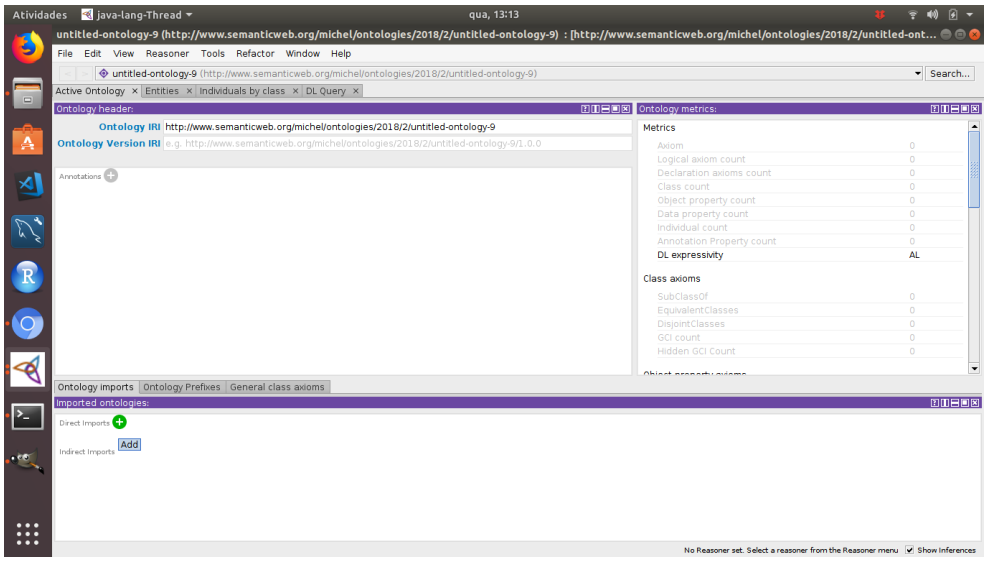

Figura A.1: Clique nessa opção ferramenta Protégé.

Fonte: Elaborada pelo autor

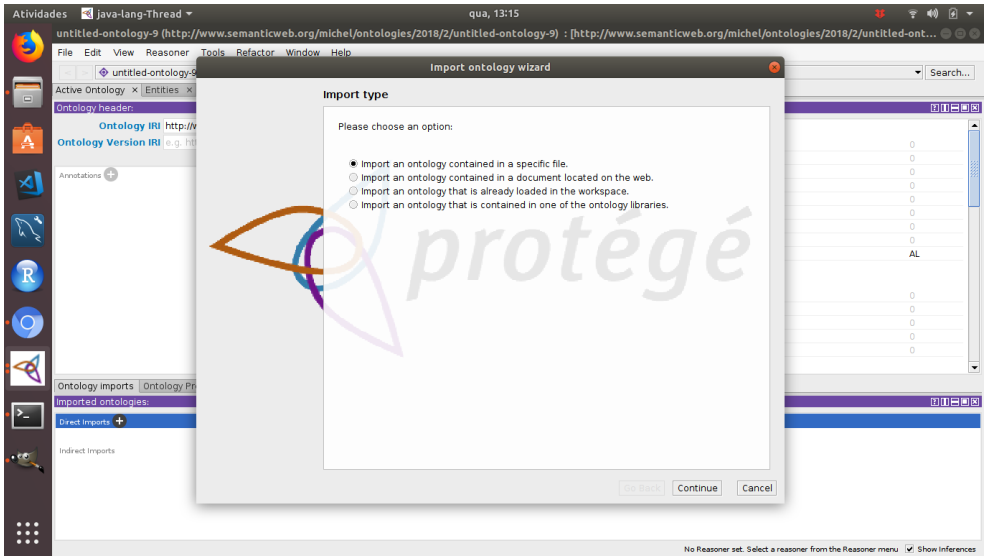

Figura A.2: Selecionar essa opção.

Fonte: Elaborada pelo autor

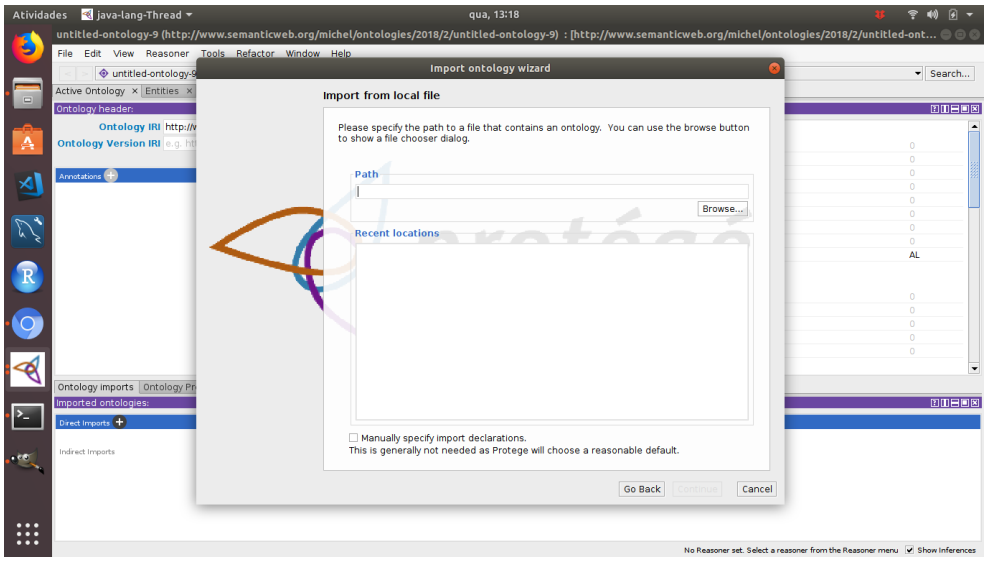

Figura A.3: Informar o diretório onde foi feio o download da ontologia.

Fonte: Elaborada pelo autor

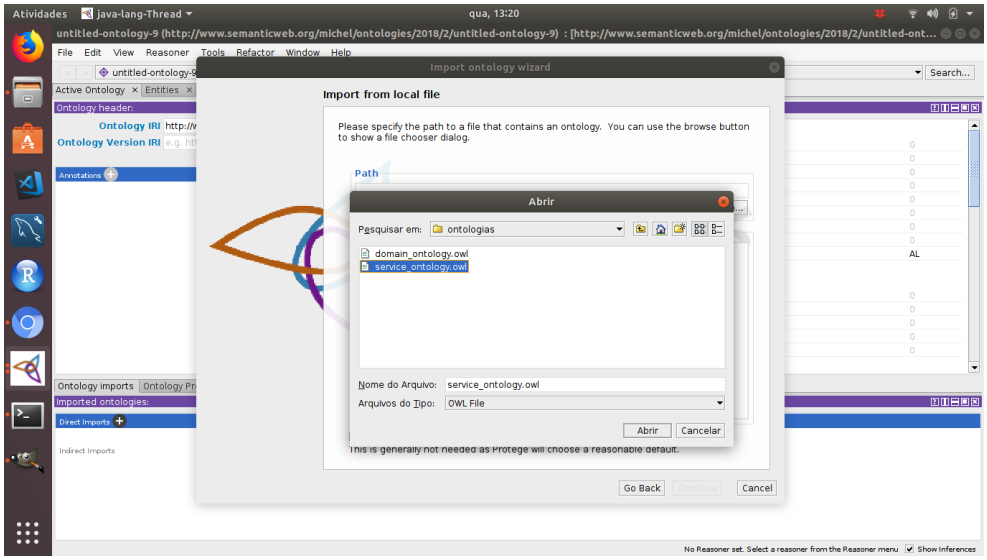

Figura A.4: Selecionar a ontologia.

Fonte: Elaborada pelo autor

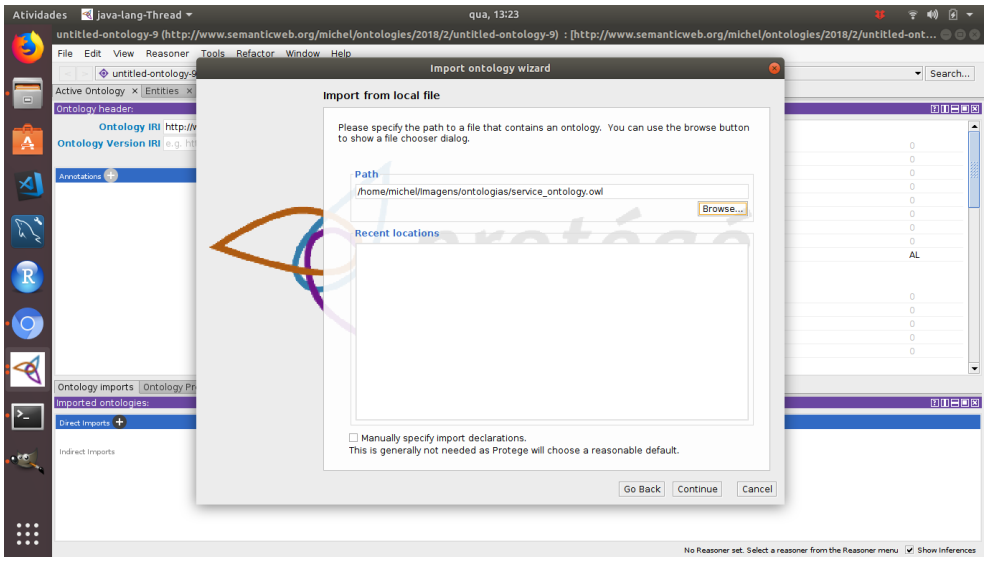

Figura A.5: Clique em continuar para prosseguir.

Fonte: Elaborada pelo autor

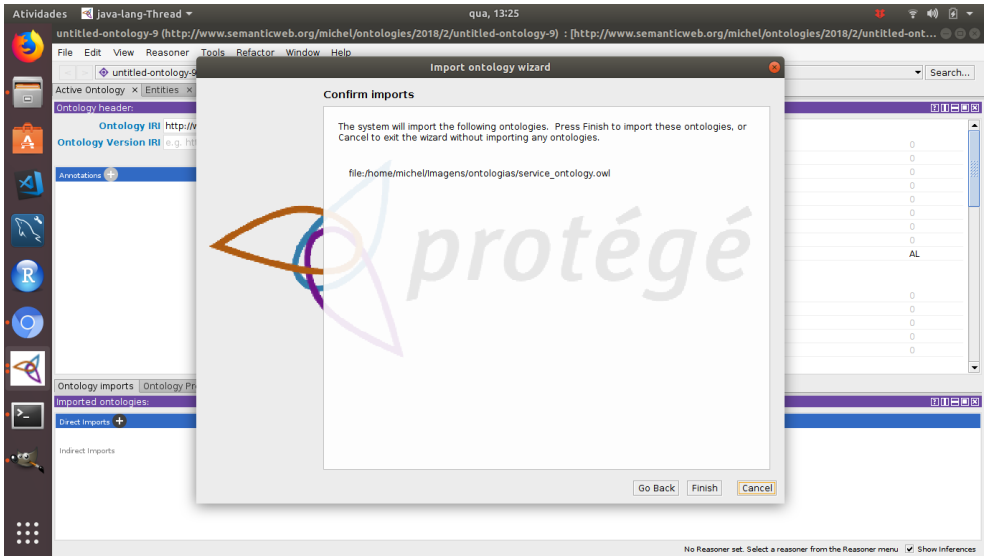

Figura A.6: Finalizando a importação da ontologia.

Fonte: Elaborada pelo autor

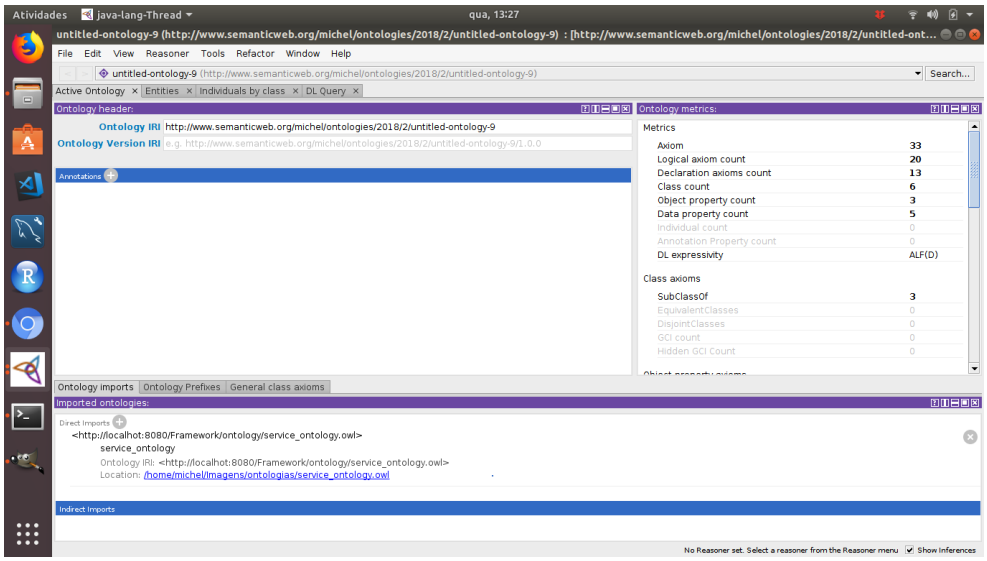

Figura A.7: Resultado da importação.

Fonte: Elaborada pelo autor

A.3 Passo 2 - Criando ontologia do serviço.

As Figuras A.8, A.9, A.10, A.11, A.12, A.13, A.14, A.15,A.16,A.13,A.13,A.13, a seguir, mostram o detalhamento de todo o processo. Faça o processo seguindo a ordem que as figuras estão apresentadas.

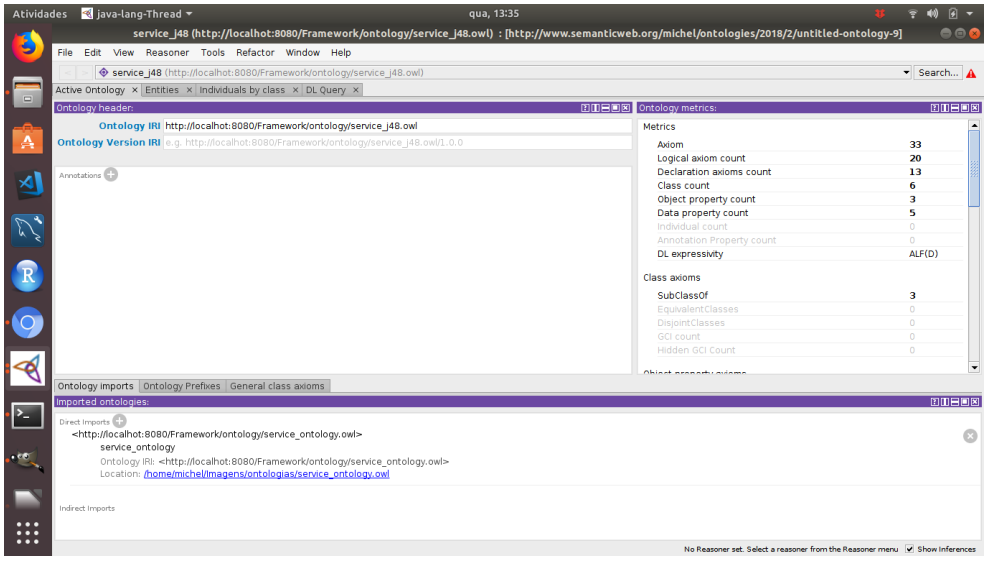

Figura A.8: Criando o J48Service.

Fonte: Elaborada pelo autor

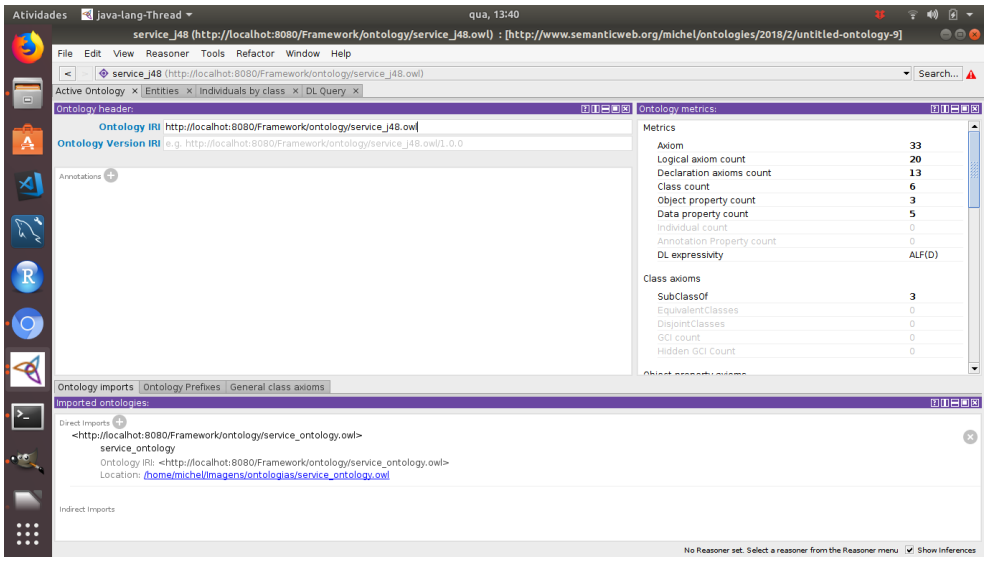

Figura A.9: Entries.

Fonte: Elaborada pelo autor

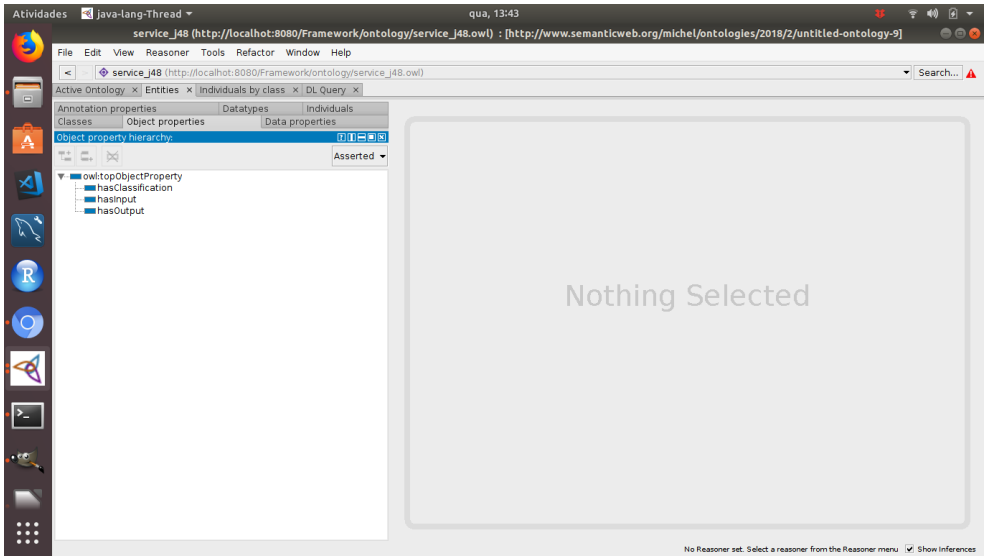

Figura A.10: Opções de Object Properteis.

Fonte: Elaborada pelo autor

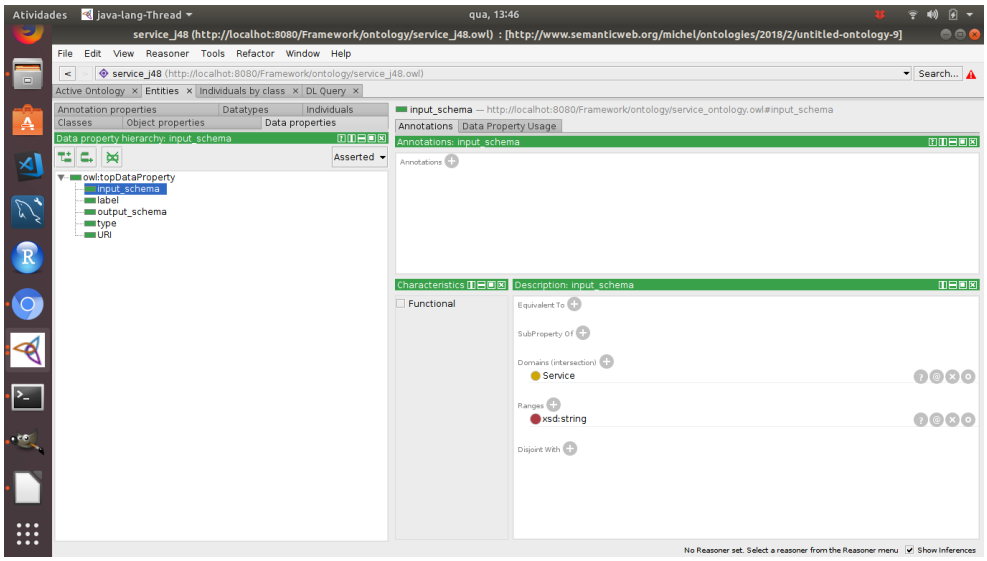

Figura A.11: Data Properteis J48Service.

Fonte: Elaborada pelo autor

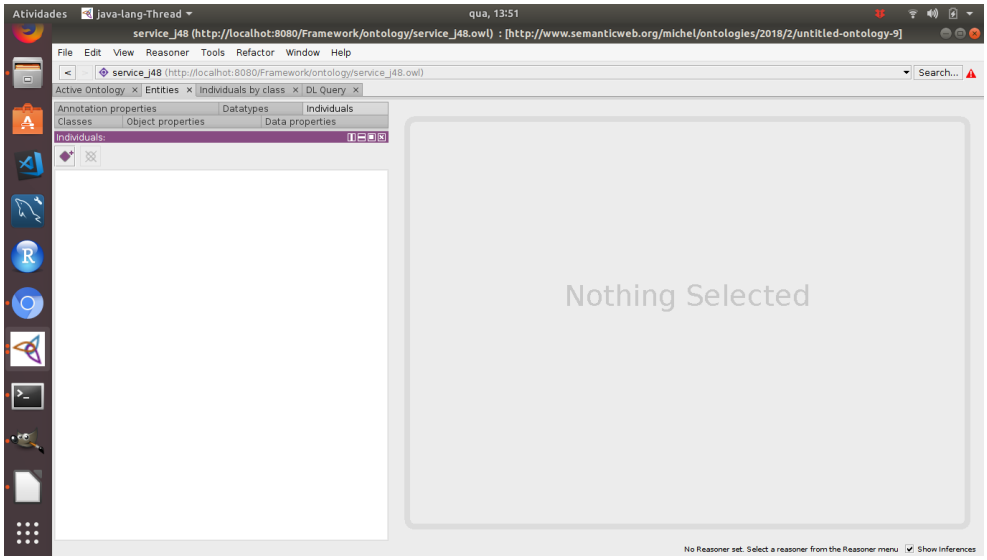

Figura A.12: Clicar na opção Individuals na parte superior da ferramenta Protégé.

Fonte: Elaborada pelo autor

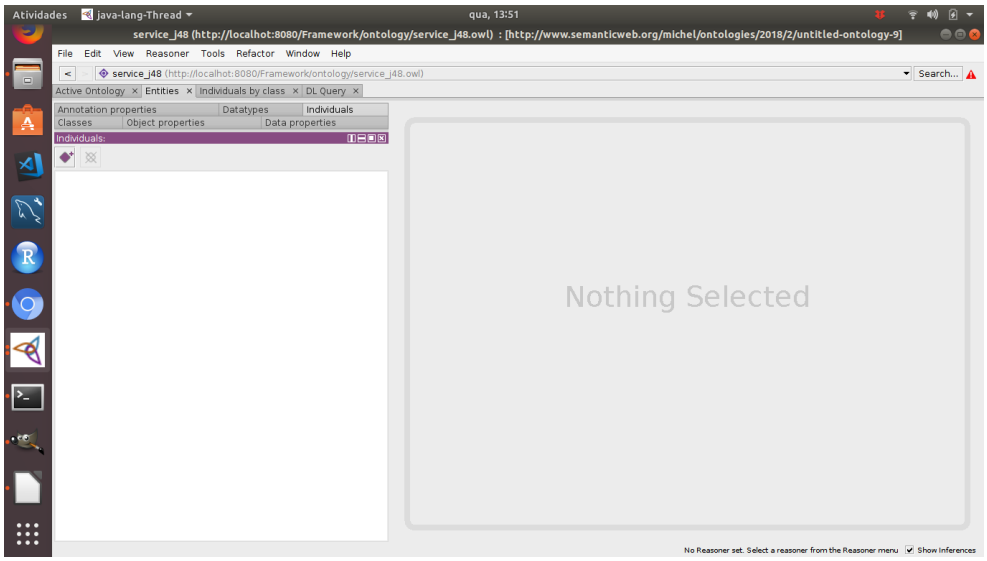

Figura A.13: Definindo o parâmetro de entrada do J48Service.

Fonte: Elaborada pelo autor

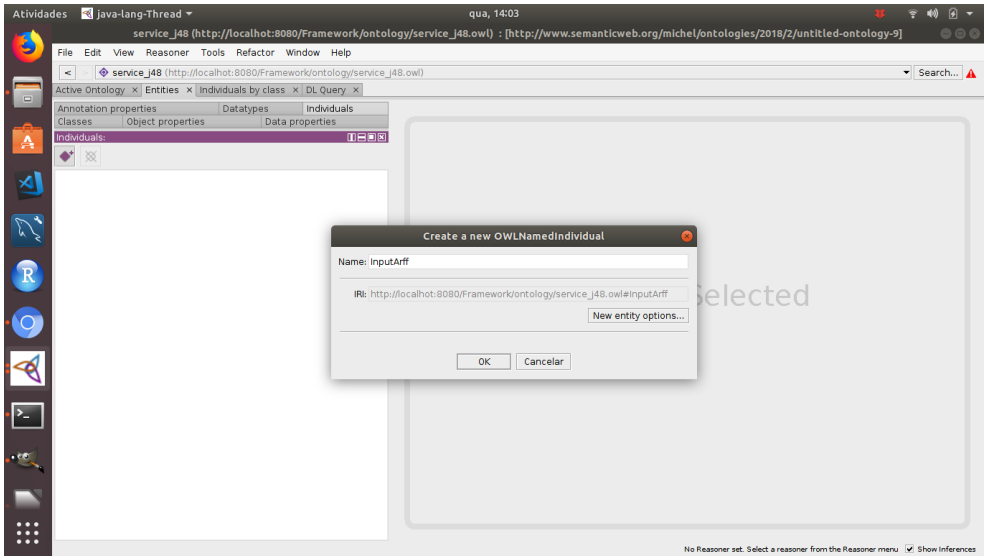

Figura A.14: Definindo o nome do parâmetro de entrada do J48Service.

Fonte: Elaborada pelo autor

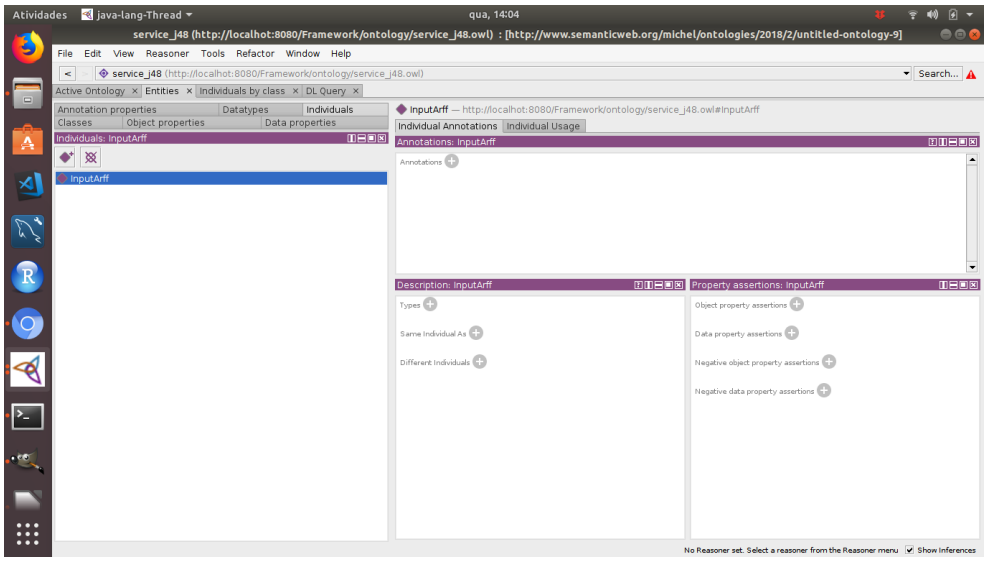

Figura A.15: Definindo o Types e o Data property assertions do InputArff.

Fonte: Elaborada pelo autor

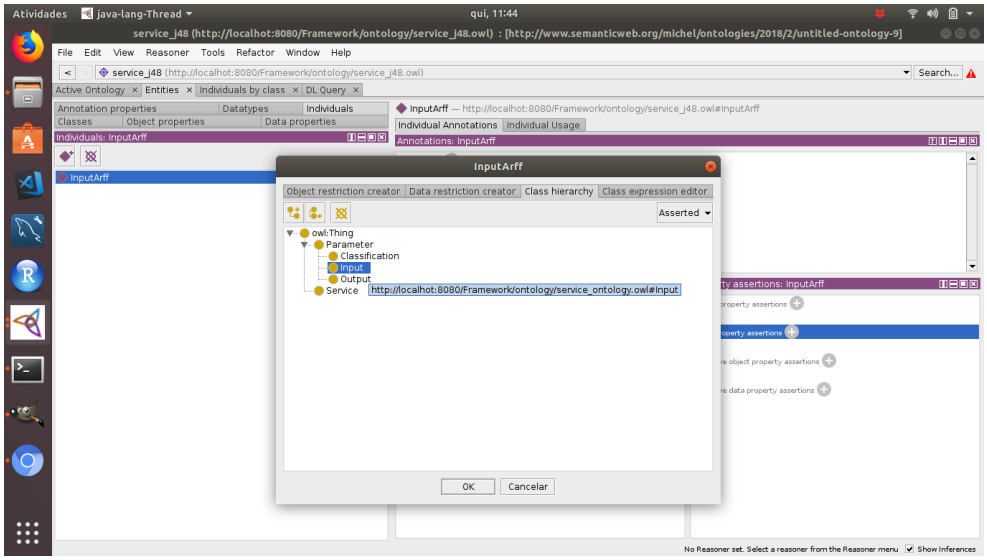

Figura A.16: Parâmetro de entrada do J48Service.

Fonte: Elaborada pelo autor

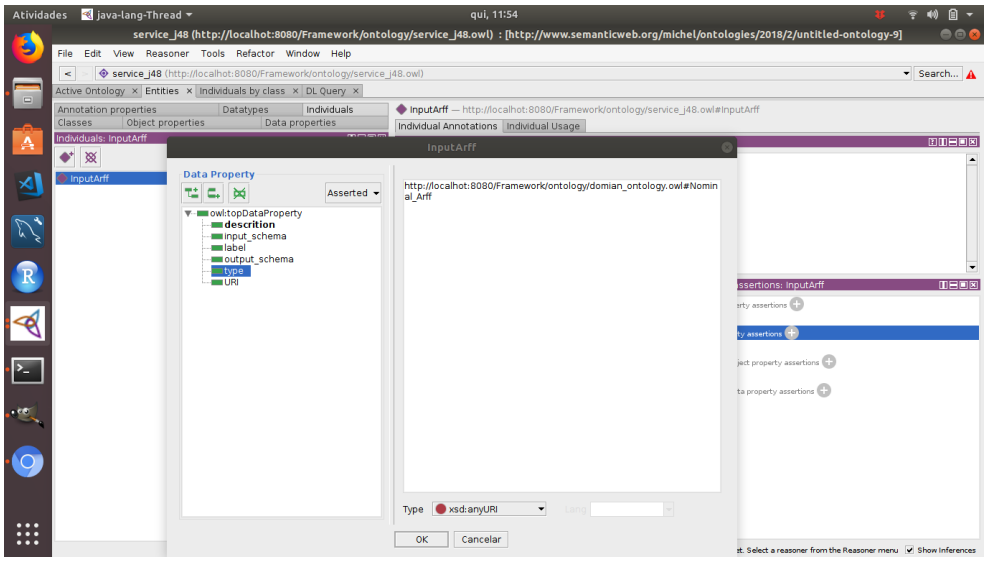

Figura A.17: Definindo o tipo AnyURi do parâmetro entrada do J48Service.

Fonte: Elaborada pelo autor

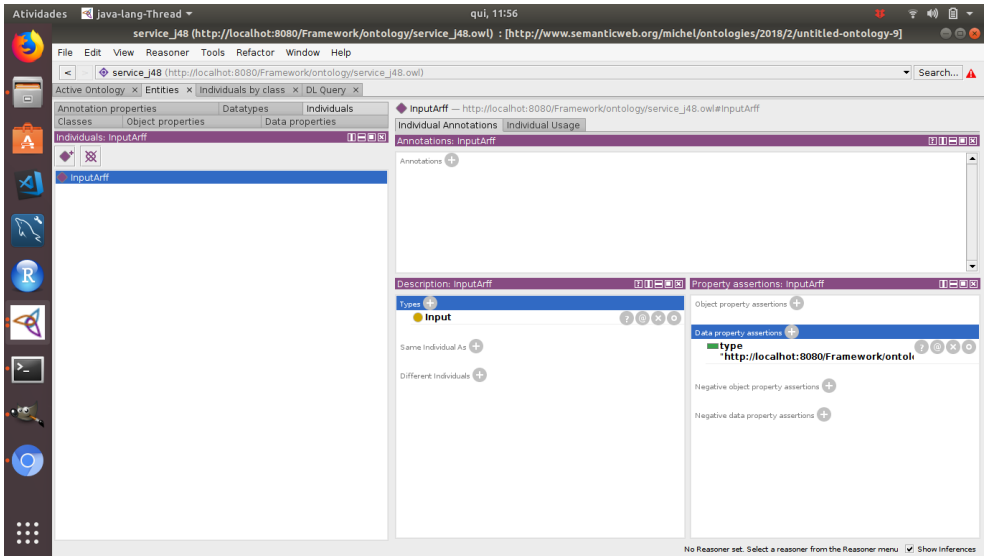

Figura A.18: Resultado após definir o Types e o Data property assertions do InputArff.

Fonte: Elaborada pelo autor

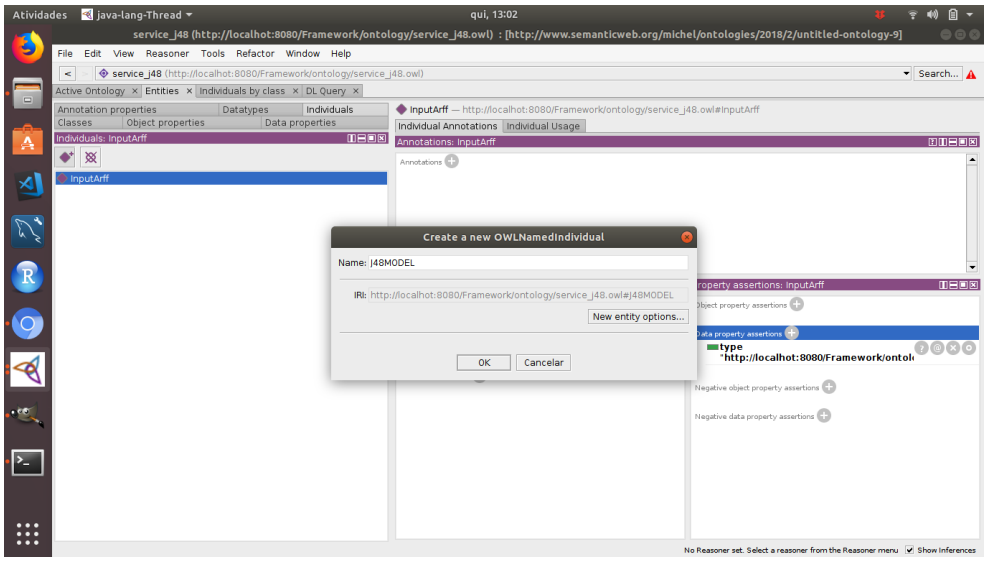

Figura A.19: Parâmetro de saída é J48MODEL.

Fonte: Elaborada pelo autor

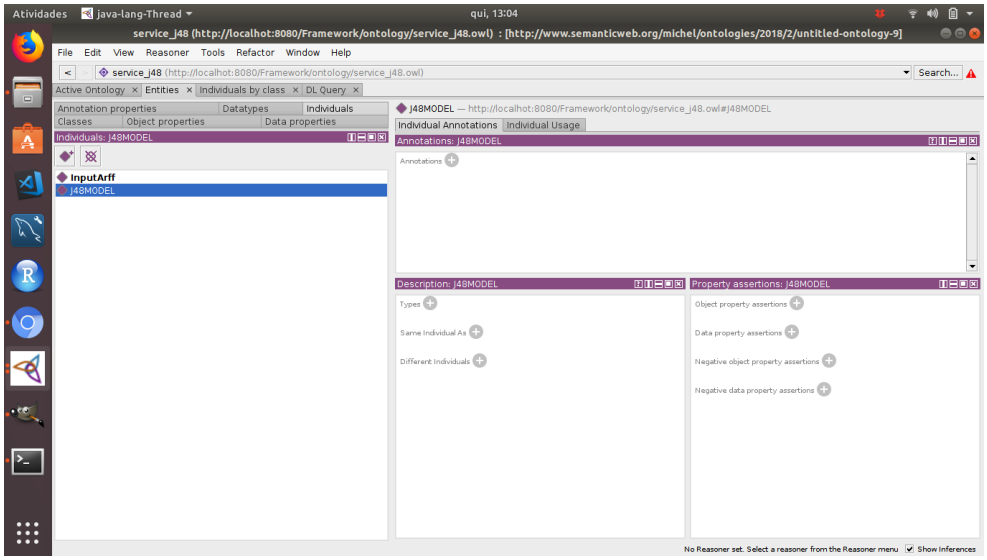

Figura A.20: Resultado do J48MODEL.

Fonte: Elaborada pelo autor

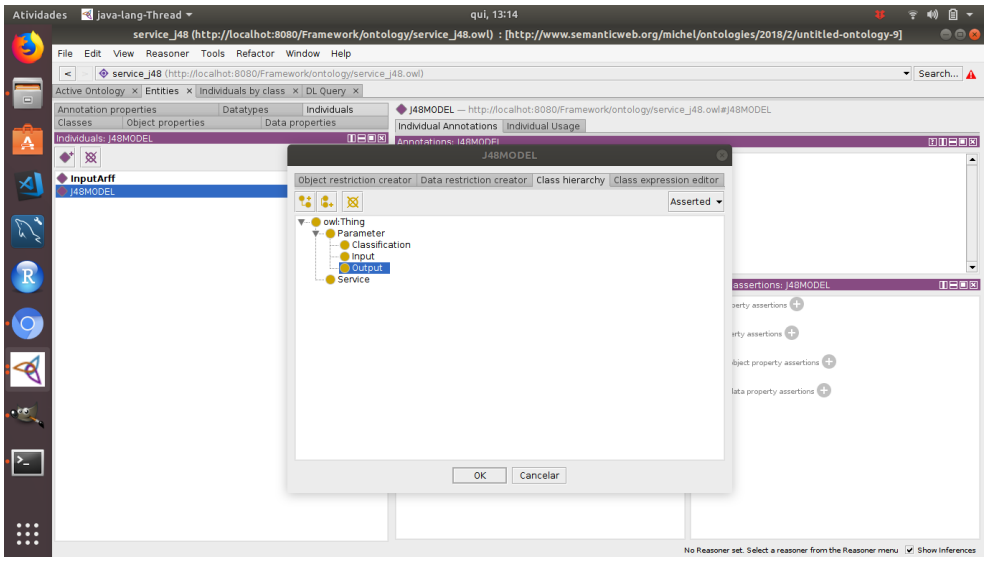

Figura A.21: Definindo o Type do J48MODEL

Fonte: Elaborada pelo autor

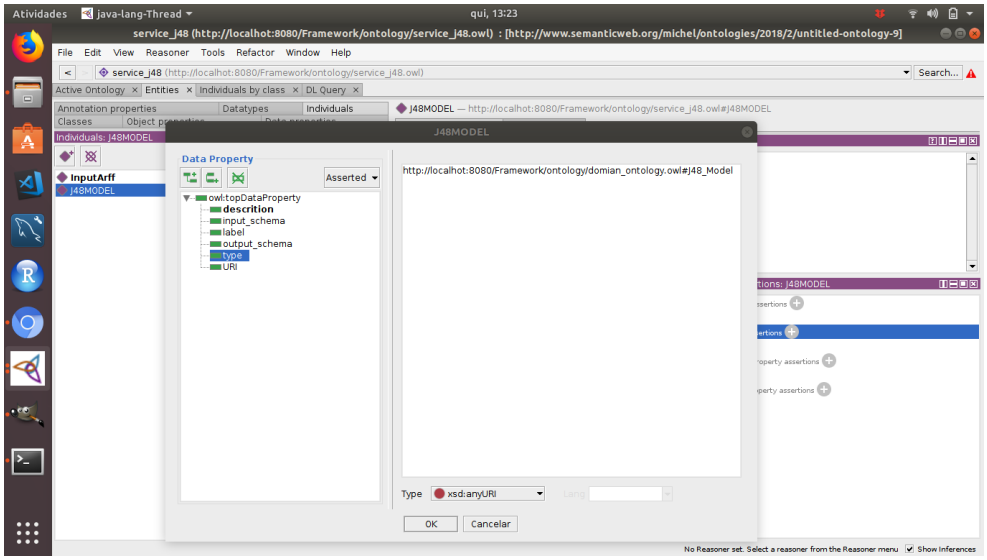

Figura A.22: Data property assertions do J48MODEL.

Fonte: Elaborada pelo autor

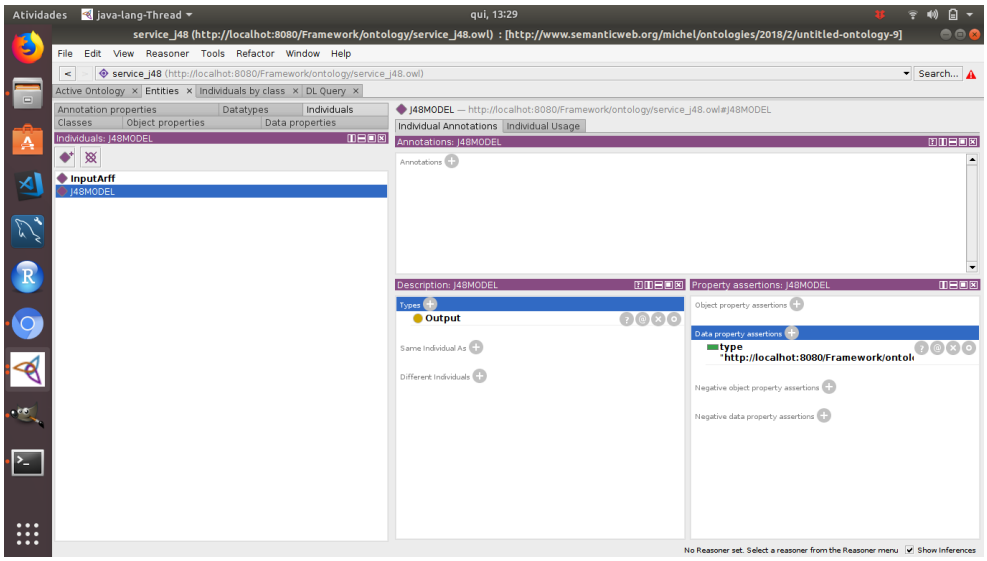

Figura A.23: Resultado do Data property assertions do J48MODEL.

Fonte: Elaborada pelo autor

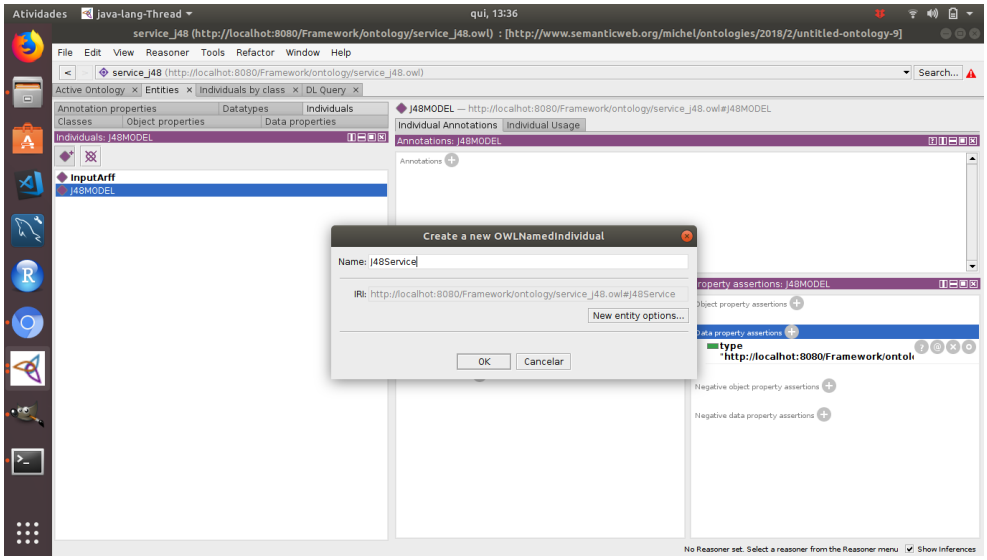

Figura A.24: Individuo com o nome J48Service.

Fonte: Elaborada pelo autor

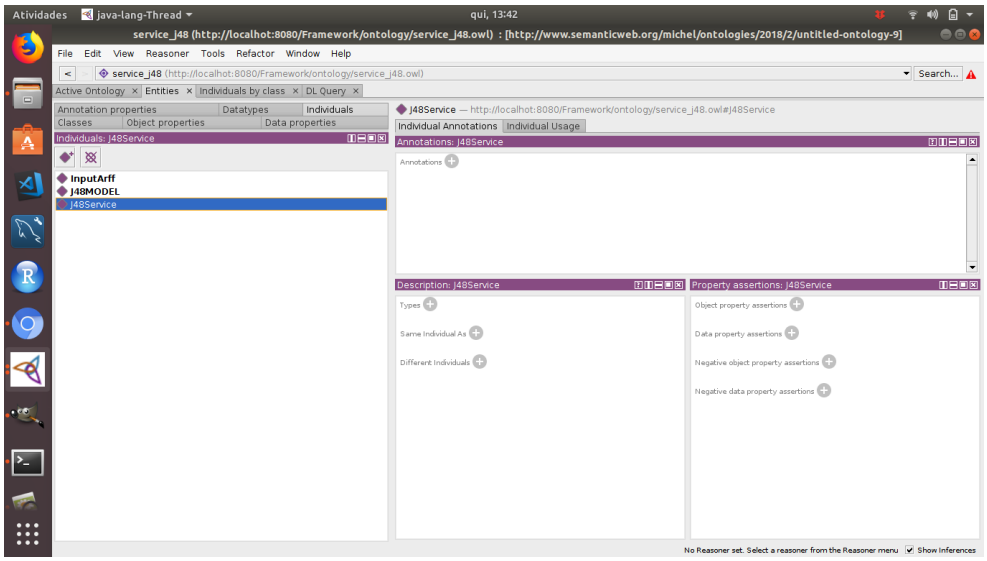

Figura A.25: Resultado do J48Service.

Fonte: Elaborada pelo autor

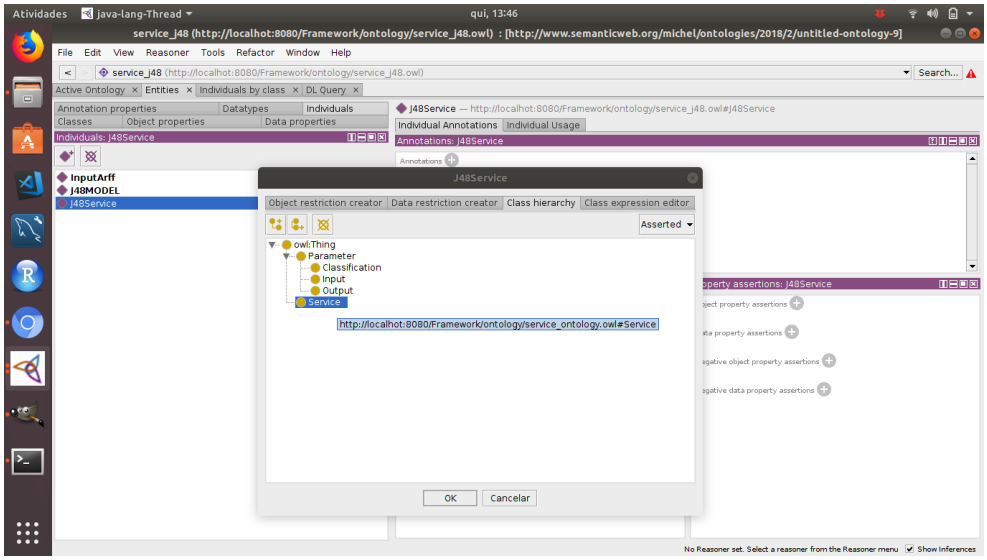

Figura A.26: Definidno os Types do serviço.

Fonte: Elaborada pelo autor

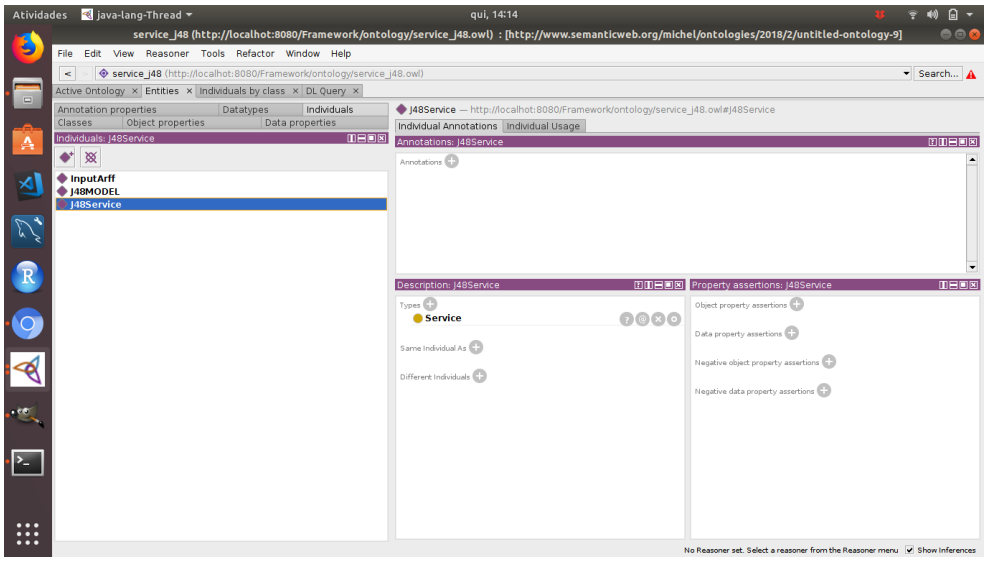

Figura A.27: Tipo do serviços J48Service definido.

Fonte: Elaborada pelo autor

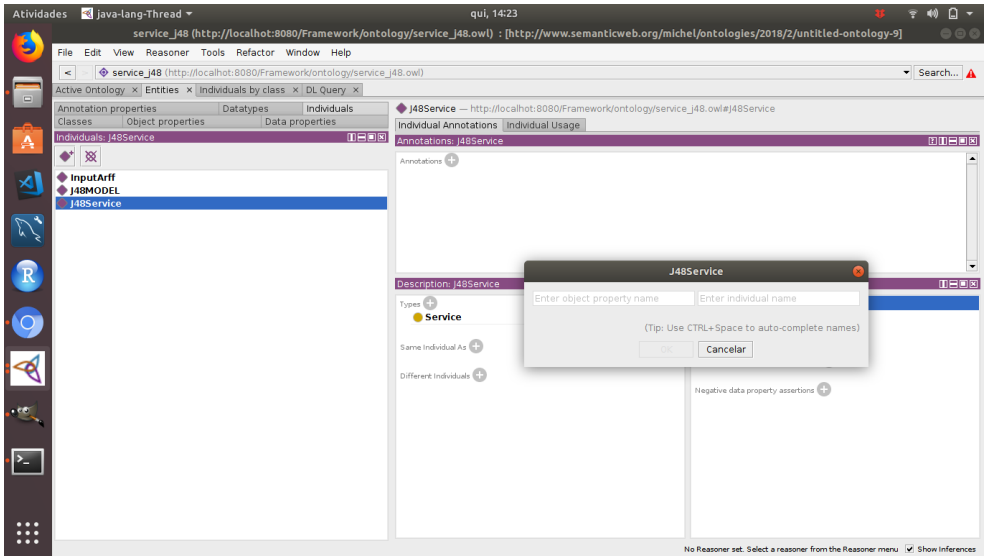

Figura A.28: Object property assertions do J48Service.

Fonte: Elaborada pelo autor

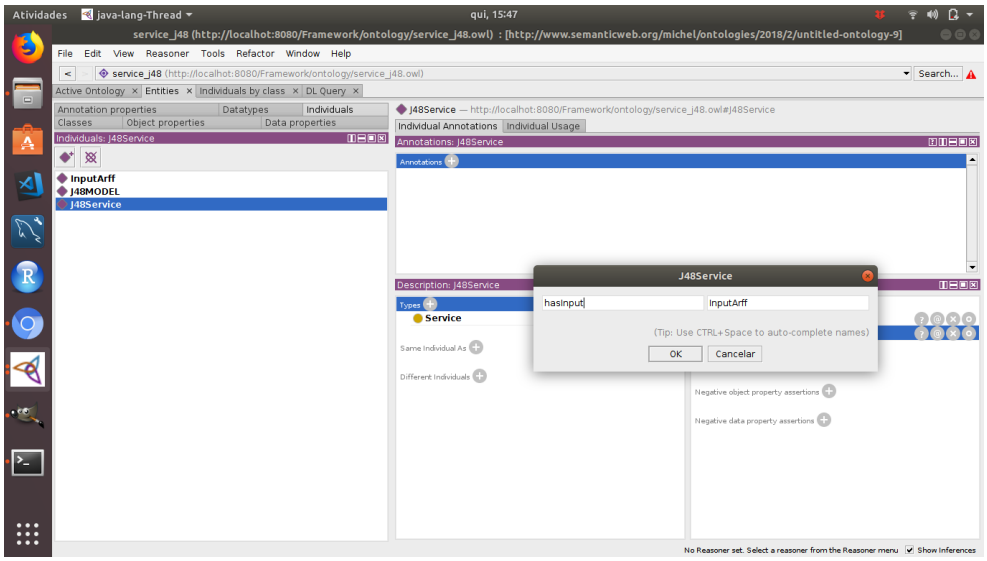

Figura A.29: HasIpunt e o hasOutput do J48Service.

Fonte: Elaborada pelo autor

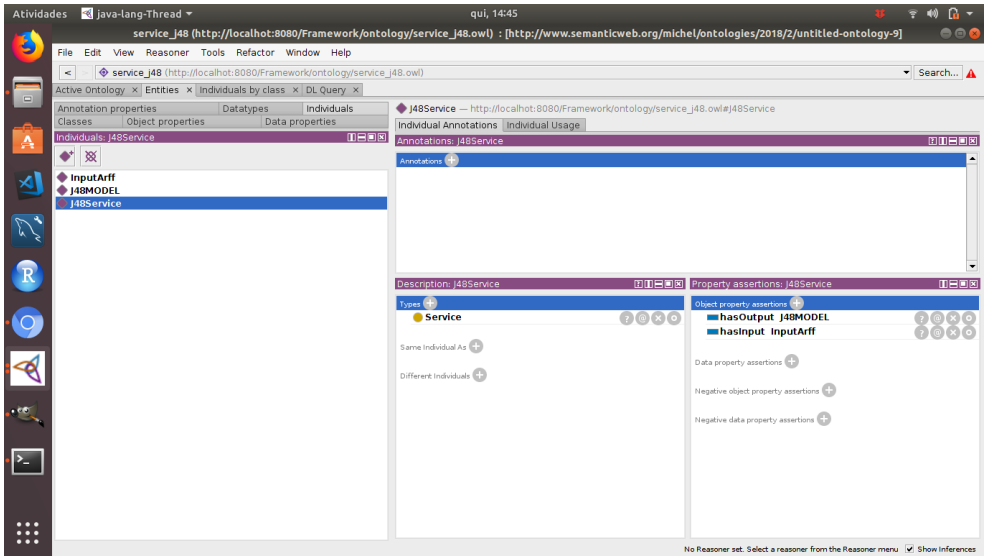

Figura A.30: Resultado dos hasInput e hasOutput definidos para J48Service.

Fonte: Elaborada pelo autor

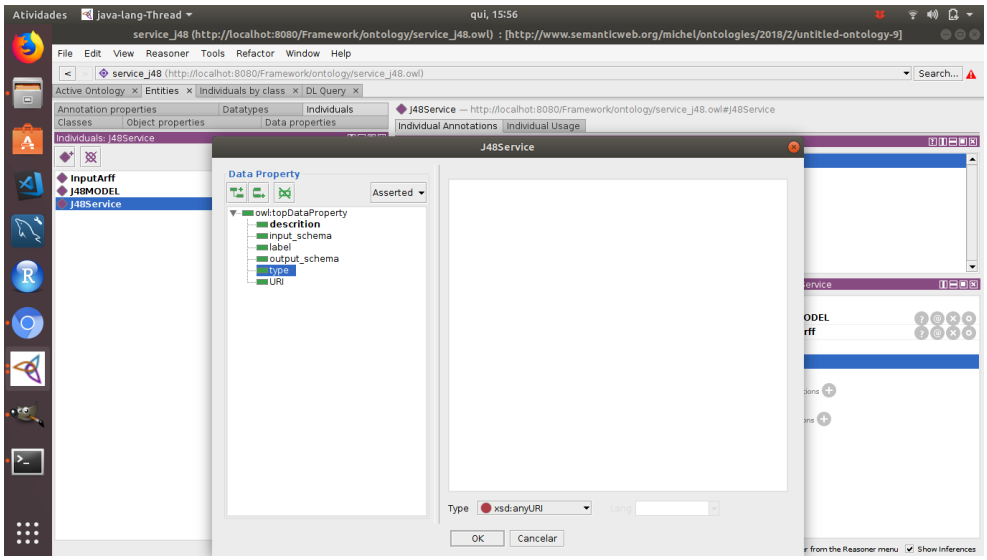

Figura A.31: Data property assertion do J48Service

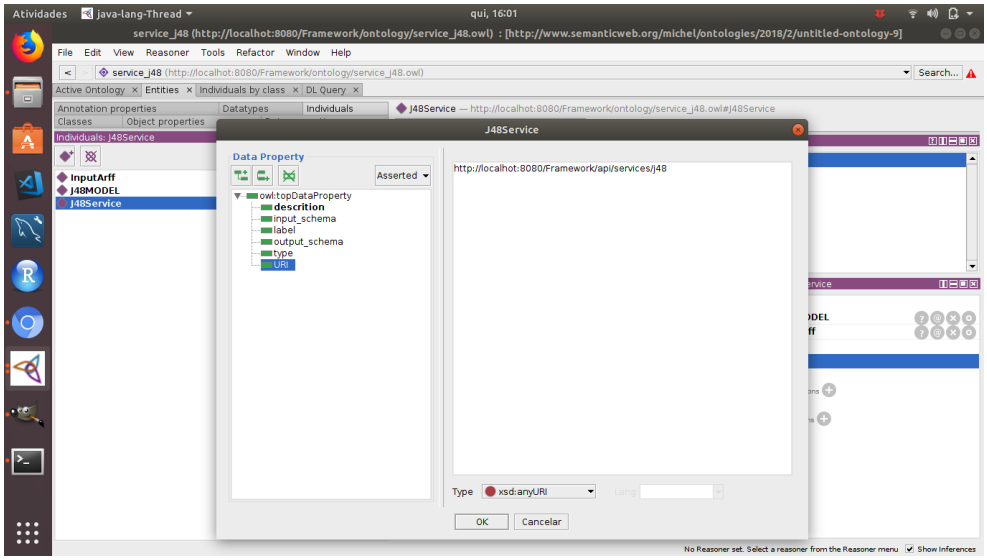

Figura A.32: Data property assertion URI do J48Service.

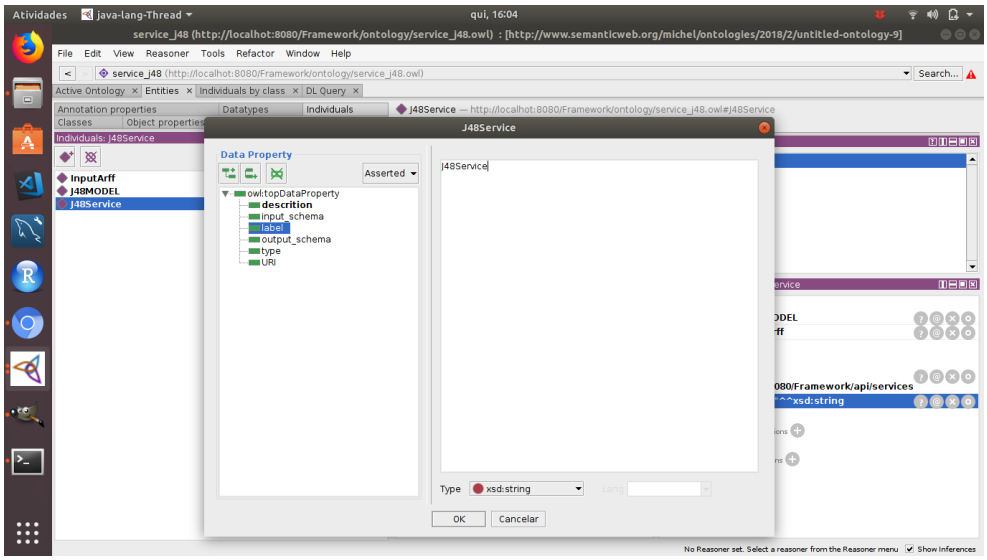

Figura A.33: Label J48Service.

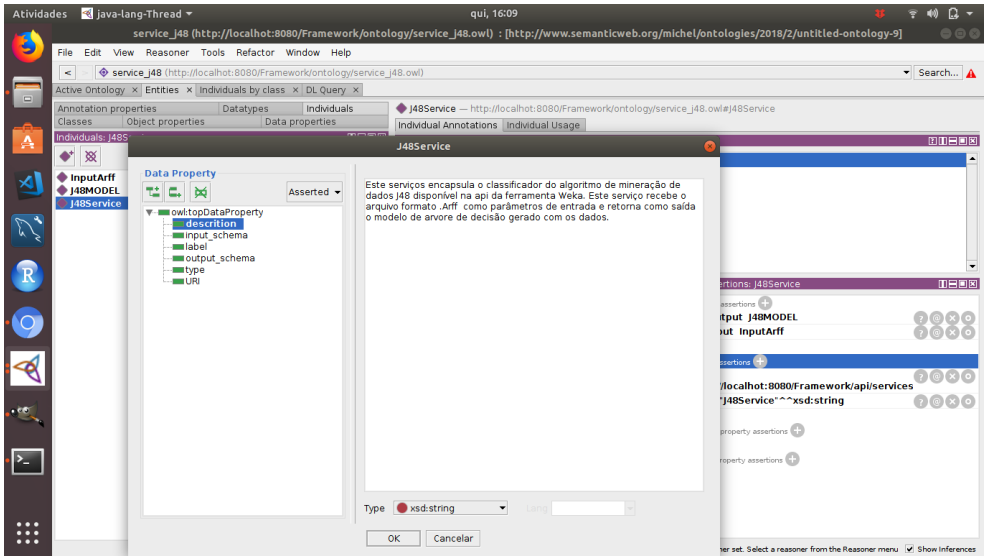

Figura A.34: Descrition do J48Service.

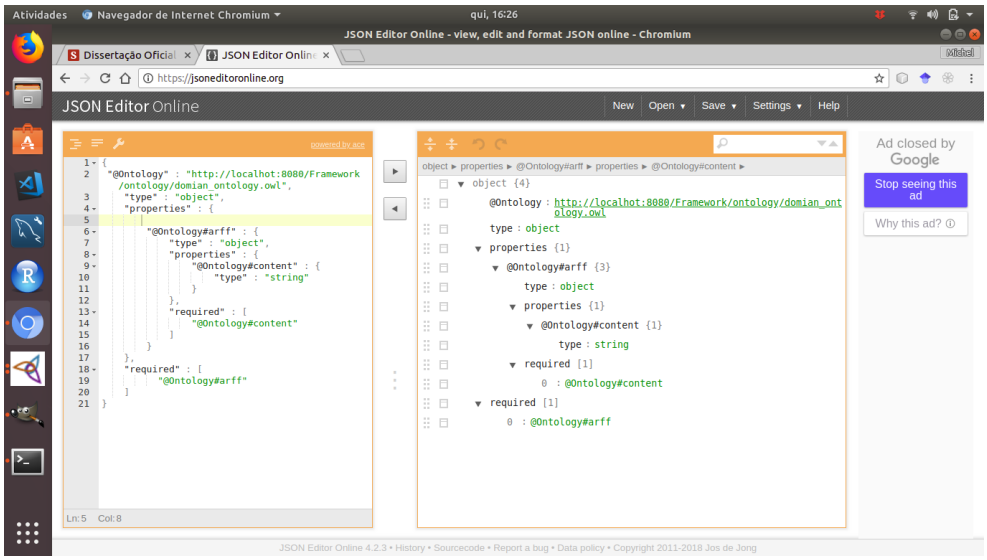

Figura A.35: InputSchema que define o formado de entrada do J48Service.

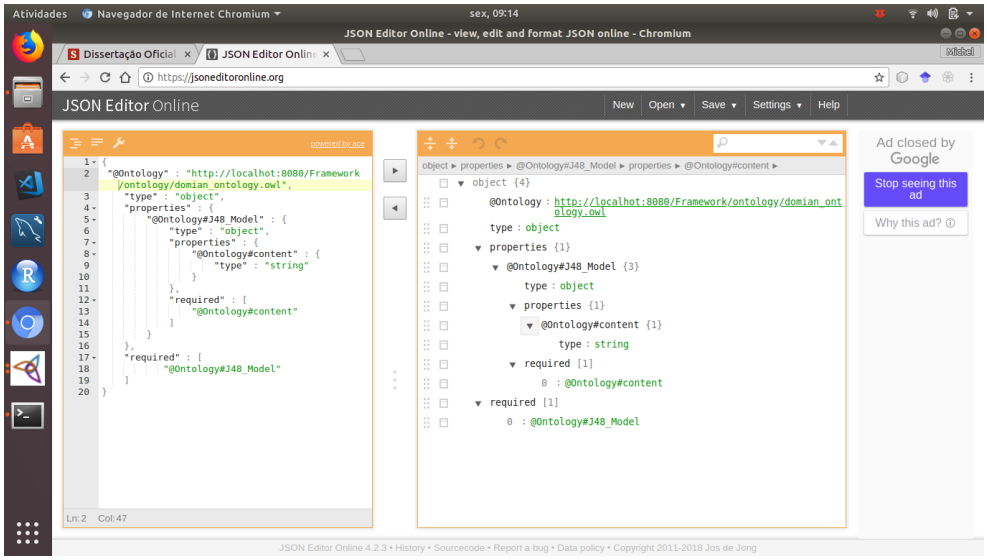

Figura A.36: OutputSchema do J48Service.

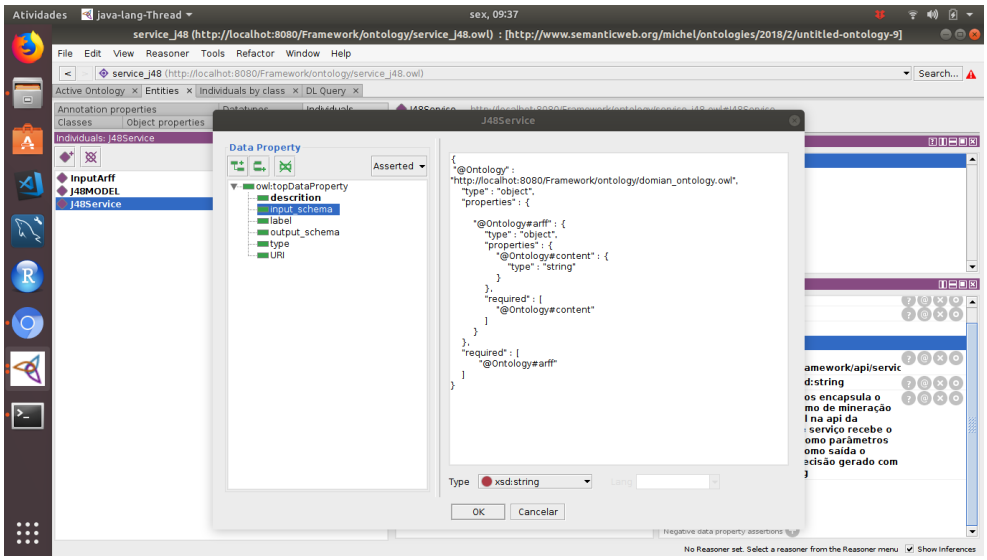

Figura A.37: InputSchema do J48Service.

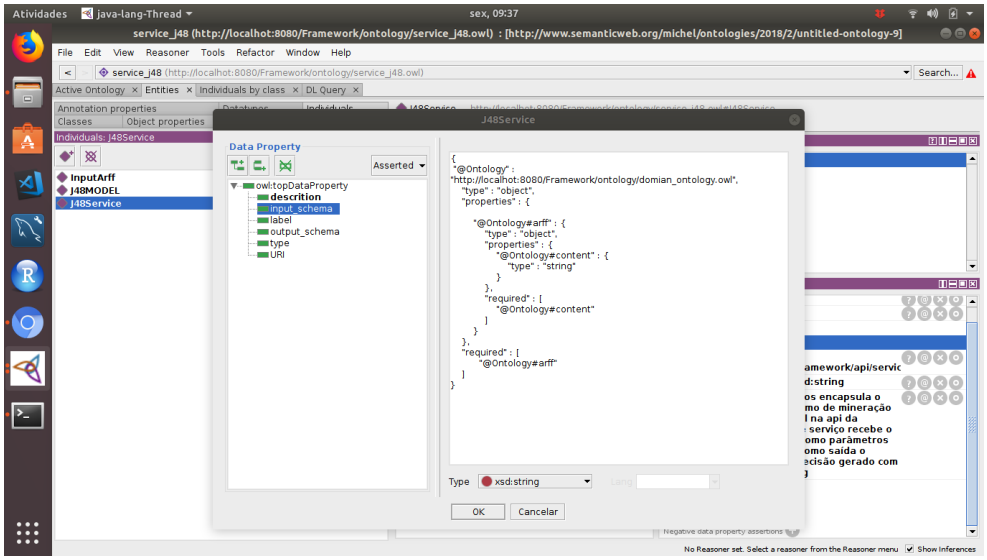

Figura A.38: Definir o Data property assertion OutputSchema do J48Service.

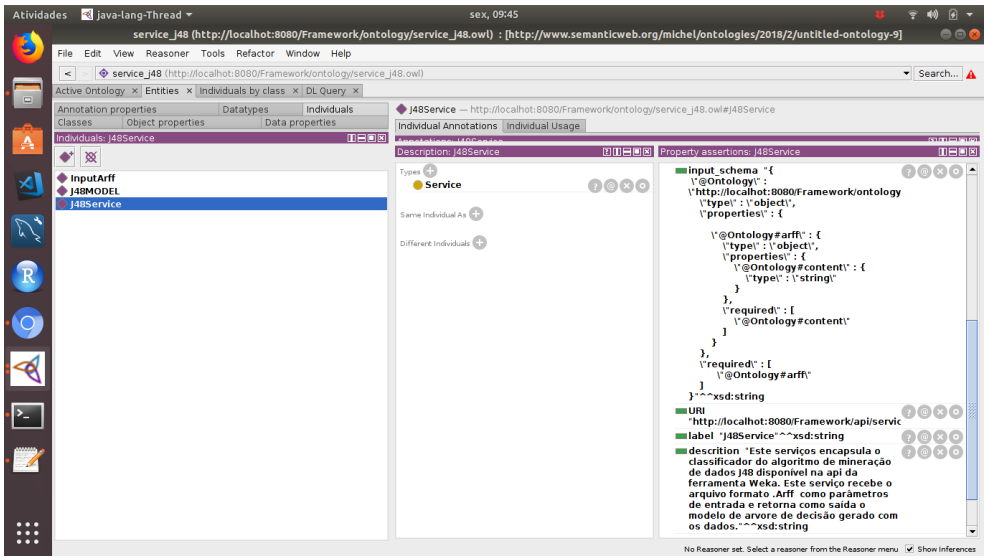

Figura A.39: Finalizando o Serviços Web Semântico J48Service.

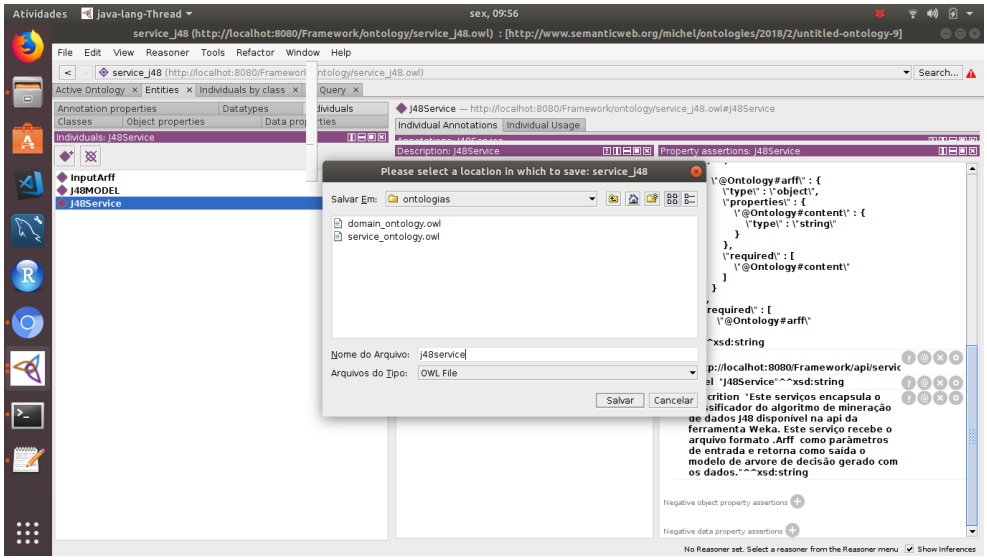

Figura A.40: Salvar o serviço.