UNIVERSIDADE DE SÃO PAULO
FACULDADE DE FILOSOFIA, CIÊNCIAS E LETRAS DE RIBEIRÃO PRETO
DEPARTAMENTO DE COMPUTAÇÃO E MATEMÁTICA

MATHEUS DE LARA CALACHE

Suporte à anotação gráfica e colaborativa de serviços web semânticos

Ribeirão Preto–SP

2020

MATHEUS DE LARA CALACHE

Suporte à anotação gráfica e colaborativa de serviços web semânticos

Versão Corrigida

Versão original encontra-se na FFCLRP/USP.

Dissertação apresentada à Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP) da Universidade de São Paulo (USP), como parte das exigências para a obtenção do título de Mestre em Ciências.

Área de Concentração: Computação Aplicada.

Orientador: Dr. Cléver Ricardo Guareis de Farias

Ribeirão Preto–SP

2020

MATHEUS DE LARA CALACHE

Graphical and collaborative annotation support for semantic web services

Corrected Version

The original version is found at FFCLRP/USP.

Dissertation presented to Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP) from the Universidade de São Paulo (USP), as part of the requirements to hold the Master of Science degree.

Field of Study: Applied Computing.

Supervisor: Dr. Cléver Ricardo Guareis de Farias

Ribeirão Preto–SP

2020

Matheus de Lara Calache

Suporte à anotação gráfica e colaborativa de serviços web semânticos. Ribeirão Preto–SP, 2020.

123p. : il.; 30 cm.

Dissertação apresentada à Faculdade de Filosofia, Ciências e Letras
de Ribeirão Preto da USP, como parte das exigências para
a obtenção do título de Mestre em Ciências,
Área: Computação Aplicada.

Orientador: Dr. Cléver Ricardo Guareis de Farias

1. Arquitetura orientada a serviços; 2. Serviços web semânticos; 2. Padrão SAWSDL; 3. Notação visual; 4. Edição colaborativa; 5. Grasews.

Matheus de Lara Calache

Suporte à anotação gráfica e colaborativa de serviços web semânticos

Dissertação apresentada à Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP) da Universidade de São Paulo (USP), como parte das exigências para a obtenção do título de Mestre em Ciências.

Trabalho aprovado. Ribeirão Preto–SP, 3 de Abril de 2020:

Orientador
Dr. Cléver Ricardo Guareis de Farias

Professor
Dr. Ildeberto Aparecido Rodello

Professor
Dr. Joaquim Cezar Felipe

Professor
Dr. Wanderley Lopes de Souza

Ribeirão Preto–SP
2020

Este trabalho é dedicado à memória dos meus avós, Djalma e Marisa, à minha esposa, Marcela e, por fim, à comunidade científica do Brasil, para que tenha forças para sobreviver em tempos tão sombrios.

Agradecimentos

Ao meu orientador, Prof. Dr. Cléver Ricardo Guareis de Farias, por todo apoio, orientação, dedicação e ensinamentos durante os quatro anos de elaboração deste trabalho, inclusive durante suas férias e períodos de festividades.

À minha esposa, Marcela Scatolin Calache, por toda paciência, incentivo e apoio emocional durante os quatro anos de desenvolvimento deste trabalho. Se tem alguém que sabe o quão difíceis foram esses quatro anos, esse alguém é ela.

Aos meus avós Djalma e Marisa (*in memoriam*) por todo o apoio durante a minha vida, sem o qual eu não estaria onde eu estou.

À minha amiga Vânia Moreira, por tornar possível a conciliação dos horários de trabalho com as atividades relacionadas a esta pesquisa.

Aos meus colegas do LSSB e do DCM, por todo conhecimento compartilhado e apoio.

E aos demais professores do DCM, com os quais eu tive oportunidade de ter aulas que contribuíram para a fundamentação teórica deste trabalho.

Resumo

CALACHE, Matheus de Lara. **Suporte à anotação gráfica e colaborativa de serviços web semânticos**. 2020. 123p. Dissertação (Mestrado em Ciências) - Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Universidade de São Paulo, 2020.

Os serviços web têm se tornado um paradigma cada vez mais importante no desenvolvimento de software. De modo a facilitar a busca, a composição e o reuso de serviços web, descrições de serviços web podem ser anotadas semanticamente com definições providas por uma ontologia, criando os chamados serviços web semânticos. Um serviço web semântico é desenvolvido segundo diferentes abordagens e padrões recomendados pela W3C, tais como OWL-S, SAWSDL e WSMO-Lite. Dentre estes padrões, SAWSDL tem tido maior adoção devido à sua simplicidade. Diferentes ferramentas de suporte à anotação semântica segundo o padrão SAWSDL estão disponíveis no mercado. Porém, estas ferramentas possuem baixo nível de abstração, exigindo de seus usuários um extenso conhecimento técnico de WSDL/XML e de outras tecnologias relacionadas. A tarefa de anotação semântica poderia ser facilitada caso esta pudesse ser realizada por meio de notações visuais em um nível maior de abstração, de tal forma que as sintaxes de WSDL/XML e do próprio padrão SAWSDL pudessem ser omitidos e a anotação semântica pudesse ser realizada diretamente em elementos gráficos. A anotação semântica também poderia ser beneficiada se feita de forma colaborativa. Diferentes pessoas com diferentes especializações poderiam colaborar na criação de serviços web semânticos, independentemente de suas localizações geográficas. Neste sentido, o objetivo deste trabalho foi investigar o desenvolvimento colaborativo de serviços web semânticos por meio de notações visuais, segundo a abordagem SAWSDL. Adicionalmente, propusemos uma notação visual para a representação dos principais elementos de uma especificação WSDL com o propósito de prover anotação semântica segundo SAWSDL. Na sequência, desenvolvemos uma ferramenta de suporte gráfico e edição colaborativa à anotação semântica, denominada Grasews, facilitando o processo de anotação semântica e a disseminação de conhecimento acerca de serviços web semânticos. Finalmente, demonstramos a utilização da solução desenvolvida por meio de uma prova de conceito envolvendo a anotação semântica de um conjunto de serviços simples.

Palavras-chave: Arquitetura orientada a serviços; Serviços web semânticos; Padrão SAWSDL; Notação visual; Edição colaborativa; Grasews.

Abstract

CALACHE, Matheus de Lara. **Graphical and collaborative annotation support for semantic web services**. 2020. 123p. Dissertação (Mestrado em Ciências) - Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Universidade de São Paulo, 2020.

Web services have become increasingly important for software development. In order to facilitate the search, composition, and reuse of web services, their descriptions can be semantically annotated using definitions provided by an ontology, thus creating the so-called semantic web services. A semantic web service is developed according to different approaches and standards recommended by W3C, such as OWL-S, SAWSDL, and WSMO-Lite. Among these standards, SAWSDL has attracted interest due to its simplicity. A restricted number of tools are currently available to support the development of semantic annotations, according to SAWSDL. However, these tools support the annotation process at a low abstraction level, therefore requiring from users an extensive technical knowledge on WSDL/XML, among other technologies. The semantic annotation task could be facilitated if it were carried at a higher abstraction level using graphical notations, thus abstracting most of the sintact details of WSDL/XML and SAWSDL itself. Additionally, the semantic annotation could also benefit if it were carried collaboratively. Different people from different and complementary backgrounds could collaborate to create semantic web services, regardless of their geographic locations. In that sense, this work aimed at investigating the collaborative development of semantic web services supported by graphic notations, according to SAWSDL. We have proposed a visual notation to represent the main elements of a WSDL 2.0 especification focused on the semantic annotation using SAWSDL. Then, we have developed a graphical and collaborative supporting tool for semantic annotation, called Grasews, thus facilitating the semantic annotation process and the dissemination of knowledge on semantic web services. Finally, we have demonstrated the developed solution by a proof of concept involving the semantic annotation of a set of simple services.

Keywords: Service oriented architecture; Semantic web services; SAWSDL standard; Visual notation; Collaborative editing; Grasews.

Lista de abreviaturas e siglas

API	Application Programming Interface
ASP	Active Server Pages
CSCW	Computer-Supported Cooperative Work
CSS	Cascading Style Sheets
DBS	Desenvolvimento Baseado em Serviços
DDD	Domain-Driven Design
EAP	Estrutura Analítica de Projeto
EF	Entity Framework
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JS	JavaScript
JSON	JavaScript Object Notation
MVC	Model-View-Controller
ORM	Object-Relational Mapper
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
RDF	Resource Description Framework
REST	Representational State Transfer
SAWSDL	Semantic Annotations for WSDL and XML Schema
SGBD	Sistema Gerenciador de Banco de Dados
SOLID	Single Responsibility, Open–Closed, Liskov Substitution, Interface Segregation, Dependency Inversion
SOAP	Simple Object Access Protocol
TCSC	Trabalho Cooperativo Suportado por Computadores

TI	Tecnologia da Informação
UML	Unified Modeling Language
UDDI	Universal Description Discovery and Integration
UI	User Interface
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Description Language
WSDL-S	Web Service Semantics
WSMO-Lite	Web Service Modeling Ontology Lite
XHTML	EXtensible HyperText Markup Language
XML	Extensible Markup Language
XSD	XML Schema
XSLT	EXtensible Stylesheet Language Transformation

Lista de figuras

Lista de tabelas

Lista de listagens

Sumário

1

Introdução

Serviços web semânticos representam uma solução tecnológica para auxiliar na descoberta, no reuso e na composição de serviços web. Um serviço web semântico é criado por meio de anotações semânticas realizadas na descrição de um serviço web. Diferentes padrões de anotação semântica surgiram com o objetivo de prover suporte ao desenvolvimento de serviços web semânticos. O padrão SAWSDL consiste em uma solução simples para o desenvolvimento de serviços web semânticos. Contudo, o suporte ferramental para a anotação semântica é limitado. Atualmente, as ferramentas disponíveis exigem de seus usuários um extenso conhecimento de tecnologias e padrões envolvidos na anotação semântica. Adicionalmente, os formatos de distribuição destas ferramentas dificultam o suporte para a criação e a adoção de serviços web semânticos. Por fim, as atuais ferramentas disponíveis não proveem suporte à anotação semântica de forma colaborativa.

O objetivo deste trabalho é investigar o suporte ao desenvolvimento colaborativo de serviços web semânticos por meio de notações visuais, resultando na criação de uma notação visual para a representação de serviços web semânticos segundo o padrão SAWSDL juntamente com a criação de uma nova ferramenta de suporte gráfico e colaborativo para o desenvolvimento deste serviços.

O restante deste capítulo está estruturado da seguinte forma: a seção 1.1 apresenta a motivação para o desenvolvimento colaborativo de serviços web semânticos por meio de notações visuais; a seção 1.2 apresenta os objetivos do trabalho; a seção 1.3 apresenta a metodologia utilizada para desenvolvimento deste trabalho; e, por fim, a seção 1.4 apresenta a estrutura dos demais capítulos desta dissertação.

1.1 Motivação

Nos últimos anos, presenciamos o surgimento de novos modelos arquitetônicos para o desenvolvimento de aplicações na Web (PAPAZOGLOU; GEORGAKOPOULOS, 2003). Em

particular, podemos destacar o Desenvolvimento Baseado em Serviços (DBS) (IEEE, 2011; SOA Manifesto, 2013; IBM, 2019). De acordo com este modelo arquitetônico, aplicações podem ser desenvolvidas a partir da composição de um conjunto de serviços desenvolvidos por diferentes organizações. Serviços web representam uma solução tecnológica baseada em padrões abertos da Web amplamente utilizada em diferentes domínios do conhecimento para o desenvolvimento baseado em serviços (DUTTA; DEVI; ARORA, 2017; GUARDIA et al., 2017; KACI; NACEF; HENNI, 2018; OYUCU; POLAT, 2018; HUANG et al., 2019). Serviços web são descritos por meio de descrições de serviços, normalmente desenvolvidas usando o padrão *Web Service Description Language* (WSDL) (W3C, 2007c).

De modo a facilitar a descoberta, o reuso e a composição de serviços web, tanto por computadores quanto por seres humanos, descrições de serviços web podem ser anotadas semanticamente por meio de associações a conceitos de uma ontologia. Uma ontologia é um artefato computacional criado para representar de forma explícita o conhecimento de um domínio por meio de definições de conceitos e relacionamentos entre os mesmos (GRUBER, 1993). Descrições de serviços web anotadas com conceitos de uma ontologia são chamadas de serviços web semânticos (CARDOSO; SHETH, 2006). Diferentes tecnologias e padrões web podem ser utilizados para anotar semanticamente descrições de serviços web, tais como: *Web Ontology Language for Services* (OWL-S) (W3C, 2004a), *Semantic Annotations for WSDL and XML Schema* (SAWSDL) (W3C, 2007b) e *Web Service Modeling Ontology Lite* (WSMO-Lite) (W3C, 2010). Dentre estes padrões, SAWSDL destaca-se em razão de sua simplicidade.

O processo de anotação semântica deve ter suporte ferramental adequado. Contudo, a disponibilidade de ferramentas de suporte à anotação semântica é ainda limitada. A baixa disponibilidade de suporte ferramental para a anotação semântica de serviços web dificulta uma maior adoção da tecnologia de serviços web semânticos. Atualmente, exemplos de ferramentas para suporte à anotação semântica incluem: Radiant (BELHAJJAME; EMBURY, 2014), Iridescent (STAVROPOULOS; VRAKAS; VLAHAVAS, 2013), EasyWSDL (EASYSAWSDL, 2016) e OWL-S Editor (DENKER; ELENIUS; MARTIN, 2016; SCICLUNA, 2016).

Notações visuais são uma das formas mais antigas e efetivas de representação do conhecimento (DAVIES, 1990; MATHEWS, 1991; RAO, 2005). Diferentes tipos de recursos visuais, tais como formas geométricas, cores, ícones, texturas e brilhos, podem ser utilizados para criar modelos em um dado domínio (SMITH et al., 2004; MOODY, 2009). Adicionalmente, notações visuais são mais eficazes para a comunicação e a transmissão de informações do que outras formas de comunicação, incluindo a comunicação verbal e textual (MOODY, 2009). Tal característica advém da melhor capacidade do cérebro humano em processar (paralelamente) representações visuais (espaciais).

Atualmente, notações visuais têm sido amplamente utilizadas em diversas áreas

de conhecimento. Por exemplo, *Unified Modeling Language* (UML) (OMG, 2017) tem sido utilizada na representação de artefatos de *software*; *Business Process Management Notation* (BMPN) (OMG, 2011) tem sido utilizada na representação de processos de negócio; *Systems Biology Graphical Notation* (SBGN) (NOVèRE et al., 2009; TOURé et al., 2018) tem sido utilizada na representação de modelos biológicos *in silico*; enquanto que *Synthetic Biology Open Language* (SBOL) (QUINN et al., 2015) tem sido utilizada na representação de modelos de engenharia genética.

As ferramentas disponíveis de suporte ao desenvolvimento de serviços web semânticos possuem baixo nível de abstração. As abordagens de anotação semântica implementadas pelas atuais ferramentas pressupõem que as anotações devam ser inseridas diretamente nas descrições de serviços WSDL/XML, exigindo de seus usuários um extenso conhecimento técnico. Tanto os elementos que compõem uma descrição de serviço web quanto os conceitos e relacionamentos de uma ontologia poderiam ser mais facilmente compreendidos por meio de notações visuais ao invés das suas representações textuais.

Por fim, nem sempre a tarefa de anotar semanticamente um serviço web é realizada pela mesma pessoa (equipe) que desenvolveu o serviço. Esta tarefa pode ser realizada por um especialista de domínio, i.e., uma pessoa com amplo conhecimento da área de aplicação, mas não necessariamente com conhecimento técnico de serviços web e padrões relacionados. Neste sentido, o processo de anotação semântica deveria, de forma ideal, envolver a participação de diferentes atores, desde desenvolvedores de serviços até especialistas de domínio. Como resultado, o processo de anotação semântica seria facilitado e permitiria que usuários com diferentes especializações colaborassem com sua realização. Porém, atualmente, as ferramentas disponíveis para a anotação semântica não oferecem recursos que deem suporte ao trabalho colaborativo.

1.2 Objetivo

Este trabalho tem por objetivo principal investigar o suporte ao desenvolvimento colaborativo de serviços web semânticos por meio de notações visuais, segundo a abordagem *Semantic Annotations for WSDL and XML Schema* (SAWSDL) (W3C, 2007b). Neste sentido, este trabalho possui os seguintes objetivos específicos: i) propor uma notação visual para representar descrições WSDL de serviços web e anotações semânticas associadas ao padrão SAWSDL; ii) prover suporte à anotação gráfica e colaborativa por meio do desenvolvimento de uma aplicação web.

A utilização de tecnologias web na implementação de uma nova ferramenta facilitará o acesso a um suporte ferramental adequado. O suporte à anotação semântica por meio de notações visuais facilitará a criação de serviços web semânticos, visto que detalhes técnicos

serão abstraídos na anotação semântica. Finalmente, o suporte à edição colaborativa permitirá que usuários com diferentes especializações possam anotar semanticamente serviços web de forma colaborativa, simultânea e remota. Com o desenvolvimento de uma ferramenta web de suporte à anotação semântica, de mais fácil distribuição e com suporte ao trabalho colaborativo, acreditamos que o desenvolvimento e a utilização de serviços web semânticos, em geral, e serviços SAWSDL, em particular, sejam facilitados.

1.3 Metodologia

Inicialmente, realizamos uma revisão bibliográfica sobre serviços web, ontologias, abordagens de anotações semânticas para serviços web, notações visuais e sistemas de edição colaborativa. Adicionalmente, realizamos uma pesquisa sobre as principais ferramentas de anotação semântica propostas na literatura.

Após completar os estudos de fundamentação teórica do trabalho, realizamos uma pesquisa de diferentes *frameworks* e bibliotecas para o suporte ao desenvolvimento da nossa ferramenta de anotação colaborativa. Adicionalmente, pesquisamos diferentes abordagens tecnológicas para permitir a anotação semântica de forma colaborativa. Estas atividades tiveram por objetivo construir uma base sólida de conhecimento para compreender os desafios das diferentes abordagens para anotação semântica de um serviço web, bem como para planejar e projetar a criação da nova ferramenta.

Após os estudos que embasaram o desenvolvimento deste trabalho, propusemos uma notação visual para representar os diferentes tipos de elementos que compõem uma descrição de serviço web WSDL 2.0, bem como uma notação visual para representar hierarquicamente conceitos de uma ontologia e os diferentes tipos de anotações semânticas definidos em SAWSDL.

Em seguida, levantamos os requisitos funcionais e não-funcionais da ferramenta e criamos modelos da interface de usuário (UI - *User Interface*), também chamados de *mockups*, para suas principais funcionalidades. A fase de análise dos requisitos também possibilitou a total compreensão do escopo de desenvolvimento da nova ferramenta, bem como a criação da estrutura analítica de projeto (EAP) (PMI, 2017). A EAP auxiliou o desenvolvimento do projeto pois forneceu uma decomposição hierárquica orientada à entrega do trabalho a ser executado durante o projeto.

Após a fase de análise dos requisitos, iniciamos o desenvolvimento da nova ferramenta de suporte à anotação semântica. A ferramenta foi desenvolvida utilizando a linguagem de programação C#, do *Microsoft .NET Framework*, bem como um conjunto de padrões e práticas de desenvolvimento que facilitam a estruturação e o reuso do código, tais como padrões de projeto (*design-patterns*); a abordagem de desenvolvimento

Domain-Driven Design (DDD) (EVANS; FOWLER, 2004); entre outros.

Finalmente, de modo a validar o funcionamento da nova ferramenta, desenvolvemos uma prova de conceito dividida em duas etapas. A primeira etapa consistiu da anotação semântica de um serviço no domínio de análise de genômica funcional. Este processo foi utilizado para demonstrar o uso da ferramenta a fim de obter resultados similares aos resultados obtidos por meio da utilização de outra ferramenta (metodologia) originalmente. Já a segunda etapa consistiu da anotação semântica de um serviço no domínio cinematográfico. Este processo foi então utilizado para demonstrar o uso da ferramenta para o trabalho colaborativo de anotação semântica.

1.4 Estrutura do documento

O restante do documento está estruturado da seguinte forma: o capítulo 2 apresenta a fundamentação teórica necessária à compreensão dos principais conceitos utilizados neste trabalho; o capítulo 3 apresenta nossa proposta de notação visual para a representação de descrições de serviços, ontologias e anotações semânticas segundo a abordagem SAWSDL; o capítulo 4 apresenta o Grasews, uma nova ferramenta para suporte à anotação semântica de serviços web; o capítulo 5 apresenta uma prova de conceito realizada com ferramenta Grasews de modo a validar o seu funcionamento; e, por fim, o capítulo 6 apresenta a conclusão do trabalho e elenca direções futuras para esta pesquisa.

2

Fundamentação Teórica

Este capítulo apresenta uma visão geral acerca de cinco conjuntos fundamentais de conceitos necessários à compreensão do contexto e das soluções tecnológicas para o desenvolvimento deste trabalho. O capítulo está estruturado da seguinte forma: a seção 2.1 apresenta uma visão geral sobre Desenvolvimento Baseado em Serviços; a seção 2.2 apresenta uma visão geral sobre representação de conhecimento, web semântica e serviços web semânticos; a seção 2.4 apresenta a pesquisa realizada sobre ferramentas disponíveis para suporte à anotação semântica; a seção 2.3 apresenta uma visão geral sobre sistemas de edição colaborativos; a seção 2.5 apresenta uma visão geral sobre notações visuais; e por fim, a seção 2.6 apresenta uma visão geral sobre a abordagem de desenvolvimento *Domain-Driven Design*.

2.1 Desenvolvimento Baseado em Serviços

Esta seção apresenta uma visão geral dos fundamentos de Desenvolvimento Baseado em Serviços, bem como introduz os principais padrões e tecnologias para o desenvolvimento de serviços web.

2.1.1 Visão Geral

Os sistemas computacionais possuem um papel cada vez mais relevante no suporte às atividades (de negócio) de uma organização. O desenvolvimento de um sistema computacional é uma tarefa não trivial. Além da identificação dos requisitos funcionais de um sistema, cada vez mais as equipes de desenvolvimento se veem confrontadas com a necessidade de suporte a um conjunto cada vez maior de requisitos não funcionais, tais como segurança, escalabilidade, disponibilidade e interoperabilidade. Adicionalmente, as

necessidades do acesso ao sistema por usuários geograficamente distribuídos contribuem para aumentar a complexidade destes sistemas.

A arquitetura de um sistema computacional descreve as propriedades fundamentais de um sistema em seu ambiente de execução em termos de seus elementos (componentes), relacionamentos e princípios que norteiam o desenvolvimento e a evolução deste sistema (IEEE, 2011). A descrição da arquitetura de um sistema é utilizada como base para seu projeto, seu desenvolvimento e sua manutenção.

O Desenvolvimento Baseado em Serviços (DBS), também chamado de Arquitetura Orientada a Serviços (*Service Oriented Architecture* – SOA) (PAPAZOGLOU; GEORGAKOPOULOS, 2003; VALIPOUR et al., 2009), é um paradigma de desenvolvimento de software que facilita o projeto, o desenvolvimento, a manutenção e a evolução de um sistema computacional. De acordo com esse paradigma, serviços representam os elementos fundamentais no desenvolvimento de uma nova aplicação.

Um serviço consiste em uma entidade de software (componente) aberta, com interface bem definida e cujas funcionalidades podem ser acessadas (remotamente) por entidades clientes. Uma descrição de serviço é utilizada para definir as funcionalidades e o comportamento esperado de um serviço, bem como os detalhes técnicos necessários à localização do serviço e à invocação de suas funcionalidades.

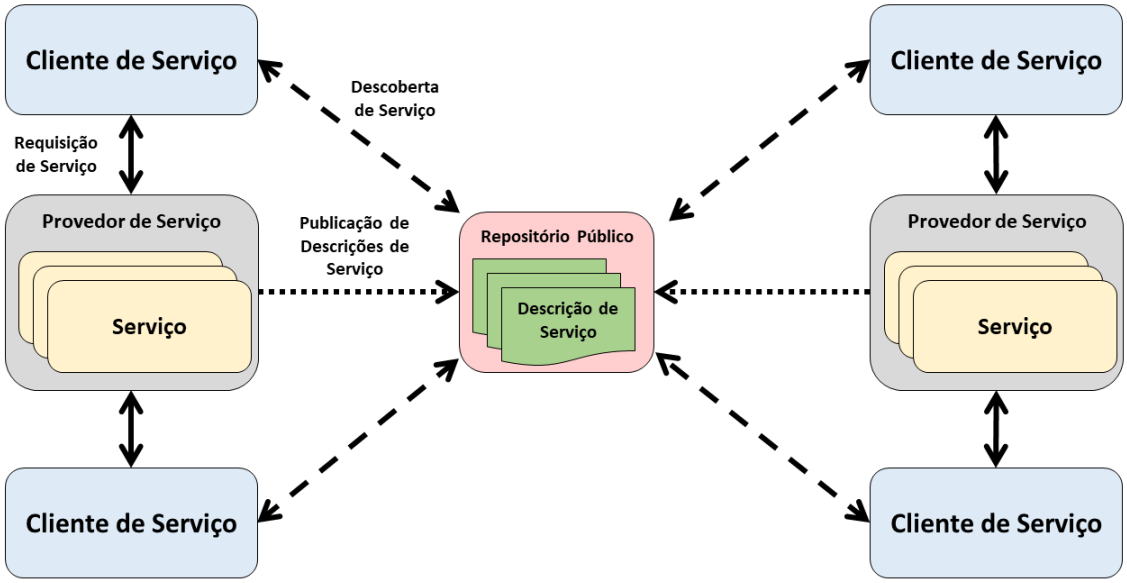
Serviços são oferecidos por provedores de serviços e consumidos por clientes de serviços. Um provedor de serviço consiste em uma organização responsável por manter a infraestrutura técnica necessária à execução do serviço, bem como por disponibilizar a descrição do serviço. Um cliente de serviço representa uma entidade que utiliza as funcionalidades de um serviço para atingir seus objetivos. Clientes podem realizar buscas em repositórios públicos contendo descrições de serviços para identificar serviços de interesse. Um serviço pode ser utilizado de forma isolada ou combinada com outros serviços para o rápido desenvolvimento de uma nova aplicação, criando as chamadas composições de serviço.

A Figura 1 ilustra os principais elementos de uma arquitetura orientada a serviços. A seta pontilhada representa a publicação de uma descrição de serviço em um repositório público da web, feita por um provedor de serviços. A seta tracejada representa a descoberta de um serviço web, por meio de descrições existentes em um dado repositório. Finalmente, a seta sólida representa uma requisição de um serviço.

Um dos principais objetivos do DBS é disponibilizar as funcionalidades implementadas por diferentes aplicações na forma de serviços, por meio do encapsulamento, da organização e da orquestração dessas funcionalidades. Neste sentido, as diferentes funcionalidades providas por um sistema (existente) podem ser utilizadas como base para o desenvolvimento de diferentes serviços. Um mesmo serviço também pode ser reutilizado

no desenvolvimento de novas aplicações. Adicionalmente, o DBS provê suporte à interoperabilidade, por meio do uso de descrições de serviço padronizadas e independentes de linguagens de programação e do uso de protocolos abertos para permitir a interação entre as entidades clientes e os serviços propriamente ditos.

Figura 1 – Arquitetura orientada a serviços.



Fonte: Autoria própria.

2.1.2 Serviços Web

Serviços web representam uma solução tecnológica concreta para o desenvolvimento baseado em serviços. Um serviço web utiliza um conjunto de padrões abertos da web para o seu desenvolvimento e invocação (utilização), sendo identificado univocamente por um *Uniform Resource Identifier* (URI). Serviços web são atualmente desenvolvidos de acordo com duas principais abordagens: serviços SOAP e serviços RESTful.

2.1.2.1 Serviços SOAP

Serviços web SOAP são implementados utilizando um conjunto de tecnologias e padrões abertos de desenvolvimento providos pela W3C (DACONTA; OBRST; SMITH, 2003). Os três principais padrões usados no desenvolvimento de serviços web SOAP são o protocolo *Simple Object Access Protocol* (SOAP) (W3C, 2007a), a linguagem de descrição de serviços *Web Service Description Language* (WSDL) (W3C, 2007c) e o registro de serviços *Universal Description Discovery and Integration* (UDDI) (OASIS, 2004).

Interações entre um serviço web SOAP e um cliente de serviço ocorrem com a utilização do protocolo SOAP (W3C, 2007a). O protocolo SOAP serializa os dados (objetos) envolvidos em uma interação usando XML. A transferência desses objetos, contidos nas mensagens de requisições, por parte de um cliente, e de respostas, por parte de um provedor, é normalmente realizada por meio do protocolo *Hypertext Transfer Protocol* (HTTP).

A informação contida em uma mensagem criada a partir dos objetos serializados é armazenada no elemento *envelope*, o qual é formado por dois outros elementos: *header* e *body*. O elemento *envelope* define o documento XML como uma mensagem SOAP. O elemento *header* contém informações específicas da mensagem SOAP, como, por exemplo, o endereço de IP de origem. Apesar de ser opcional, o elemento *header*, se presente, deve ser o primeiro elemento-filho de envelope. O elemento *body* armazena as informações referentes à requisição, como o nome dos métodos e o objeto serializado que será enviado na comunicação.

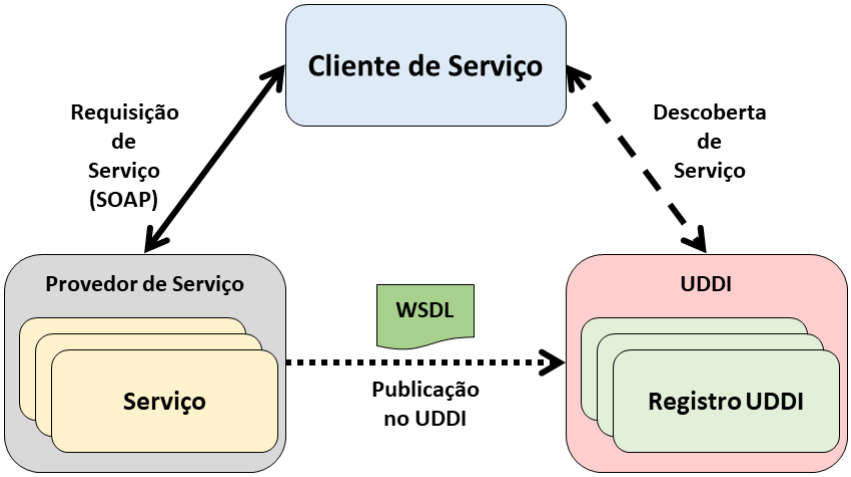
A linguagem WSDL é utilizada para descrever as funcionalidades e a localização de um serviço web SOAP (W3C, 2007c). O desenvolvimento de uma especificação WSDL possibilita que um serviço web seja consumido por outras aplicações por meio da disponibilização da descrição dos métodos do serviço, dos seus parâmetros e dos seus valores de retorno. Informações adicionais sobre a linguagem WSDL podem ser encontradas na seção 2.1.2.3 deste documento.

UDDI consiste de um padrão para o registro (publicação) e a busca (descoberta) de descrições de serviços web SOAP em um repositório da Internet (PAPAZOGLOU; GEORGAKOPOULOS, 2003; OASIS, 2004). A partir do registro UDDI de um serviço web, realizado por um provedor de serviços, um cliente de serviços pode pesquisar e descobrir este serviço web. Além de contribuir com a busca de serviços web a partir de suas descrições, o UDDI provê informações necessárias para que um serviço possa ser reutilizado por um cliente (OASIS, 2004), tais como informações sobre a organização que desenvolveu o serviço web ou sobre as regras de negócio associadas.

Um registro UDDI é criado com base em alguns elementos de uma especificação WSDL. UDDI contém todos os metadados de um serviço web, incluindo uma referência para a descrição WSDL de um serviço e um conjunto de definições que habilita a invocação deste serviço. Um registro UDDI fornece uma interface única (padronizada) que facilita a classificação, a localização, a invocação e o gerenciamento de metadados de um serviço.

A Figura 2 ilustra a utilização dos diferentes padrões no desenvolvimento e invocação de um serviço web SOAP. A seta pontilhada representa a publicação de uma descrição de serviço WSDL em um repositório UDDI. A seta tracejada representa a descoberta de um serviço web junto a um repositório UDDI. Finalmente, a seta sólida representa a requisição de serviço SOAP realizada por um cliente de serviço.

Figura 2 – Padrões abertos utilizados em um serviço web SOAP.



Fonte: Autoria própria.

2.1.2.2 Serviços RESTful

REST (*Representational State Transfer*) foi proposto como um modelo arquitetônico para guiar o projeto de desenvolvimento de sistemas computacionais distribuídos cujas interações entre as diferentes partes deste sistema são realizadas por meio do protocolo HTTP (FIELDING, 2000; CARDOSO; SHETH, 2006). REST é definido a partir de um conjunto de seis restrições arquitetônicas:

1. *Arquitetura Cliente-Servidor*, a qual separa a arquitetura de uma aplicação e as responsabilidades em duas partes, fazendo com que o cliente e o provedor não se preocupem com as atividades um do outro;

2. *Interações sem estado*, as quais definem que toda requisição contenha todas as informações necessárias para que o provedor consiga entendê-la e processá-la, sem a necessidade de informações adicionais ou providas por terceiros;

3. *Informações em cache*, as quais permitem que recursos muito requisitados por clientes possam ser armazenados, temporariamente, em cache, evitando processamento desnecessário e aumentando significativamente o desempenho;

4. *Interface uniforme*, a qual define um contrato para a comunicação entre clientes e provedores de serviços. Este contrato apresenta regras usadas para tornar um componente genérico, facilitando sua manutenção e expansão;

5. *Sistema em camadas*, o qual determina que o sistema seja estruturado em camadas hierárquicas, sendo que cada camada possui uma função específica. Essa estrutura

possibilita que responsabilidades possam ser distribuídas entre as camadas, de modo que cada componente do sistema tenha papel e responsabilidades bem definidos;

6. *Código sob demanda*, o qual, apesar de ser opcional, permite que o cliente possa estender a parte lógica do provedor por meio de *applets* e *scripts*. Isso permite que diferentes clientes possam se comportar de maneiras diferentes, conforme suas necessidades, mesmo que consumam os mesmos serviços disponíveis no provedor.

Os serviços criados com base no modelo arquitetônico REST são chamados de serviços RESTful. Estes serviços não são vinculados a uma tecnologia em particular, ou seja, podem ser desenvolvidos em qualquer plataforma ou linguagem de programação. Os serviços RESTful são mais simples de serem desenvolvidos do que os serviços SOAP, pois não impõem restrições quanto ao formato das mensagens trocadas entre as entidades envolvidas em uma interação. Tal característica permite que o desenvolvedor possa optar por um formato mais adequado para as mensagens do sistema de acordo com sua necessidade. Os formatos de mensagens mais comuns são JSON, XML e texto puro.

2.1.2.3 Descrição de Serviços Web

A descrição de um serviço web é composta por um conjunto de informações que definem como este serviço pode ser acessado. Tais informações incluem as operações disponíveis, os parâmetros e os valores de retorno, bem como a localização (endereço) do serviço.

Inicialmente, a linguagem *Web Service Description Language* (WSDL) focou na especificação de serviços SOAP (até a versão 1.1 desta linguagem (W3C, 2001)). Contudo, WSDL 1.1 não suportava a descrição de um serviço RESTful, pois não provia recursos suficientes para a descrição de todos os métodos HTTP possíveis de serem usados em um serviço RESTful. Assim, a versão 2.0 de WSDL (W3C, 2007c) foi criada a fim de resolver tais restrições.

Outras abordagens, tais como *Web Application Description Language* (WADL) (W3C, 2009) e *OpenAPI* (SMARTBEAR, 2017), podem ser utilizadas para a descrição de serviços RESTful. Porém, estas abordagens não são o foco deste trabalho.

A linguagem WSDL descreve um serviço web em duas partes fundamentais: parte abstrata e parte concreta. Dentro de cada parte, a descrição WSDL utiliza diferentes elementos que possibilitam a reutilização da descrição e também a separação de modelos independentes. A parte abstrata de uma descrição WSDL contém os elementos *types*, *operation* e *interface*, enquanto que a parte concreta contém os elementos *binding*, *service* e *endpoint*.

O elemento *types* (*<wsdl:types>*) representa os tipos de dados usados na comunicação. O elemento *operation* (*<wsdl:operation>*) representa uma descrição abstrata de

uma ação (operação) suportada por um serviço, ou seja, representa uma funcionalidade suportada pelo serviço web. Adicionalmente, o elemento relaciona parâmetros de entrada (*<wsdl:input>*) e de saída (*<wsdl:output>*).

Operações (*<wsdl:operation>*) são agrupadas por meio do elemento *interface* (*<wsdl:interface>*). Tal agrupamento tem por objetivo obter um formato padronizado para o transporte das mensagens. O elemento *binding* (*<wsdl:binding>*) especifica os detalhes do formato do transporte para uma ou mais interfaces. O elemento *endpoint* (*<wsdl:endpoint>*) associa um endereço de rede a um *binding*. Finalmente, o elemento *service* (*<wsdl:service>*) agrupa os elementos *endpoint*, a fim de implementar uma interface (*<wsdl:interface>*) comum.

Falhas, também chamadas de exceções, consistem de comportamentos não esperados e não tratados no fluxo normal de execução de uma operação de um serviço. A ocorrência de falhas é representada por um elemento do tipo *fault* (*<wsdl:fault>*). O elemento *fault* é especificado diretamente dentro do elemento *interface*, no mesmo nível dos elementos *operation*. Desta forma, uma mensagem de falha pode ser reutilizada por diferente operações. Os elementos *infault* (*<wsdl:infault>*) e *outfault* (*<wsdl:outfault>*) representam instâncias de falhas de entrada e saída, respectivamente, que podem ocorrer dentro de cada operação. Na ocorrência de uma falha, se esta estiver tratada pelo serviço, a mensagem de resposta (*<wsdl:output>*) de uma operação é substituída pela mensagem da falha.

2.2 Serviços Web Semânticos

Esta seção apresenta uma visão geral acerca da representação de conhecimento por meio de ontologias OWL, introduz a definição de serviços web semânticos e, por fim, apresenta três diferentes abordagens para o desenvolvimento de serviços web semânticos.

2.2.1 Representação do Conhecimento

O conhecimento de um dado domínio pode ser formalmente representado por meio de uma conceitualização (GRUBER, 1993; GRUBER, 1995). Uma conceitualização consiste em definições de entidades (objetos) que presume-se existir em uma determinada área de conhecimento (interesse), incluindo as relações existentes entre estes objetos. Assim, uma conceitualização representa uma visão simplificada e abstrata de um domínio de conhecimento. Toda base de conhecimento, todo sistema baseado em conhecimento ou agente baseado em conhecimento estão relacionados à alguma conceitualização, de forma explícita ou implícita.

Uma ontologia representa uma especificação explícita de uma conceitualização. Na

Filosofia, uma ontologia representa um relato sistemático da existência de objetos de um dado domínio. Quando o conhecimento de um domínio é representado por meio de um formalismo declarativo, o conjunto formado pelos conceitos que podem ser representados é chamado de universo de discurso (GRUBER, 1993).

Na área da computação, uma ontologia é um artefato criado para representar de forma explícita o conhecimento de um domínio por meio de definições de conceitos e relacionamentos entre os mesmos. Uma ontologia é geralmente desenvolvida por especialistas do domínio de interesse. Atividades como análises conceituais e modelagem de domínios com base em metodologias-padrões têm sido elaboradas ao longo dos anos com o objetivo de aprimorar e padronizar o desenvolvimento de novas ontologias (GUARINO, 1998). O estudo de ontologias tem sua importância reconhecida nas áreas de Inteligência Artificial, Linguística Computacional, Teoria de Banco de Dados, entre outras.

Atualmente, um volume crescente de dados tem sido disponibilizado na Web. A Web Semântica tem por objetivo associar dados (documentos) disponíveis na Web a significados bem definidos (BERNERS-LEE; HENDLER; LASSILA, 2001). Esta associação facilita a recuperação de informações, a integração e o reuso destes dados, tanto por máquinas quanto por seres humanos (CARDOSO; SHETH, 2006). Para criar esta realidade, anotações semânticas devem ser adicionadas aos dados contidos em documentos HTML e XHTML da Web. As anotações semânticas são feitas a partir de referências a conceitos definidos em uma ontologia.

Dentre as linguagens utilizadas para a representação de conhecimento, destaca-se a linguagem *Web Ontology Language* (OWL), atualmente na versão 2.0 (W3C, 2012a; W3C, 2012b). OWL é uma linguagem para a construção de ontologias em diferentes domínios de conhecimento. O conhecimento em uma ontologia OWL é representado por meio da especificação de classes, propriedades, instâncias e valores. Uma ontologia OWL pode ser representada usando-se diferentes sintaxes, tais como *Functional-Style*, RDF/XML, *Manchester*, *Turtle* e OWL/XML. A linguagem OWL faz parte de um conjunto de recursos da Web Semântica, que incluem RDF (W3C, 2014), RDFS (W3C, 2004b), SPARQL (W3C, 2008), entre outros.

2.2.2 Definição de Serviços Web Semânticos

O Desenvolvimento Baseado em Serviços pressupõe o reuso e a integração de serviços (composição de serviços). Porém, como identificar se um serviço atende às necessidades (funcionais) de um dado cliente e como identificar e resolver eventuais problemas de incompatibilidade semântica de modo a obter a interoperabilidade desejada entre diferentes serviços? Para responder a tais questionamentos, são necessárias informações precisas sobre as funcionalidades de um determinado serviço, bem como sobre os dados de entrada

e de saída deste serviço.

Serviços web semânticos (SWS) representam a extensão de serviços web com a aplicação dos princípios da Web Semântica. SWS representam serviços web cujas interfaces são semanticamente anotadas com conceitos (termos) definidos em uma ontologia. Neste sentido, SWS são criados a partir da adição de definições semânticas a interfaces de serviços web, bem como a operações e a dados de entrada e saída destas operações (CARDOSO; SHETH, 2006; W3C, 2007b; STAVROPOULOS; VRAKAS; VLAHAVAS, 2013).

O desenvolvimento de SWS proporciona diferentes benefícios, dentre os quais destacam-se (SHETH, 2007): i) facilidade de reuso de serviços, dado que descrições semânticas colaboram para encontrar serviços mais relevantes; ii) interoperabilidade semântica, alcançada por meio da anotação e do mapeamento semântico realizado entre os dados trocados entre serviços web; iii) facilidade de configuração e composição, permitindo ligações dinâmicas entre diferentes serviços web; iv) e, por fim, nível mais elevado de automação no ciclo de vida de uma atividade desenvolvida por uma organização, ou seja, auxilia na configuração dinâmica (análises de descoberta e restrições) e execução (tratamento de exceções durante a execução) dos processos desenvolvidos por uma organização.

2.2.3 Abordagem SAWSDL

Semantic Annotations for WSDL and XML Schema (SAWSDL) é um formato padrão da W3C para anotações semânticas em documentos WSDL (W3C, 2007b). SAWSDL define mecanismos para permitir que anotações semânticas possam ser adicionadas em um documento WSDL a partir de conceitos contidos em uma ontologia. Assim, SAWSDL pode ser visto como um conector, associando descrições puramente sintáticas, representadas por serviços web e suas descrições, a conceitos semânticos, representados por ontologias (KOPECKý et al., 2007). A partir desta associação, aplicações computacionais podem interpretar (parcialmente ou completamente) esses conceitos, a fim de automatizar tarefas, tais como a descoberta, a seleção e a composição de serviços web.

SAWSDL é independente de linguagens específicas para a representação de ontologias, bastando que os conceitos semânticos possam ser identificados a partir de URIs. Normalmente, os *frameworks* de anotação semântica para serviços web utilizam OWL/RDF (*Resource Description Framework*) em conjunto com SAWSDL. SAWSDL contém dois tipos de atributos: *Model Reference* e *Schema Mapping*.

Um atributo *Model Reference* (*sawsdl:modelReference*) representa uma anotação de um elemento (entidade) de um documento WSDL com referências a um conjunto de conceitos definidos em uma ou mais ontologias (KOPECKý et al., 2007). O valor contido no atributo *Model Reference* é um conjunto de zero ou mais URIs, separados por espaços,

que identificam, cada qual um diferente conceito (W3C, 2007b). Dada a definição de uma interface WSDL, o atributo *Model Reference* pode ser utilizado para prover uma descrição semântica para a interface (*<wsdl:interface>*) como um todo e/ou para cada operação (*<wsdl:operation>*) definida nesta interface.

Embora os nomes dos elementos de entrada e de saída possam indicar o comportamento esperado de uma operação, tal objetivo nem sempre pode ser atingido dada a possível incerteza e ambiguidade intrínseca aos identificadores utilizados. Neste sentido, cada elemento de uma entrada, saída ou falha associado a cada operação definida pode ser individualmente anotado com o atributo modelReference para prover uma descrição semântica para este elemento. A anotação de uma falha é composta por uma referência a um conceito semântico que provê uma descrição geral (alto nível) da falha. A anotação da ocorrência de uma falha não descreve a mensagem da falha em si. Esta mensagem deve ser criada como um elemento XSD, do documento *XML Schema*, e, portanto, este elemento também pode ser anotado.

O atributo *Model Reference* também pode ser utilizado para anotar os elementos pertencentes ao documento *XML Schema* (XSD), como, por exemplo, *<xs:element>*, *<xs:attribute>*, *<xs:simpleType>* e *<xs:complexType>*. Os elementos *<xs:element>* e *<xs:attribute>* podem ser anotados com *Model Reference*, com o propósito de descrever um conceito definido em um modelo semântico. Ao anotar um elemento *<xs:simpleType>*, qualquer elemento ou atributo deste tipo simples também será associado ao conceito do modelo semântico referenciado.

Um elemento *<xs:complexType>* pode ser anotado segundo duas abordagens distintas: *Bottom Level Annotation* e *Top Level Annotation*. *Bottom Level Annotation* refere-se à anotação aplicada a um elemento, que compõe o tipo complexo, ou a um atributo. Deste modo, todos os elementos e atributos de um tipo complexo podem ser anotados, caracterizando, portanto, uma anotação mais granularizada e específica para os elementos-filhos. Já *Top Level Annotation* refere se à anotação do elemento complexo como um todo, caracterizando, portanto, uma anotação mais genérica, sem se especializar nos elementos e atributos contidos no compartimento-pai. Apesar da existência de duas abordagens, elas são independentes, podendo ser utilizadas simultaneamente em um elemento *<xs:complexType>* (*Top Level Annotation*) e seus elementos-filhos (*Bottom Level Annotation*).

Um atributo do tipo *Schema Mapping* é utilizado para tratar eventuais incompatibilidades existentes entre o modelo semântico (ontologia) e a estrutura de parâmetros de entrada (*input*) e saída (*output*) de um serviço web (WSDL). O atributo *Schema Mapping* permite que transformações (conversões) de tipos de dados contidos em um serviço web e em uma ontologia sejam realizadas por meio de métodos *Lifting* e *Lowering*.

O método *Lifting*, representado pelo atributo *sawsdl:liftingSchemaMapping*, permite

que dados presentes em um nível sintático (WSDL/XML) sejam mapeados para um nível semântico. Neste sentido, *Lifting* permite que dados no formato XML, produzidos por um serviço web, sejam transformados em instâncias de um modelo semântico, como, por exemplo, instâncias de classes OWL. O método *Lowering*, representado pelo atributo *sawsdl:loweringSchemaMapping*, permite que dados possam ser manipulados de forma ontológica, i.e., *Lowering* transforma os tipos de dados de um nível semântico em tipos de dados de um nível sintático (WSDL/XML). Neste sentido, *Lowering* permite que instâncias de classes OWL possam ser transformadas de volta em dados no formato XML.

Cabe ressaltar que o SAWSDL não especifica a forma como a transformação dos dados sintáticos para os semânticos (ou vice-versa) será realizada. Os atributos *sawsdl:liftingSchemaMapping* e *sawsdl:loweringSchemaMapping* apenas fazem referência a um URI contendo um conjunto de regras de transformação. Estas regras podem ser definidas usando, por exemplo, as linguagens *EXtensible Stylesheet Language Transformation* (XSLT) (W3C, 1999) e XQuery (W3C, 2017).

2.2.4 Abordagem OWL-S

Web Ontology Language for Services (OWL-S) (W3C, 2004a) é uma abordagem baseada em OWL para a anotação semântica de serviços web, padronizada pela W3C. OWL-S consiste em uma ontologia usada para a descrição de serviços web. Um serviço web no padrão OWL-S é representado por uma instância da classe *ServiceProfile*, suportado por uma instância da classe *ServiceModel* e descrito por uma instância da classe *ServiceGrounding*. As instâncias destas três classes, também chamadas sub-ontologias, proveem três tipos de conhecimento essenciais para a descrição de um serviço web.

2.2.4.1 ServiceModel

ServiceModel define de forma abstrata como o serviço funciona. A partir da especificação de *ServiceModel*, as atividades de invocação, composição, monitoramento e recuperação de um serviço são passíveis de serem realizadas.

A dinâmica de interação (troca de mensagens) entre um cliente e um serviço web pode ser representada por meio de um processo. Um processo pode ser classificado como: atômico, composto ou simples. Um processo atômico representa um serviço que espera uma mensagem de entrada (*input*), realiza algum processamento e produz uma mensagem de saída (*output*). Um processo atômico não possui interações com outros processos (serviços). Um processo complexo, também chamado de processo composto, representa um serviço composto por outros serviços (subprocessos). A interação entre estes subprocessos é necessária para que o processo possa ser executado corretamente. Neste caso, existem

múltiplas mensagens de entrada e/ou de saída. Em um processo composto, é necessário orquestrar todas as mensagens, a fim de criar um fluxo de processamento e, então, obter o resultado esperado. Um processo complexo pode requerer múltiplos protocolos e múltiplas ações em um servidor. Finalmente, um processo simples representa um processo de forma abstrata, quer seja um processo atômico ou complexo.

Na especificação de *ServiceModel*, um processo é representado por meio de instâncias da classe *Process*, independentemente da sua classificação. *Process* descreve como utilizar o serviço a partir de detalhes do conteúdo semântico das requisições, das condições sob as quais os resultados específicos ocorrerão (pré-condições) e dos passos, se necessários, que levarão a estes resultados. *Process* também descreve como o serviço deve ser requisitado bem como quais resultados poderão ser obtidos por meio da execução do serviço (efeitos), assemelhando-se aos efeitos colaterais ocorridos no ambiente de execução do serviço. A classe *Process* está associada a um conjunto de entradas (*Inputs*), saídas (*Outputs*), pré-condições (*Preconditions*) e efeitos (*Effects*), também chamados de IOPEs (WALI; GIBAUD, 2012).

2.2.4.2 ServiceProfile

ServiceProfile define de forma abstrata o que um serviço faz. *ServiceProfile* inclui uma descrição da funcionalidade oferecida pelo serviço, das limitações quanto ao uso do serviço, da qualidade do serviço oferecido e, por fim, dos requisitos necessários para que um cliente possa utilizar este serviço com sucesso.

A fim de descrever as funcionalidades de um serviço web, OWL-S propõe uma especificação de um `ServiceProfile` por meio de uma instância da classe `Profile`. A classe `Profile` está associada a um conjunto de propriedades funcionais, viz., `hasParameter`, `hasInput`, `hasOutput`, `hasPrecondition` e `hasResult`. Todas as propriedades funcionais representam definições contidas no `ServiceModel`. A propriedade `hasParameter` representa uma instância de um parâmetro, seja de entrada (*input*) ou de saída (*output*). A propriedade `hasInput` representa uma instância de um parâmetro de entrada (*input*), enquanto a propriedade `hasOutput` representa uma instância de um parâmetro de saída (*output*). A propriedade `hasPrecondition` especifica uma pré-condição para a execução do serviço. Por fim, a propriedade funcional `hasResult` especifica um resultado ocasionado pela execução do serviço.

Além das propriedades funcionais, a especificação de `ServiceProfile` também é composta por seis propriedades não funcionais: `serviceParameter`, `serviceCategory`, `serviceName`, `textDescription`, `serviceClassification` e `serviceProduct`. A propriedade `serviceParameter` representa uma lista de propriedades adicionais que devem acompanhar a descrição do serviço. Cada elemento desta lista representa uma instância da

classe `ServiceParameter`. A classe `ServiceParameter` pode ser usada para representar atributos adicionais que incluem a qualidade provida pelo serviço, a classificação possível de ser dada ao serviço ou quaisquer outras informações relevantes para a especificação do serviço. Uma instância da classe `ServiceParameter` é composta pelas propriedades `serviceParameterName`, que representa o nome do parâmetro, e `sParameter`, que representa o valor do parâmetro.

A propriedade `serviceCategory` faz referência a um valor contido em alguma ontologia ou taxonomia de serviços. Tal valor representa uma instância da classe `ServiceCategory`, a qual descreve categorias dos serviços baseadas em uma classificação. Uma instância da classe `ServiceCategory` é composta pelas propriedades `categoryName`, que representa o nome da categoria em si, e `Taxonomy`, que representa um esquema taxonômico.

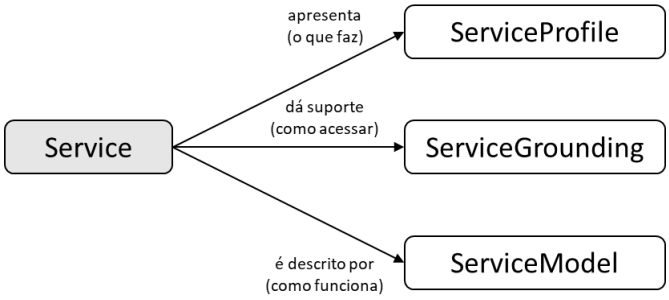
As propriedades `serviceName` e `textDescription` proveem descrições textuais do serviço. A primeira representa o nome do serviço que está sendo oferecido e pode ser utilizada como um identificador do serviço, enquanto a segunda representa uma breve descrição textual do serviço, incluindo um resumo do serviço, condições de uso e quaisquer outras informações que o provedor do serviço queira compartilhar com seus clientes. A propriedade `serviceClassification` define um mapeamento de `ServiceProfile` para uma ontologia de serviços OWL. Finalmente, a propriedade `serviceProduct` representa um mapeamento de `ServiceProfile` para uma ontologia de produtos OWL. Cada instância de `ServiceProfile` pode ser considerada um sumário de aspectos relevantes das funcionalidades oferecidas pelo serviço.

2.2.4.3 ServiceGrounding

`ServiceGrounding` define concretamente como acessar o serviço. `ServiceGrounding` descreve detalhes, tais como o formato das mensagens, os protocolos de comunicação e o endereço de um serviço web. `ServiceGrounding` realiza o mapeamento de cada operação e mensagem contidas na descrição de um serviço web (WSDL) para um processo contido na ontologia OWL-S. Adicionalmente, `ServiceGrounding` deve especificar os identificadores das operações e dos parâmetros de entrada e saída de um serviço web de forma não ambígua. Finalmente, `ServiceGrounding` deve especificar os dados de forma que eles sejam compatíveis uns com os outros, possibilitando a troca de mensagens, conforme contida em `ServiceModel`.

A Figura 3 resume as três classes de OWL-S e seus objetivos: `ServiceProfile` para representar o serviço OWL-S, `ServiceModel` para suportar o serviço OWL-S e, finalmente, `ServiceGrounding` para descrever o serviço OWL-S.

Figura 3 – Classes do padrão OWL-S.



Fonte: Adaptado de (W3C, 2004a)

2.2.5 Abordagem WSMO-Lite

Web Service Modeling Ontology (WSMO) (W3C, 2005a; W3C, 2010) também é um padrão da W3C para anotações semânticas em serviços web. Este padrão procura facilitar o descobrimento, a combinação e a utilização dos serviços disponíveis na Web de forma automática. Os objetivos do padrão WSMO são introduzir novas linguagens fundadas em formalismos expressivos e promover a modelagem *top-down*, ou seja, a semântica deve ser mantida como base para o desenvolvimento de um serviço web.

A estrutura do padrão WSMO é composta por quatro principais elementos: *ontologies*, que provê uma terminologia utilizada por outro elemento WSMO; *Web service descriptions*, que descreve os aspectos funcionais e comportamentais de um serviço web; *goals*, que representa os objetivos do usuário; e finalmente, *mediators*, que provê suporte à interoperabilidade entre diferentes elementos WSMO.

O padrão *Lightweight Semantic Descriptions for Services on the Web* (WSMO-Lite) (W3C, 2010) foi proposto como uma solução alternativa ao WSMO para realizar anotações semânticas em documentos WSDL. WSMO-Lite adaptou o modelo do padrão WSMO, considerado uma solução complexa para o desenvolvimento de serviços web semânticos (W3C, 2010; VITVAR et al., 2008), simplificando as anotações semânticas, principalmente, por meio da eliminação da definição de objetivos (*goals*) e mediadores (*mediators*) e pela utilização de taxonomias para descrever e classificar uma funcionalidade. WSMO-Lite, diferentemente do padrão WSMO, possui uma abordagem *bottom-up* de modelagem. Com essa abordagem, os serviços web e suas descrições são criados sem considerar as suas anotações semânticas. Após a criação de um serviço web, as entidades da descrição deste serviço podem ser anotadas semanticamente.

As anotações do padrão WSMO-Lite baseiam-se em cinco definições (contratos) de serviços genéricos (VITVAR et al., 2008): `Information Model`, que define o modelo de dados de entrada, saída e mensagens de falhas; `Functional Descriptions`, que define as funcionalidades do serviço, ou seja, o que o serviço é capaz de oferecer para seus clientes;

`Non-Functional Descriptions`, que detalha o ambiente de execução do serviço e a implementação do serviço; `Behavioral Description`, que define comportamentos internos e externos, chamados de `public choreography` e `private workflow`, respectivamente; e, por fim, `Technical Descriptions`, que detalha as mensagens para serialização, os protocolos de comunicação e os pontos físicos de acesso.

2.3 Sistemas Colaborativos

Esta seção apresenta uma introdução a sistemas colaborativos, com foco na edição colaborativa de documentos.

2.3.1 Visão Geral

Uma sociedade muitas vezes é caracterizada pela forma com a qual seus indivíduos interagem (ELLIS; GIBBS; REIN, 1991). O uso de computadores e outros dispositivos eletrônicos de comunicação, presentes tanto em ambientes organizacionais quanto em residenciais, contribuem para o surgimento de novas formas de interação. Uma interação pode ser classificada conforme sua relação com as dimensões do espaço e do tempo. Em relação ao tempo, uma interação pode ser realizada ao mesmo tempo (síncrona) ou em tempos diferentes (assíncrona). Em relação ao espaço, um interação pode ser realizada no mesmo local ou em diferentes locais. A Tabela 1 apresenta uma matriz de classificação das formas de interação entre espaço e tempo.

Tabela 1 – Classificação das formas de interação em relação ao tempo e ao espaço. Adaptado de (ELLIS; GIBBS; REIN, 1991)

	Mesmo tempo	**Tempos diferentes**
No mesmo local	Interação presencial	Interação assíncrona
Em locais diferentes	Interação síncrona distribuída	Interação assíncrona distribuída

.

Uma interação também pode ser classificada conforme os indivíduos presentes na interação (atores). Os atores podem ser tanto seres humanos (usuários) quanto máquinas (computadores e outros dispositivos eletrônicos). Atores podem interagir entre si por meio de diferentes formas. Os tipos de interações podem ser classificados como interações usuário-usuário, que podem ser realizadas entre um grupo de dois ou mais usuários; interações usuário-máquina, que podem ser realizadas entre um usuário e uma máquina; e, finalmente, interações máquina-máquina, que podem ser realizadas entre um grupo de duas ou mais máquinas.

A interação entre um grupo de usuários contribui para o trabalho em grupo, também chamado de trabalho colaborativo. O trabalho colaborativo tem se tornado cada vez mais essencial em atividades de uma organização (ELLIS; GIBBS; REIN, 1991). Contudo, a maioria dos sistemas computacionais suportam interações apenas entre seus usuários e o sistema em si (interações do tipo usuário-máquina). Tarefas comuns como a edição de um documento de texto ou a escrita de um código-fonte de *software* são geralmente realizadas de forma individual, tendo o envolvimento apenas de um único usuário (ser humano) e seu computador (máquina). Mesmo sistemas modelados para serem utilizados por multi-usuários proveem baixo suporte para interações de usuários com usuários (interações do tipo usuário-usuário). Este tipo de suporte é claramente necessário a fim de obter uma maior eficiência na realização de uma tarefa.

Trabalho Cooperativo Suportado por Computador, do inglês *Computer-Supported Cooperative Work* (CSCW), é uma área de estudo da computação. Esta área tem por objetivo investigar formas colaborativas com as quais grupos de trabalho realizam suas tarefas e como dispositivos computacionais podem auxiliar na realização destas tarefas. Sistemas computacionais que suportam a realização de tarefas por um grupo de pessoas com um mesmo objetivo são chamados de *groupware*. Sistemas colaborativos que suportam interações simultâneas são chamados de *groupwares* de tempo real (*groupwares* síncronos). Enquanto que sistemas colaborativos que suportam interações realizadas de forma não simultânea são chamados de *groupwares* assíncronos.

Edição colaborativa refere-se à edição de um documento sendo realizada por mais de um usuário (DILLON, 1993). A edição colaborativa deve ser realizada com base em objetivo compartilhado entre as pessoas de um grupo. Cada pessoa realiza a sua contribuição individual a fim de atingir o objetivo de forma coletiva. Escolhas efetivas na conscientização, participação e coordenação do grupo são atividades críticas para o sucesso dos resultados obtidos pela escrita colaborativa (LOWRY; CURTIS; LOWRY, 2004).

2.3.2 Desenvolvimento Colaborativo de Software

Sistemas computacionais tornaram-se vitais para a sociedade contemporânea (SOUZA; MARCZAK; PRIKLADNICKI, 2012). Diferentes organizações e negócios estão cada vez mais dependentes de funcionalidades fornecidas por estes sistemas, os quais têm se tornado cada vez mais complexos. A fim de atender aos desafios impostos por este aumento de complexidade, equipe de profissionais normalmente trabalham em conjunto no desenvolvimento de sistemas computacionais complexos. Estes profissionais trabalham colaborativamente para produzir soluções com sucesso, i.e., com qualidade, eficiência e eficácia. Entre estes especialistas, alguns papéis podem ser citados, como desenvolvedores, analistas de requisitos, arquitetos de *softwares*, analistas de negócio, gerentes de projetos,

entre outros. Os membros de uma equipe de desenvolvimento de *software* precisam coordenar suas atividades, planejar novas ações, tomar decisões, realizar as atividades previstas e, também, comunicar-se com os demais membros da equipe.

O desenvolvimento de um sistema computacional requer a criação (colaborativa) de diferentes tipos de artefatos, tais como a especificação (textual) de requisitos do sistema, diagramas de casos de uso e de classe UML, e o código-fonte do sistema em uma dada linguagem de programação. Mudanças em quaisquer destes artefatos devem ser alinhadas (sincronizadas) corretamente de modo a evitar erros na interpretação, construção e execução deste sistema computacional.

Diversas ferramentas de desenvolvimento de *software* dão suporte à colaboração entre os profissionais envolvidos no processo de desenvolvimento. Estas ferramentas proveem suporte desde a especificação colaborativa de requisitos até a construção de diagramas e códigos-fonte. Tal suporte pode ocorrer de maneira síncrona ou assíncrona, passando por editores colaborativos de texto, que possibilitam que diferentes desenvolvedores de *software* escrevam o código-fonte ao mesmo tempo, até sistemas que possibilitam gerenciar o acesso compartilhado de artefatos computacionais entre um determinado grupo de usuários (profissionais).

Durante o processo de edição colaborativa de um documento qualquer, a comunicação é essencial, visto que os autores devem concordar com o que será feito e também reportar (informar) o que foi feito após a conclusão de cada parte escrita (tarefa concluída). A edição colaborativa é regularmente aplicada em documentos textuais ou documentos de código-fonte de *softwares*. Contribuições assíncronas e remotas são muito eficientes, visto que os atores não necessitam se reunir a fim de realizar o trabalho (edição) colaborativamente. Porém, o gerenciamento das tarefas deve ser realizado com muito cuidado e atenção, envolvendo a divisão e a distribuição das tarefas entre os colaboradores envolvidos no processo. As tarefas podem ser feitas sequencialmente, a fim de que não haja problemas de sincronização entre os colaboradores, ou estas podem ser realizadas simultaneamente (forma síncrona). De qualquer maneira, o processo de edição deve ser minuciosamente planejado, documentado e revisado.

2.4 Ferramentas de Suporte à Anotação Semântica

Poucas ferramentas computacionais estão disponíveis para auxiliar e/ou automatizar a tarefa de anotar semanticamente um serviço web utilizando a abordagem SAWSDL. Essas ferramentas são geralmente utilizadas para anotar diretamente no código WSDL/XML das

descrições de serviços web. Esta seção apresenta uma visão geral de quatro ferramentas, bem como uma avaliação das mesmas.

2.4.1 Radiant

Radiant (MILLER et al., 2005) é um *plugin* do IDE Eclipse. Radiant faz parte do conjunto de ferramentas METEOR-S, usadas para criação de processos e serviços web semânticos. Radiant provê suporte para as linguagens *Web Service Semantics* (WSDL-S) (W3C, 2005b) e SAWSDL (W3C, 2007b), permitindo que usuários adicionem, por meio de uma interface gráfica, anotações semânticas às descrições de serviços web (MILLER et al., 2005). Para facilitar a compreensão dos conceitos de uma ontologia envolvidos na anotação semântica, Radiant também provê um visualizador de ontologia.

Radiant abstrai parcialmente detalhes técnicos de documentos WSDL e OWL, facilitando a tarefa de anotar um serviço web. A abstração ocorre por meio da representação visual de elementos de uma descrição de serviço web (WSDL) e de uma ontologia em um formato de árvore (*tree-view*). A Figura 4 ilustra a interface gráfica de Radiant. Na lateral esquerda, encontram-se representados os elementos WSDL. Na lateral direita, encontram-se representados os elementos de uma ontologia. Finalmente, no centro, encontra-se a especificação WSDL anotada.

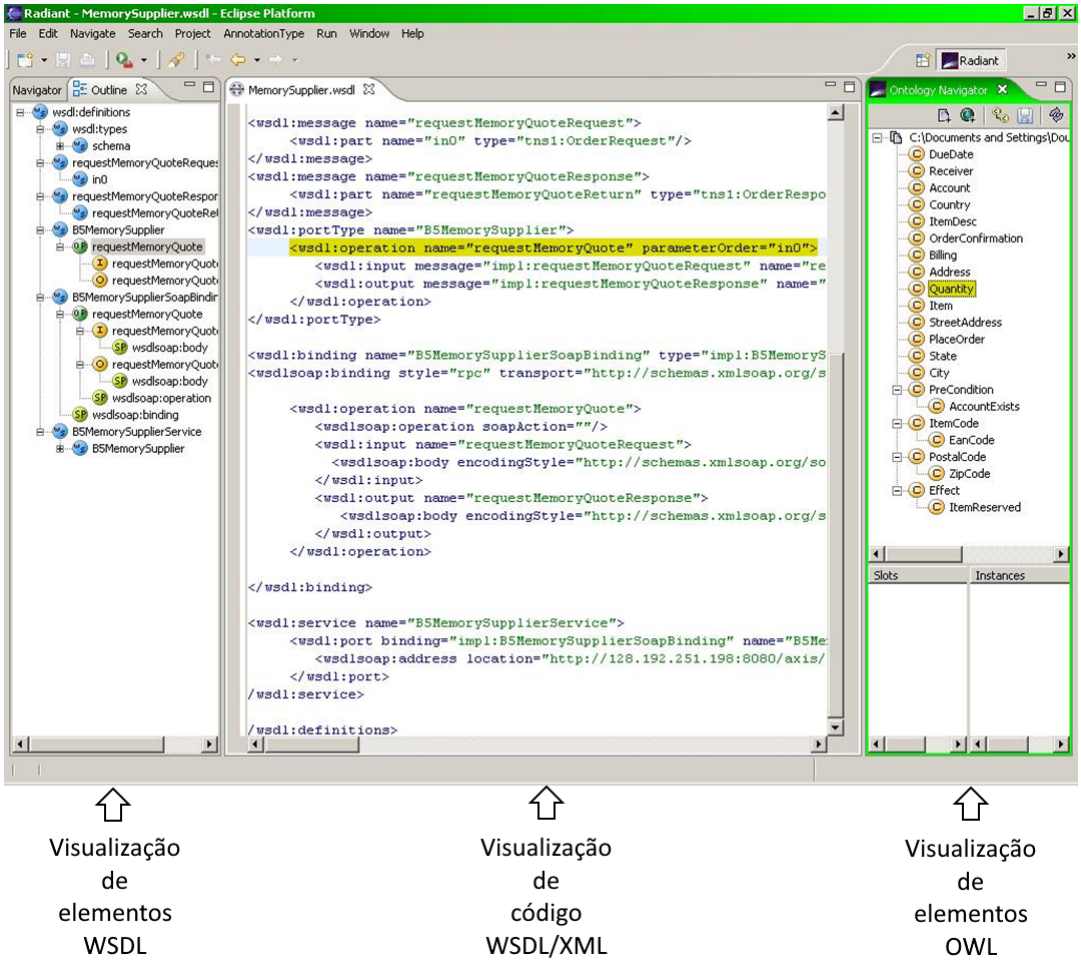
Por meio de Radiant, também é possível anotar um documento WSDL utilizando o recurso *drag-and-drop*. O usuário pode arrastar um conceito de uma ontologia para cima de um elemento WSDL. Com isso, a anotação semântica é adicionada automaticamente ao documento WSDL. As anotações semânticas podem ser visualizadas por meio do painel central.

Apesar destas representações visuais (abstrações parciais) contribuírem para uma melhor compreensão dos elementos envolvidos na anotação semântica, ainda faz-se necessário lidar diretamente com código WSDL/XML de descrições de serviços e de atributos SAWSDL. Adicionalmente, o usuário necessita percorrer uma ontologia inteira a fim de obter os conceitos apropriados para a anotação semântica de um serviço web. Como uma ontologia pode conter um grande conjunto de conceitos e relacionamentos, anotar semanticamente um serviço pode requerer um esforço significativo.

2.4.2 Iridescent

Iridescent (STAVROPOULOS; VRAKAS; VLAHAVAS, 2013) é uma ferramenta *standalone* que permite anotar descrições de serviços web de acordo com o padrão SAWSDL. As anotações são inseridas com base nas referências de termos de uma ontologia representada

Figura 4 – Interface gráfica de usuário da ferramenta Radiant.



Fonte: Adaptado de (CARDOSO; MILLER; EMANI, 2008)

no formato OWL (W3C, 2012a).

Por meio de uma interface gráfica de usuário, Iridescent permite que entidades identificadas na descrição de um serviço web sejam anotadas com os termos contidos na ontologia. A visualização de uma descrição de serviço web e uma ontologia ocorre por meio de uma representação visual no formato de árvore (*tree-view*), semelhante ao *plugin* Radiant (MILLER et al., 2005).

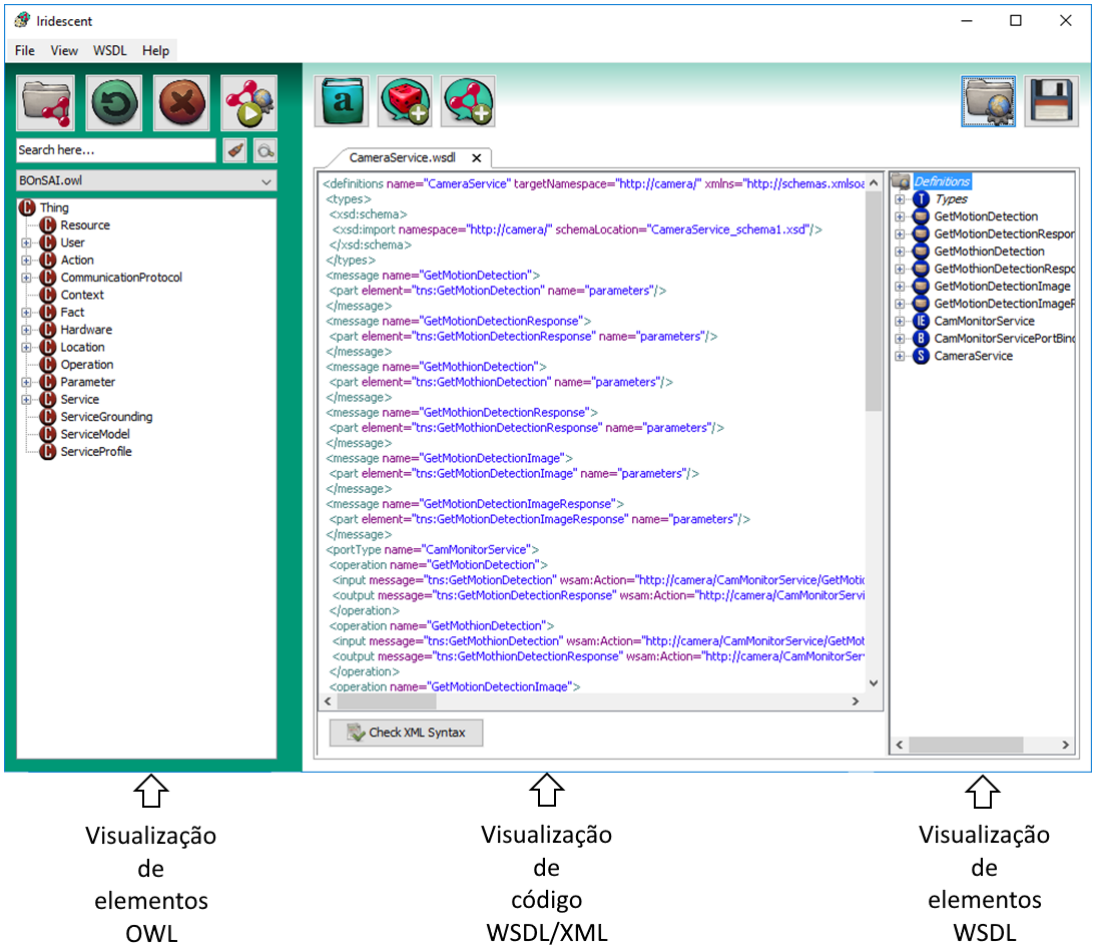
Iridescent oferece recursos que automatizam parcialmente o processo de anotação semântica nos serviços web. A partir de um algoritmo de busca, conceitos de uma ontologia são sugeridos para os termos contidos em uma descrição de um serviço, servindo como um guia para o usuário. O usuário pode acatar ou não as sugestões da ferramenta. Iridescent também provê recursos para que os métodos *Lifting Schema Mapping* e *Lowering Schema Mapping* possam ser implementados e descritos no WSDL.

A utilização do recurso *drag-and-drop* também auxilia o processo. O usuário pode arrastar elementos (conceitos) da representação visual de uma ontologia, do painel lateral

direito, para um elemento WSDL presente no código disponibilizado pelo painel central. As anotações semânticas (SAWSDL) podem ser vistas diretamente no código WSDL/XML.

A Figura 5 ilustra a interface gráfica de usuário de Radiant. Na lateral esquerda, encontram-se representados os elementos de uma ontologia. Na lateral direita, encontram-se representados os elementos WSDL. Finalmente, no centro, encontra-se a especificação WSDL anotada

Figura 5 – Interface gráfica de usuário da ferramenta Iridescent.



Fonte: Adaptado de (STAVROPOULOS; VRAKAS; VLAHAVAS, 2013)

2.4.3 EasyWSDL & EasySAWSDL

EasyWSDL (EASYWSDL, 2016) é uma biblioteca Java para suporte à leitura, à edição e à criação de documentos WSDL e XML Schema (XSD). EasyWSDL foi desenvolvida pelo *OW2 Consortium* (OW2, 2016), organização da comunidade *open-source*, que tem como missão promover o desenvolvimento de aplicações *middleware* e de negócio, plataformas de computação em nuvem, entre outras.

A arquitetura de EasyWSDL é extensível. Assim, a extensão EasySAWSDL (EASY-SAWSDL, 2016) foi proposta para permitir a anotação de documentos WSDL segundo o padrão SAWSDL (W3C, 2007b). EasyWSDL fornece classes e métodos específicos para a criação e a manipulação de elementos de um documento WSDL, enquanto que EasySAWSDL fornece classes e métodos específicos para a anotação semântica. A manipulação de elementos WSDL e atributos SAWSDL são abstraídos por meio da linguagem de programação. A biblioteca provê métodos claros que permitem que desenvolvedores, familiarizados com a linguagem de programação Java, possam compreender e manipular mais facilmente o documento. Neste sentido, desenvolvedores não necessitam lidar diretamente com dados sintáticos das especificações WSDL e SAWSDL para a criação de serviços web semânticos.

A Listagem 1 apresenta um exemplo de utilização da biblioteca EasySAWSDL para a anotação de uma especificação WSDL. As linhas 1 e 2 contém instruções para a leitura de um documento WSDL. O objeto `reader` é utilizado para a leitura de um documento WSDL, enquanto que o objeto `desc` é utilizado para o armazenamento de um documento WSDL. A linha 3 contém uma instrução necessária para a escrita de atributos SAWSDL. A linha 4 contém uma instrução para a leitura de uma descrição WSDL (objeto `desc`) por meio do escritor SAWSDL. A linha 5 contém uma instrução para obter um elemento WSDL (objeto `ele`) dado um identificador. Finalmente, a linha 6 contém uma instrução para adicionar um atributo *Model Reference*, com o uso de uma URI de um conceito de uma ontologia.

```
1    WSDLReader reader = WSDLFactory.newInstance().newWSDLReader();
2    Description desc = reader.read(new URL("http://grasews/svc.wsdl"));
3    SAWSDLWriter writer = SAWSDLFactory.newInstance().newSAWSDLWriter();
4    Document doc = writer.getDocument(desc);
5    Element ele = doc.getElementById("elementId");
6    ele.addModelReference(new URI("http://grasews.owl\#concept"));
```

Listagem 1 – Exemplo de uso da biblioteca EasySAWSDL.

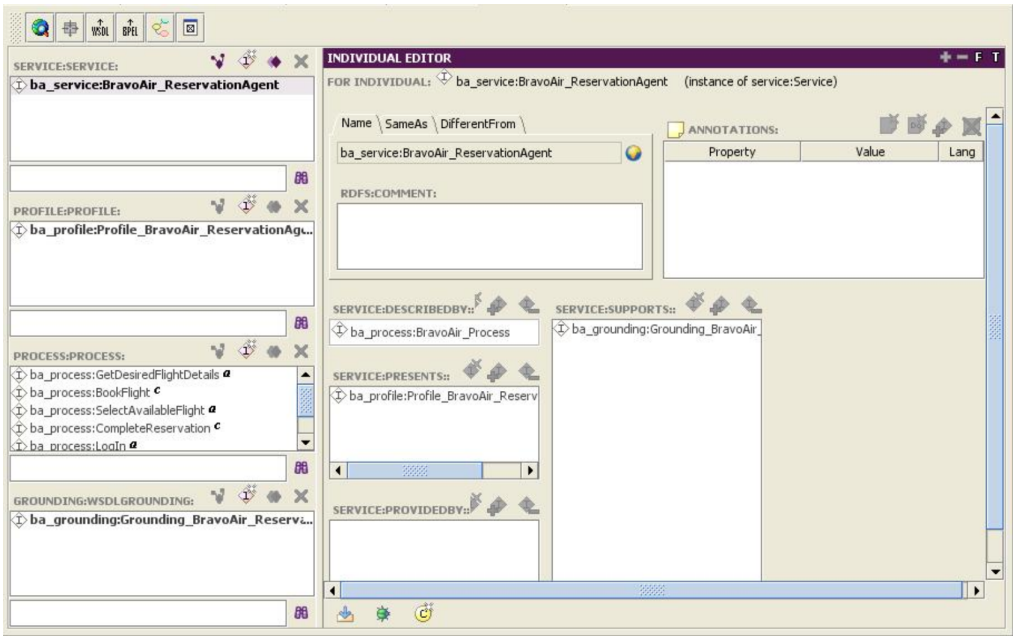
2.4.4 OWL-S Editor

OWL-S Editor é uma ferramenta de anotação semântica com suporte ao padrão OWL-S, distribuída nos formatos plugin (SAADATI; DENKER, 2016; DENKER; ELENIUS; MARTIN, 2016; ELENIUS, 2005) e *standalone* (SCICLUNA, 2016). OWL-S Editor plugin é uma extensão para o editor de ontologias Protégé (PROTéGé, 2016). Este plugin tem por objetivo fornecer uma interface fácil e intuitiva para o desenvolvimento de serviços OWL-S, mesmo para usuários inexperientes (SAADATI; DENKER, 2016). OWL-S Editor plugin é apresentado para o usuário no formato chamado *tab-widget*, formato no qual uma

nova aba é acoplada à ferramenta principal (Protégé). Esta aba, portanto, contém todas as funcionalidades necessárias para criar e editar uma ontologia OWL-S.

OWL-S Editor *plugin* permite a criação de instâncias das classes `Service`, `Profile`, `Process` e `Grounding`, bem como a criação de instâncias de suas propriedades, como, por exemplo, os processos para cada operação na classe `Process` e o mapeamento das operações para os processos na classe `Grounding`. A Figura 6 ilustra a interface gráfica de usuário de OWL-S Editor *plugin*, onde na lateral esquerda são apresentados os painéis para a criação de instâncias das classes `Service`, `Profile`, `Process` e `Grounding` e ao centro são apresentados os painéis para a criação de propriedades referentes às instâncias de classes. As propriedades mandatórias de serem criadas para uma ontologia OWL-S são destacadas em uma marcação em vermelho pela ferramenta, auxiliando o usuário no desenvolvimento da ontologia. Dessa forma, o resultado obtido pelo OWL-S Editor *plugin* é um novo documento OWL, contendo a especificação OWL-S de um serviço web. Apesar de o padrão ser apresentado como quatro ontologias separadas (`Service`, `Profile`, `Process` e `Grounding`), o OWL-S Editor plugin cria apenas um documento OWL com todas as ontologias desenvolvidas.

Figura 6 – Interface gráfica de usuário da ferramenta OWL-S Editor *plugin*.



Fonte: Autoria própria.

OWL-S Editor no formato *standalone* tem como objetivo auxiliar usuários a anotarem semanticamente um serviço web, por meio de uma ontologia no formato OWL-S. OWL-S Editor *standalone* possui três componentes principais: `Creator`, `Validator` e `Visualiser`. O componente `Creator` possibilita que descrições OWL-S sejam criadas por meio de um modelo (*template*) ou por meio do modo *wizard*, chamado `OwlsWiz`. O modo `OwlsWiz` aceita como parâmetro de entrada um documento WSDL e extrai informações

parciais deste documento, a fim de criar uma descrição no formato OWL-S. A funcionalidade `Visual Composer`, pertencente ao modo `OwlsWiz`, permite que expressões lógicas e fluxos de dados possam ser construídos no `ServiceModel`. O `Visual Composer` permite criar um processo composto a partir de processos atômicos extraídos do documento WSDL, utilizando recursos visuais, como *drag-and-drop*. A criação de um processo composto é feita por meio de um formato padrão de diagramas, semelhante a um diagrama de atividades da UML. O componente `Validator` permite que URIs utilizados na descrição OWL-S sejam verificados e validados, garantindo que a sintaxe da ontologia esteja em conformidade com o padrão OWL-S. Finalmente, o componente `Visualiser` possibilita que o usuário visualize as descrições OWL-S de uma forma gráfica.

2.4.5 Avaliação Ferramental

As ferramentas de suporte à anotação semântica segundo o padrão SAWSDL, ou seja, Radiant, Iridescent, e EasyWSDL & EasySAWSDL, foram avaliadas segundo três critérios de usabilidade: i) facilidade de aprendizado, ii) maximização da produtividade e iii) minimização da taxa de erros. Facilidade de aprendizado refere-se ao esforço (tempo) necessário para aprender a utilizar a ferramenta e, portanto, atingir os resultados esperados. Quanto maior a facilidade de aprendizado, menor é a curva de aprendizagem. Maximização da produtividade refere-se à eficiência com a qual as ferramentas conseguem auxiliar seus usuários a atingirem os resultados esperados. Quanto maior a maximização da produtividade, maior é a eficiência. Finalmente, minimização da taxa de erros refere-se à facilidade com a qual exceções e fluxos não esperados executados por seus usuários. Quanto maior a minimização da taxa de erros, menor é a presença de erros.

Para realizar esta avaliação, utilizamos as três ferramentas para reproduzir parcialmente um conjunto de anotações semânticas desenvolvidas para diferentes serviços web na área de expressão gênica segundo o padrão SAWSDL (GUARDIA et al., 2017). Cada diferente critério de usabilidade foi avaliado segundo uma classificação envolvendo níveis alto, médio e baixo.

No critério de facilidade de aprendizado, o *plugin* Radiant foi avaliado em nível médio em razão da dificuldade associada à instalação e à configuração deste *plugin*. Já a ferramenta Iridescent foi avaliada em nível alto neste critério por se tratar de uma ferramenta *standalone* e, consequentemente, por não possuir dependências de outras ferramentas. Adicionalmente, Iridescent possui uma interface gráfica mais intuitiva quando comparado ao *plugin* Radiant. Com isso, esta ferramenta apresentou uma maior facilidade de aprendizado em relação às demais. EasyWSDL & EasySAWSDL foram avaliadas em nível baixo no critério de facilidade de aprendizado, por se tratar de uma biblioteca de programação e, portanto, dependerem de conhecimento técnico prévio na linguagem de

programação utilizada (Java).

Em relação ao critério de maximização da produtividade, as ferramentas Radiant e Iridescent foram avaliadas em nível médio, dado que, uma vez disponíveis, estas ferramentas permitiram a anotação semântica de forma simples. Entretanto, as anotações produzidas não são claramente percebidas dado que o usuário necessita visualizar a própria especificação WSDL para ter total compreensão das ações (anotações) realizadas. A biblioteca EasyWSDL & EasySAWSDL foi avaliada em nível baixo, dada a necessidade de criar um programa para permitir a anotação semântica de uma dada especificação WSDL.

Por fim, em relação ao critério de minimização da taxa de erros, Radiant e EasyWSDL & EasySAWSDL não apresentaram erros durante sua utilização, portanto, sendo classificados em nível alto. Entretanto, Iridescent apresentou algumas falhas não tratadas ou pouco explicadas ao usuário e, consequentemente, obtendo uma avaliação de nível médio.

A Tabela 2 resume os critérios de usabilidade avaliados para cada ferramenta.

Tabela 2 – Avaliação de usabilidade para as ferramentas disponíveis.

Ferramenta	Facilidade de aprendizado	Maximização da produtividade	Minimização da taxa de erros
Radiant	Médio	Médio	Alto
Iridescent	Alto	Médio	Médio
EasySAWSDL	Baixo	Baixo	Alto

2.5 Notação Visual

Esta seção apresenta uma visão geral acerca de um conjunto de princípios associados ao desenvolvimento de notações visuais.

2.5.1 Visão Geral

Notações visuais são uma das formas mais antigas e efetivas de representação do conhecimento (DAVIES, 1990; MATHEWS, 1991; RAO, 2005). Diferentes tipos de recursos visuais, tais como formas geométricas, cores, ícones, texturas e brilhos, podem ser utilizados para representar diferentes conceitos e seus relacionamentos em um dado domínio (SMITH et al., 2004; MOODY, 2009).

Notações visuais são mais eficazes para a comunicação e a transmissão de informações do que outras formas de comunicação, incluindo a comunicação verbal e textual (MOODY, 2009). Tal característica advém da melhor capacidade do cérebro humano em processar (paralelamente) representações visuais que utilizam arranjos espaciais de elementos gráficos (bidimensionais). Cerca de um quarto do cérebro humano é dedicado à visão. Assim, o sistema visual de um cérebro humano tem a capacidade de processar mais informações do que todos os demais sentidos combinados.

Atualmente, notações visuais tem sido amplamente utilizadas em diversas áreas de conhecimento. Por exemplo, *Unified Modeling Language* (UML) (OMG, 2017) tem sido utilizada na representação de artefatos de *software*; *Business Process Management Notation* (BMPN) (OMG, 2011) tem sido utilizada na representação de processos de negócio; *Systems Biology Graphical Notation* (SBGN) (NOVèRE et al., 2009; TOURé et al., 2018) tem sido utilizada na representação de modelos biológicos *in silico*; enquanto que *Synthetic Biology Open Language* (SBOL) (QUINN et al., 2015) tem sido utilizada na representação de modelos de engenharia genética.

2.5.2 Princípios da Notação Visual

Historicamente, pesquisadores e projetistas de notações visuais ignoraram ou subestimaram princípios básicos da sintaxe para representar visualmente elementos de um modelo (MOODY, 2009). O desenvolvimento das notações visuais existentes em diversas áreas de conhecimento, como na engenharia de *software*, foi, em sua maioria, visando atender apenas a detalhes semânticos, pouco atentando a convenções visuais.

De modo a projetar notações visuais cognitivamente eficazes, otimizando, portanto, o suporte à comunicação humana e à solução de problemas, Moody propôs um conjunto de nove princípios, chamados de Física das Notações (MOODY, 2009). Estes princípios tem como base a representação visual de constructos (MOODY, 2009; POPESCU; WEGMANN, 2014). Constructos são modelos criados mentalmente e usados por especialistas para compreender uma parte específica de um domínio ou teoria. Construções mentais ou sínteses feitas a partir da combinação de vários elementos contribuem para a definição de um constructo. Tais construções mentais têm por objetivo a compreensão da realidade que deriva das observações e percepções individuais, resultantes de experiências (passadas ou presentes) de uma pessoa.

Os princípios da Física das Notações concentram-se nas propriedades físicas de uma notação (perceptivo) em vez de suas propriedades lógicas (semânticas). Eles foram sintetizados a partir de evidências empíricas de uma ampla variedade de domínios e baseiam-se em uma teoria explícita de como as notações visuais se comunicam. Esses princípios podem ser usados para construir novas notações visuais, bem como para avaliar,

comparar e melhorar notações visuais existentes. Cada princípio é independente dos demais, podendo ser aplicado separadamente.

2.5.2.1 Princípio da Integração Cognitiva

Notações visuais devem ter informações (representações) integradas entre diferentes modelos. Ou seja, se um elemento é representado em dois modelos distintos, este elemento deve possuir a mesma representação visual em ambos os modelos. Este princípio apenas se aplica quando múltiplos modelos são usados. Os modelos podem ser do mesmo tipo (integração homogênea) ou de tipos diferentes (integração heterogênea). Mecanismos de integração cognitiva referem-se a integrações conceituais (sumarização e momento visual) e integrações perceptivas (sinalização, orientação e mapa de navegação).

A Figura 7 ilustra a aplicação do princípio da integração cognitiva, onde pode ser visto a mesma representação visual para um ator em um diagrama de sequência da UML (Figura 7a) e um diagrama de caso de uso da UML (Figura 7b)

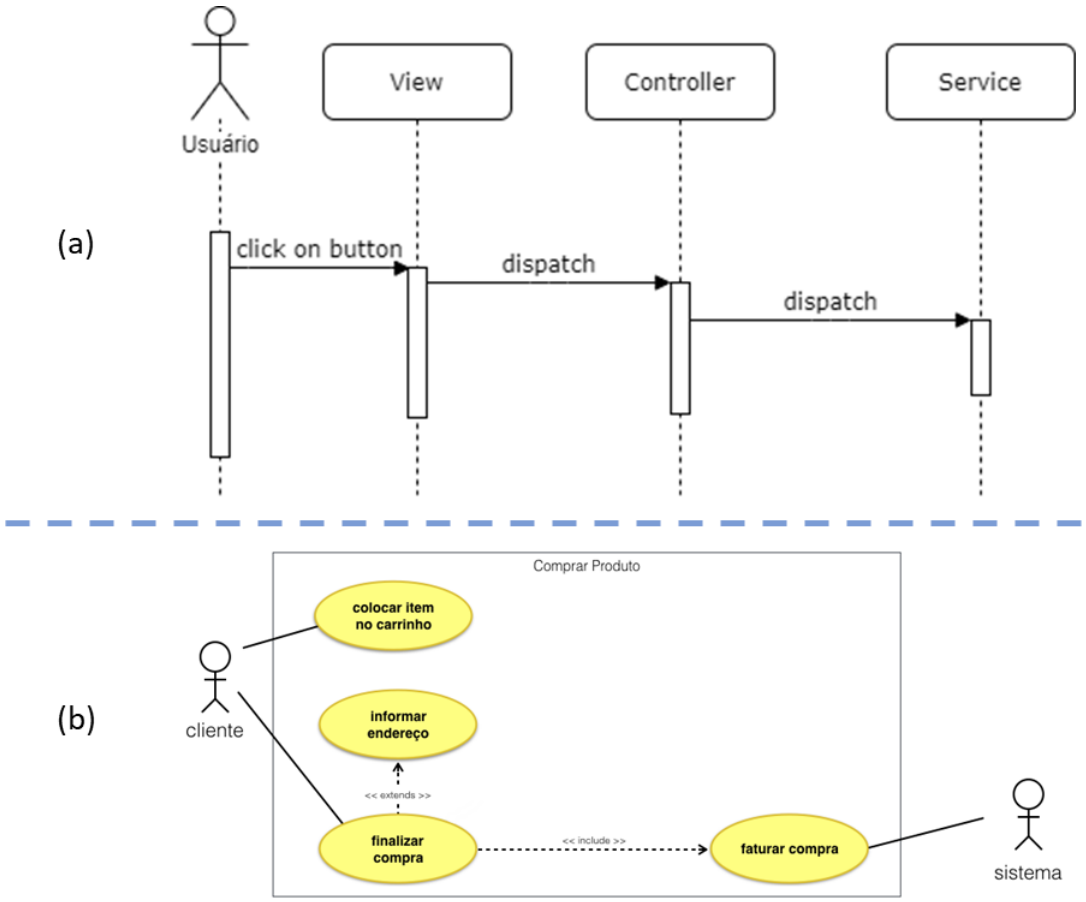
2.5.2.2 Princípio do Ajuste Cognitivo

Notações visuais devem ter diferentes dialetos para diferenciar a comunicação entre diferentes públicos-alvos (audiência). O público-alvo pode ser composto por especialistas ou por iniciantes em um dado domínio. Diferentes dialetos visuais também podem ser necessários para diferentes objetivos (tarefas). Notações cognitivamente eficazes para uma audiência iniciante podem não ser cognitivamente eficazes para uma audiência de especialistas e vice-versa. O princípio do ajuste cognitivo afirma, portanto, que variações de uma mesma notação visual são necessárias e devem ser utilizadas conforme a audiência e os diferentes objetivos que pretende-se atingir com o uso de um dado modelo.

2.5.2.3 Princípio da Diferenciação Perceptiva

Notações visuais devem ser claramente distinguíveis umas das outras. Esta distinção pode ser obtida aumentando as diferenças visuais entre as notações. Características como tamanho, forma e cor (expressividade visual) contribuem efetivamente na distinção de elementos visuais. Neste sentido, diferentes constructos semânticos devem ser facilmente identificados de acordo com suas diferentes (distintas) notações visuais. Informações textuais (codificação dupla) também podem ser utilizadas a fim de facilitar esta distinção.

Figura 7 – Princípio da integração cognitiva. (a) Representação visual de um ator no diagrama de sequência da UML. (b) Representação visual de um ator no diagrama de classes da UML.



Fonte: Autoria própria.

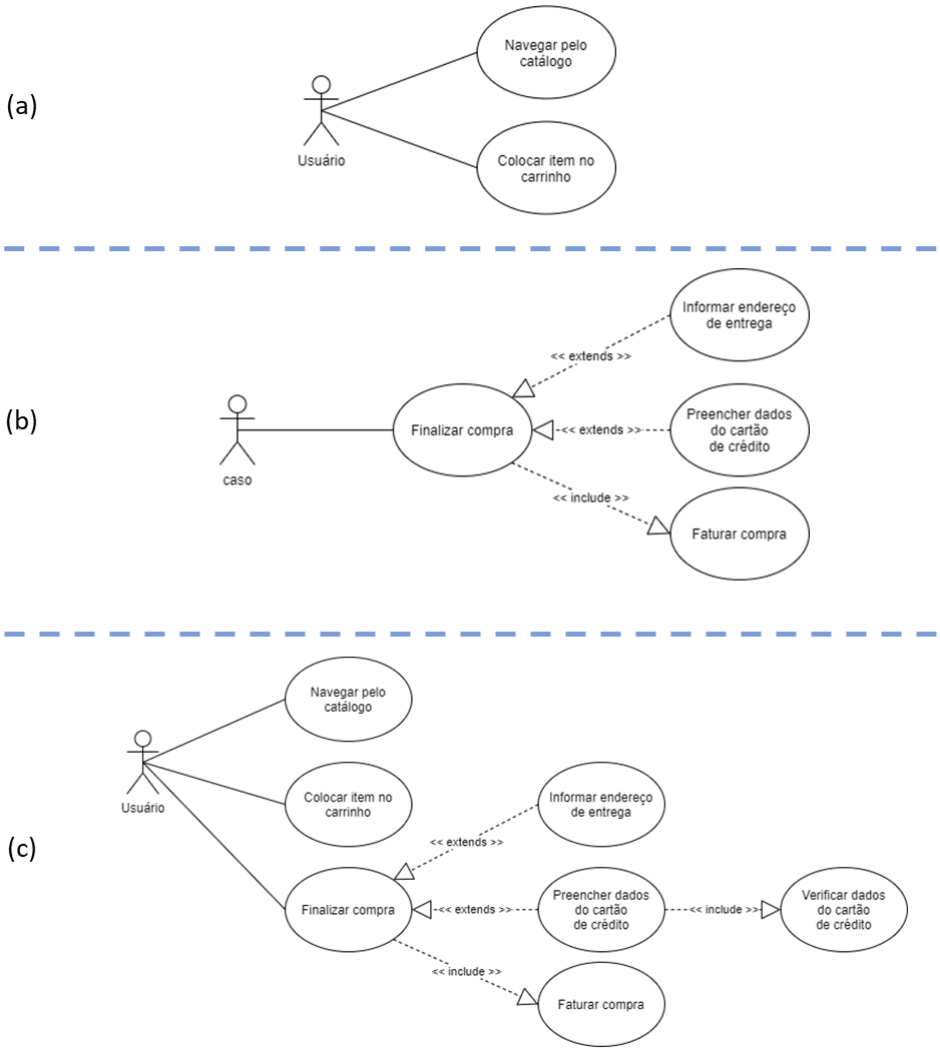
2.5.2.4 Princípio da Complexidade Gerenciável

Modelos podem se tornar bastante complexos, dependendo diretamente da complexidade dos domínios em que eles estão sendo utilizados e do conhecimento que pretende ser representado (MOODY, 2009). Esta complexidade diagramática pode também ser medida pela quantidade de elementos visuais de um modelo. Quanto mais detalhada e granular a representação visual de um conhecimento, maior é a complexidade de um modelo. Neste sentido, notações visuais devem incluir mecanismos explícitos para lidar com modelos complexos, como, por exemplo, por meio da divisão destes em sub-modelos.

A complexidade pode também ser tratada por meio da modularização de um modelo complexo. A modularização envolve tanto a criação de modelos mais genéricos (com menos detalhes), de modo a obter uma visão mais abstrata do modelo, quanto a criação de modelos mais específicos (com mais detalhes), de modo a obter uma visão mais concreta e realista do modelo em desenvolvimento.

A Figura 8 ilustra um exemplo de modularização aplicado a um diagrama de caso de uso. A Figura 8a e a Figura 8b ilustram dois diagramas de caso de uso mais simples, enquanto que a Figura 8c ilustra um diagrama de caso de uso maior, composto pela junção dos sub-modelos (modularização) ilustrados em 8a e 8b .

Figura 8 – Princípio da Complexidade Gerenciável. (a) e (b) Sub-modelos do caso de uso. (c) Caso de uso completo, com a junção de (a) e (b).



Fonte: Autoria própria.

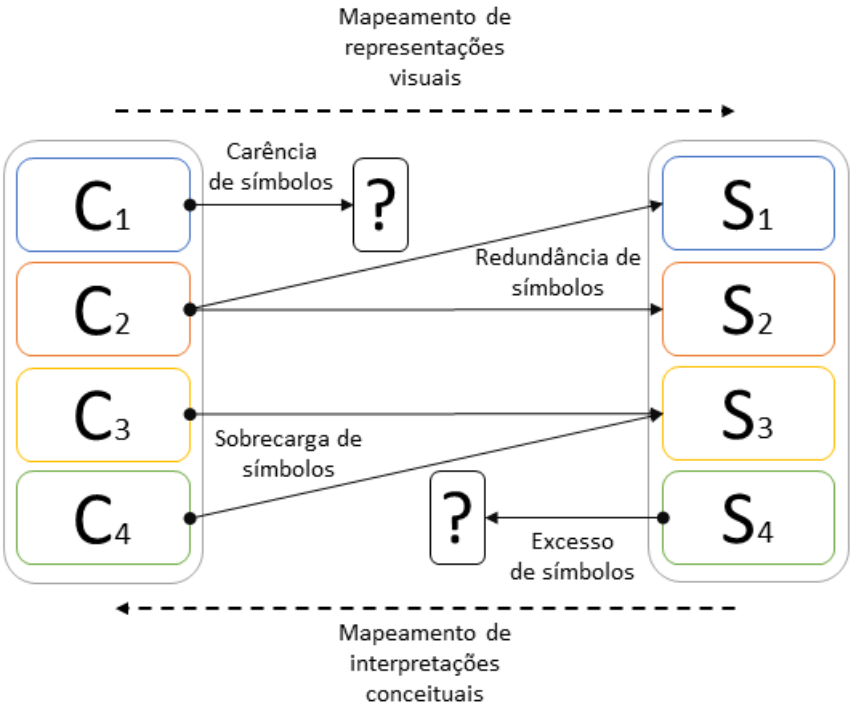
2.5.2.5 Princípio da Clareza Semiótica

O princípio da clareza semiótica define que um constructo semântico (conceito) deve ser representado por exatamente um elemento gráfico (visual) e vice-versa, isto é, deve haver uma relação 1:1 (um-para-um) entre constructos semânticos e notações visuais (MOODY, 2009).

Há quatro formas de violação deste princípio, chamadas de anomalias da clareza semiótica: i) redundância de símbolos, i.e., um constructo semântico é representado por várias notações visuais; ii) sobrecarga de símbolos, i.e., uma notação visual representa mais de um constructo semântico; iii) excesso de símbolos, i.e., a notação visual é criada e não representa qualquer constructo semântico; e, por fim, iv) carência de símbolos, i.e., não há nenhuma notação visual prevista para um certo constructo semântico.

A Figura 9 resume as quatro anomalias da clareza semiótica. Os elementos C_i, à esquerda, representam constructos semânticos, enquanto que os elementos S_i, à direita, representam símbolos.

Figura 9 – Anomalias do Princípio da Clareza Semiótica.



Fonte: Adaptado de (MOODY, 2009)

2.5.2.6 Princípio da Expressividade Visual

Notações visuais podem fazer uso de sete diferente variáveis para melhorar a expressividade de uma notação visual: posição, tamanho, brilho, textura, cor, orientação e forma. A expressividade visual é determinada pelo número de variáveis visuais usadas em uma notação e pela extensão em que elas são utilizadas. No contexto do nosso trabalho, apenas as variáveis cor, tamanho, forma e posição são consideradas relevantes.

A cor é uma das mais importantes variáveis visuais, uma vez que o contraste de cor é interpretado mais rapidamente do que as diferenças entre outras variáveis. Porém, cores devem ser usadas com cuidado e apenas em conjunto (de forma redundante) com

outras variáveis, dado que suas diferenças desaparecem quando modelos são impressos em escala de cinza ou então quando os modelos são utilizados por pessoas daltônicas.

Elementos visuais também podem ser diferenciados pelo uso de diferentes formas e tamanhos. Por exemplo, retângulos podem ser utilizados para representar atividades enquanto losangos podem ser utilizados para representar decisões na BPMN (OMG, 2011). Em um modelo de nuvem de palavras (HEIMERL et al., 2014), o tamanho de uma dada palavra-chave é proporcional à relevância desta no modelo. Quanto mais revelante, maior é o tamanho da palavra em relação ao tamanho das demais.

Elementos visuais podem ser diferenciados conforme o seu posicionamento, ou seja, conforme a sua posição no eixo X (horizontal) e no eixo Y (vertical). Por exemplo, em um organograma empresarial, profissionais líderes podem ser apresentados verticalmente (eixo Y) acima de seu time, bem como profissionais com o mesmo cargo podem ser apresentados no mesmo nível vertical (eixo Y), porém espaçados (distribuídos) horizontalmente (eixo X).

Diferentes brilhos e texturas também podem contribuir para uma maior expressividade visual. O brilho pode alterar a tonalidade de uma representação visual e muitas vezes é utilizado em conjunto com as cores a fim de se obter uma maior diferenciação entre elementos de um modelo. Por fim, orientações de notações visuais, como diferentes ângulos (rotações) utilizados para um mesmo elemento visual, podem contribuir para representar diferentes conceitos semânticos em um dado modelo.

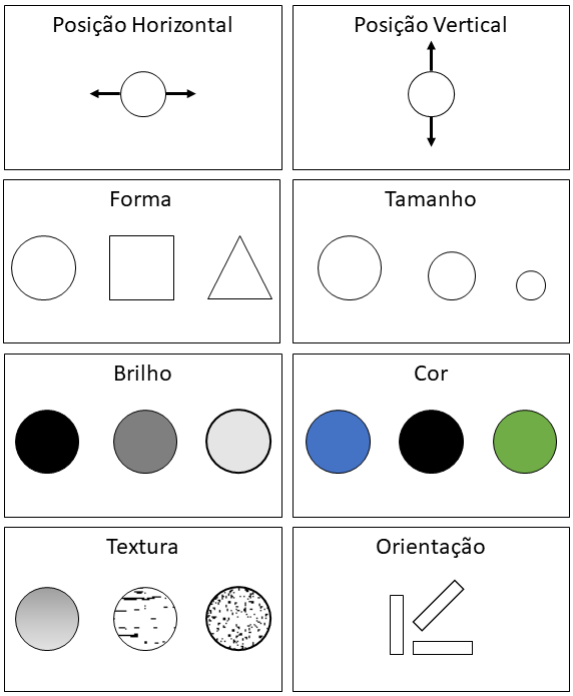
A Figura 10 ilustra exemplos de uso das variáveis visuais utilizadas para melhorar a expressividade de uma notação visual.

2.5.2.7 Princípio da Economia Gráfica

Notações visuais devem ser utilizadas com parcimônia, ou seja, o número de elementos gráficos deve ser limitado e de fácil gerenciamento. Um grande número de convenções aumenta a complexidade de um modelo, dificultando o seu entendimento.

A fim de satisfazer o princípio da economia gráfica, três abordagens podem ser utilizadas. A primeira abordagem refere-se à remoção de constructos semânticos, ou seja, reduz-se a quantidade de constructos semânticos e, consequentemente, de representações visuais necessárias. Esta redução deve ser realizada com cuidado a fim de que o princípio clareza semiótica não venha a ser violado. A segunda abordagem refere-se à simples eliminação de representações visuais. Contudo, esta abordagem é mais arriscada, visto que a anomalia carência de símbolos, da clareza semiótica, pode ser mais facilmente introduzida. Por fim, a terceira abordagem refere-se ao aumento da expressividade visual. Esta abordagem pressupõe que com o aumento de características (atributos) visuais de um

Figura 10 – Variáveis visuais utilizadas para aumentar a Expressividade Visual.



Fonte: Adaptado de (MOODY, 2009)

dado elemento, enriquecendo, portanto, a sua expressividade visual, este elemento pode ser tornar suficientemente claro para substituir simultaneamente dois ou mais elementos em um dado modelo.

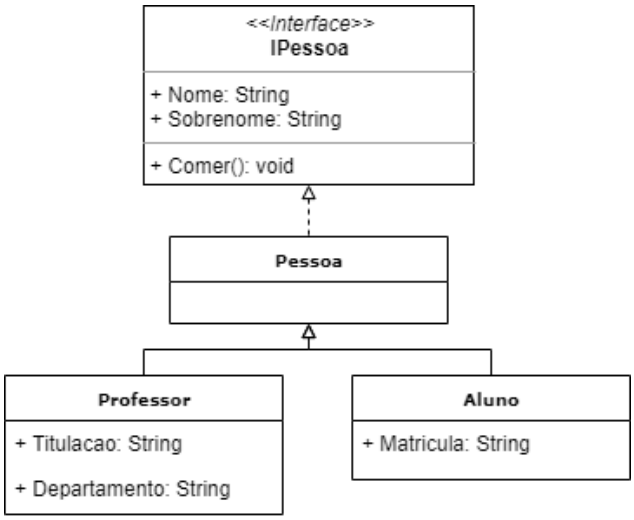
2.5.2.8 Princípio da Codificação Dupla

Textos contribuem para a compreensão de um modelo quando usados em conjunto com representações gráficas. Assim, notações visuais devem utilizar tanto elementos gráficos quanto elementos textuais. Uma representação textual não deve substituir uma representação gráfica, mas sim complementá-la. O uso de elementos textuais com elementos gráficos contribui para uma transmissão de informação mais efetiva do que quando usados de forma separada. Entretanto, é importante que as representações gráficas sejam distinguíveis com base nas ilustrações. Elementos como rótulos (*labels*) devem ser utilizados apenas para distinguir instâncias de um mesma representação gráfica, mas não entre tipos (elementos) diferentes.

A Figura 11 ilustra um exemplo do uso da Codificação Dupla aplicado ao elemento *IPessoa*. A identificação de *IPessoa* como uma interface seria dificultada se não utilizássemos a Codificação Dupla. Seria necessário um olhar atento ao relacionamento entre a classe *Pessoa* e o elemento *IPessoa*, para que este pudesse ser interpretado como uma

interface ao invés de uma classe.

Figura 11 – Princípio da Codificação Dupla.



Fonte: Autoria própria.

2.5.2.9 Princípio da Transparência Semântica

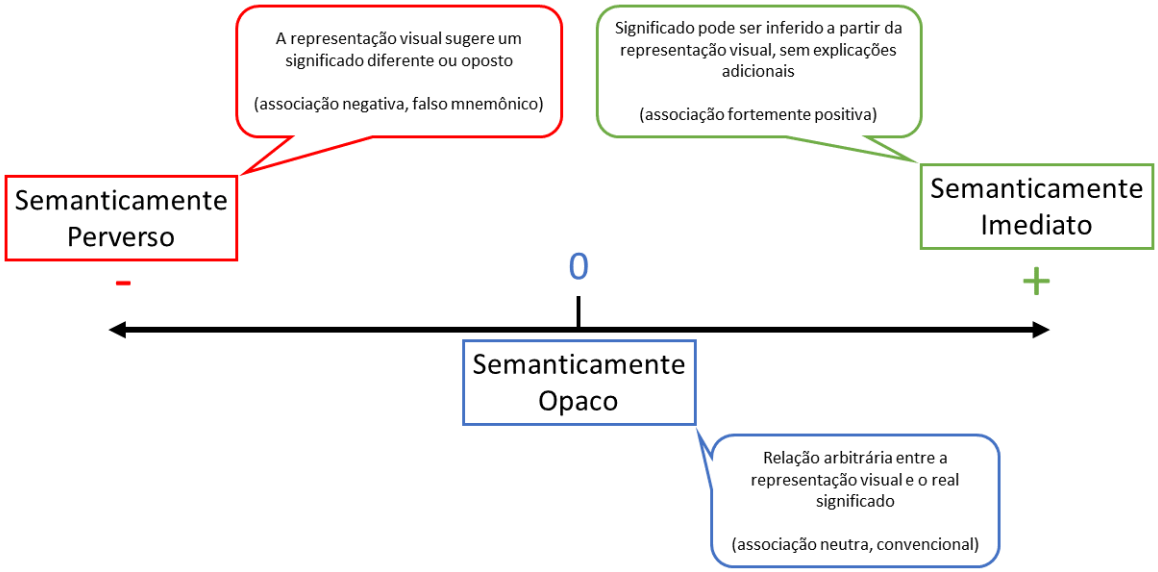
Notações visuais devem utilizar elementos gráficos cuja aparência sugerem o significado de seus constructos semânticos. Transparência semântica avalia a facilidade com a qual uma representação visual é relacionada ao seu real significado (constructo).

A transparência semântica de um elemento pode ser classificada em: semanticamente imediata, semanticamente opaca ou semanticamente perversa. Um elemento é semanticamente imediato se um leitor iniciante é capaz de inferir o seu significado por si só a partir da sua aparência. Por exemplo, um boneco para representar uma pessoa. Um elemento é semanticamente opaco (ou convencional) se existe uma relação puramente arbitrária entre a sua aparência e seu significado. Por exemplo, um retângulo para representar uma entidade em um diagrama entidade-relacionamento (DER). Finalmente, um elemento é semanticamente perverso (ou um falso mnemônico), se um leitor iniciante é susceptível a inferir um significado diferente ou até mesmo oposto ao significado do elemento. Por exemplo, uma interface pode ser interpretada como uma classe em um diagrama de classes da UML.

Uma abordagem eficaz de se modelar elementos com transparência semântica é por meio de ícones. Ícones facilitam o reconhecimento de constructos semânticos. Adicionalmente, ícones melhoram a compreensão da notação principalmente para os usuários iniciantes. Notações visuais semanticamente transparentes também reduzem a carga cognitiva, visto que elas têm mnemônicos embutidos e seus significados podem ser percebidos diretamente ou facilmente aprendidos.

A Figura 12 ilustra as medidas da transparência semântica. Quanto mais à direita (sinal positivo), mais um elemento visual é semanticamente imediato. Quanto à esquerda (sinal negativo), mais um elemento visual é semanticamente perverso. No centro do eixo, entre semanticamente imediato e semanticamente perverso, um elemento pode ser considerado semanticamente opaco (neutro).

Figura 12 – Princípio da transparência semântica.



Fonte: Adaptado de (MOODY, 2009)

2.6 *Domain-Driven Design*

Domain-Driven Design (DDD) é uma abordagem para o desenvolvimento de softwares focada em resolver requisitos complexos de desenvolvimento por meio de implementações conectadas ao redor de um núcleo (EVANS; FOWLER, 2004). O núcleo, também chamado de *domain* ou *core*, contém conceitos do domínio do negócio que ditam as regras e comportamentos de todos os componentes e camadas ao seu redor.

As premissas do DDD são: i) coloque o foco principal do projeto no domínio de negócio e em sua lógica, representados pelo núcleo do projeto; ii) baseie-se em um modelo de domínio de negócio para a criação de projetos complexos; e, por fim, iii) inicie uma colaboração criativa entre especialistas técnicos e de domínio para obter a máxima aproximação possível do centro conceitual do problema (DOMAIN-DRIVEN DESIGN COMMUNITY, 2019).

A seguir, estão listadas as principais camadas da abordagem de desenvolvimento DDD com uma visão geral de cada camada:

1. ***Apresentação***. A camada de apresentação é responsável por agrupar módulos que contenham interfaces de usuário (UI - *User Interface*) para a solução. Os módulos desta camada podem ser implementados de diversas formas, como aplicações web, *desktop/standalone*, móveis (*smart phone*, *tablet*, *smart TV*), aplicações *console*, etc;

2. ***Serviços Distribuídos***. A camada de serviços distribuídos é responsável por agrupar módulos que contenham APIs para a solução. As APIs podem ser implementadas como serviços web, tanto usando a abordagem SOAP quanto a abordagem REST, ou como bibliotecas de desenvolvimento;

3. ***Aplicação***. A camada de aplicação é responsável por agrupar módulos que contenham funcionalidades relacionadas à lógica do negócio. O foco desta camada é orquestrar chamadas a diferentes métodos de diferentes classes a fim de resolver problemas complexos do negócio;

4. ***Domínio***. A camada de domínio é considerada o núcleo da solução. A camada de domínio segue uma estratégia de desenvolvimento orientado a interfaces. As interfaces de domínio garantem que as regras e responsabilidades sejam distribuídas de forma clara e bem definidas para as demais camadas da solução. Neste sentido, as responsabilidades entre os objetos e as camadas da solução ficam mais claras e mais bem separadas. A camada de domínio e as suas interfaces são responsáveis por ditar todas as regras e as funcionalidades que são implementadas pelas demais classes e camadas da solução.

   O desenvolvimento orientado a interfaces também facilita a substituição de uma implementação por outra, como é o caso de substituição de tecnologias, linguagens de programação e bibliotecas diferentes das utilizadas na implementação original. Por exemplo, a substituição de uma camada de acesso a dados o banco de dados Oracle (ORACLE, 2019) por uma nova implementação utilizando o banco de dados SQL Server (MICROSOFT, 2019i). A substituição se torna mais fácil pois não é necessário preocupar-se com as interações e dependências desta implementação em relação às demais camadas, visto que todos os objetos da solução estão seguindo implementações de interfaces definidas pela camada de domínio. Portanto, a substituição não gera impacto algum nas demais camadas da solução.

   Finalmente, a camada de domínio é responsável por manter todas as entidades de negócio utilizadas na solução. Estas entidades, chamadas de entidades de domínio, são o fundamento para a construção de um modelo orientado ao negócio. Por meio delas, pode-se identificar comportamentos e funcionalidades primordiais a fim de resolver problemas complexos existentes no domínio de negócio;

5. ***Infraestrutura***. A camada de infraestrutura é responsável por agrupar módulos cujo foco está na implementação de funcionalidades que são dependentes da infraes-

trutura e não do negócio (domínio) em si. Por exemplo, o uso do sistema gerenciador de banco de dados (SGBD) SQL Server (MICROSOFT, 2019i) por meio do ORM *Entity Framework* (EF) (MICROSOFT, 2019d). Outro exemplo seria o uso de um SGBD Oracle (ORACLE, 2019) por meio do *Object Relational Mapper* (ORM) Dapper (DAPPER, 2019). Neste sentido, para o negócio (domínio), não importa qual SGBD ou qual ORM está sendo utilizado. Ambos são apenas componentes de infraestrutura a fim de suportar o negócio (domínio) propriamente dito.

3

Notação Visual para SAWSDL

Linguagens de modelagem são comumente utilizadas em uma organização para modelar desde regras de negócio até artefatos de *software*. Diferentes princípios propostos pela Física das Notações (MOODY, 2009) podem ser utilizados para criar representações visuais de um dado domínio de forma mais eficaz, contribuindo, portanto, para a criação de modelos de mais fácil compreensão.

Este capítulo apresenta uma proposta de notação visual para o desenvolvimento de serviços web semânticos utilizando o padrão SAWSDL. O capítulo está estruturado da seguinte forma: a seção 3.1 apresenta a notação visual proposta para a representação de elementos de uma especificação WSDL; a seção 3.2 apresenta a notação visual proposta para a representação de classes de ontologias OWL; a seção 3.3 apresenta a notação visual proposta para anotações semânticas segundo o padrão SAWSDL; e, por fim, a seção 3.4 avalia a adequação entre as notações visuais propostas e os princípios da Física das Notações.

3.1 Notação Visual para WSDL

Uma especificação WSDL pode ser representada graficamente por meio de uma estrutura de grafo. Esta estrutura facilita a compreensão dos diferentes tipos de elementos de uma especificação WSDL, bem como suas relações hierárquicas.

Os tipos de elemento de uma especificação WSDL passíveis de serem representados visualmente em uma estrutura de grafo são: *wsdl:interface*, *wsdl:operation*, *wsdl:input*, *wsdl:output*, *wsdl:infault*, *wsdl:outfault*, *wsdl:fault*, *xs:complexType* e, por fim, *xs:simpleType*. Cada elemento é representado por um nó no grafo. Estes elementos são hierarquicamente dispostos no grafo em um formato de árvore, partindo do elemento mais genérico, *wsdl:interface*, no topo da árvore, até o elemento mais específico, *xs:simpleType*, na base da árvore.

O grafo de uma especificação WSDL é composto por cinco níveis. Os primeiros três níveis do grafo são utilizados para representar a estrutura de interfaces, de operações, de mensagens de entrada, de saída e de falhas de uma especificação WSDL, enquanto que os dois últimos níveis são utilizados para representar os tipos complexos e os tipos simples de dados XSD.

As arestas que ligam elementos (nós) do grafo são representadas por setas. Uma seta origina-se sempre em um elemento mais específico e destina-se a um elemento mais genérico. Tal representação indica a existência de uma relação entre os elementos conectados. Assim, partindo da base da árvore, elementos *xs:simpleType* podem ter arestas destinadas tanto a elementos *xs:complexType* quanto a elementos *wsdl:input*, *wsdl:output* ou *wsdl:fault*. Elementos *xs:complexType* podem ter arestas destinadas tanto a elementos *wsdl:input*, *wsdl:output* e *wsdl:fault* quanto a elementos *xs:complexType* mais genéricos. Elementos *wsdl:fault* podem ter arestas destinadas tanto a elementos *wsdl:infault*, quanto a elementos *wsdl:outfault*. Elementos *wsdl:input*, *wsdl:output*, *wsdl:infault* e *wsdl:outfault* que, por sua vez, podem compôr uma operação, podem ter arestas destinadas, portanto, a elementos *wsdl:operation*. Finalmente, elementos *wsdl:operation* compõem uma interface, resultando em arestas destinadas a *wsdl:interface*, no topo da árvore.

A Figura 13 ilustra as representações visuais para cada elemento WSDL e XSD, juntamente com os níveis da estrutura hierárquica do formato de árvore. Na lateral esquerda, podemos visualizar as representações visuais dos elementos do grafo juntamente com suas cardinalidades. No centro, podemos visualizar os tipos de elementos WSDL e XSD relacionados às suas representações visuais. Por fim, na lateral direita, podemos visualizar os níveis que definem a estrutura hierárquica do grafo.

O quinto nível do grafo (nível 5) consiste da representação visual de elementos XSD do tipo *xs:simpleType*. Estes elementos podem ser utilizados tanto para compor elementos *xs:complexType* quanto para compor elementos *wsdl:input*, *wsdl:output* e/ou *wsdl:fault*. A representação visual de elementos *xs:simpleType* possui o formato geométrico de um quadrado na cor laranja. Dado que estes elementos consistem do último nível do grafo, pode haver arestas partindo destes elementos e destinadas tanto a elementos *xs:complexType* (nível 4) quanto a elementos *wsdl:input*, *wsdl:output* e/ou *wsdl:fault* (nível 3). A cardinalidade dos elementos do quinto nível em relação aos elementos do quarto nível é de N para M (N:M), onde N (nível 5) pode variar entre zero ou várias instâncias (0..*) e M (nível 4) pode variar entre uma ou várias instâncias (1..*). A cardinalidade dos elementos do quinto nível em relação aos elementos do terceiro nível é de N para M (N:M), onde N (nível 5) pode variar entre zero ou várias instâncias (0..*) e M (nível 3) pode variar entre uma ou várias instâncias (1..*). Com isso, pode haver múltiplos elementos no quinto nível relacionados a múltiplos elementos do quarto nível e no terceiro nível. Entretanto, o nível 5 pode ser opcional, pois elementos *xs:complexType* não necessariamente precisam

Figura 13 – Representações visuais de elementos WSDL.

Fonte: Autoria própria.

ser estruturados em termos de elementos *xs:simpleType*.

O quarto nível do grafo (nível 4) consiste da representação visual de elementos XSD do tipo *xs:complexType*. Esta representação visual também possui o formato geométrico de um quadrado. Porém, *xs:complexType* possui um tamanho maior em relação aos elementos *xs:simpleType* do nível anterior (nível 5). Adicionalmente, a cor cinza é utilizada para sua representação. Dado que estes elementos consistem do quarto nível do grafo, pode haver arestas partindo destes elementos e destinadas a elementos *wsdl:input*, *wsdl:output* e/ou *wsdl:fault*, no nível 3. Adicionalmente, dado que um elemento *xs:complexType* pode ser composto tanto por outros elementos *xs:complexType* quanto por elementos *xs:simpleType*, podemos ter um número variável de composições, resultando em tantos sub-níveis quantos forem necessários a fim de representar tais composições. Com isso, o nível 4 do grafo poder possuir múltiplos sub-níveis. A cardinalidade dos elementos do quarto nível em relação aos elementos do terceiro nível é de N para M (N:M), onde N (nível 4) pode variar entre uma ou várias instâncias (1..*) e M (nível 3) pode variar entre uma ou várias instâncias (1..*). Assim, pode haver múltiplos elementos no quarto nível relacionados a múltiplos elementos do terceiro nível. Entretando, o nível 4 pode ser opcional desde que o nível 5 esteja presente, pois *wsdl:input*, *wsdl:output* e *wsdl:fault* podem ser exclusivamente compostos por elementos *xs:simpleType*.

O terceiro nível do grafo (nível 3) consiste da representação visual das mensagens de entrada e saída, juntamente com as mensagens de falhas e as definições de exceções tratadas por estas falhas. Com isso, no nível 3, podemos representar os elementos WSDL

dos tipos *wsdl:input*, *wsdl:output*, *wsdl:infault*, *wsdl:outfault* e, finalmente, *wsdl:fault*. Estas representações visuais possuem o formato geométrico de um hexágono irregular. Porém, estes elementos se diferem pela cor: a representação de *wsdl:input* possui a cor verde; a representação de *wsdl:output* possui a cor rosa; por fim, a representação dos elementos *wsdl:infault*, *wsdl:outfault* e *wsdl:fault* possui a cor marrom. Os cinco elementos possuem o mesmo tamanho.

Para os elementos *wsdl:input*, *wsdl:output*, *wsdl:infault* e *wsdl:outfault*, há arestas partindo destes elementos e destinadas a elementos *wsdl:operation*, no próximo nível (nível 2). Estas arestas indicam que *wsdl:input*, *wsdl:output*, *wsdl:infault* e *wsdl:outfault* compõem um elemento *wsdl:operation*. Para elementos *wsdl:fault*, pode haver arestas destinadas tanto a elementos *wsdl:infault* quanto a *wsdl:outfault*. Estas arestas indicam que *wsdl:fault* pode compor tanto *wsdl:infault* quanto *wsdl:outfault*. A cardinalidade de elementos *wsdl:input*, *wsdl:output*, *wsdl:infault* e *wsdl:outfault*, do terceiro nível, em relação a elementos *wsdl:operation*, do segundo nível, é de N para 1 (N:1), onde N (nível 3) pode variar entre zero ou várias instâncias (0..*). A cardinalidade de elementos *wsdl:fault* em relação a elementos *wsdl:infault* e *wsdl:outfault*, todos no mesmo nível (nível 3), é de 1 para N (1:N). O nível 3 está sempre presente na estrutura hierárquica de uma especificação WSDL.

O segundo nível do grafo (nível 2) consiste da representação visual de elementos WSDL do tipo *wsdl:operation*. Estas representações visuais também possuem o formato geométrico de um hexágono irregular. Porém, estes elementos são representados na cor ciano e em um tamanho maior em comparação aos elementos do nível anterior (nível 3). Dado que estes elementos consistem do segundo nível do grafo, há arestas originadas nestes elementos e destinadas a um elemento *wsdl:interface*, do próximo nível (nível 1). Estas arestas indicam que elementos *operation* compõem uma *wsdl:interface*. A cardinalidade dos elementos do segundo nível em relação aos elementos do primeiro nível é de N para 1 (N:1), onde N (nível 2) pode variar entre uma ou mais instâncias (1..*). Com isso, pode haver múltiplos elementos no segundo nível, mas eles só podem estar relacionados a um único elemento do primeiro nível. O nível 2 está sempre presente na estrutura hierárquica de uma especificação WSDL.

Por fim, o primeiro nível do grafo (nível 1), topo da árvore, consiste da representação visual de elementos WSDL to tipo *wsdl:interface*. Estas representações visuais possuem o formato geométrico de um hexágono irregular na cor azul. Dado que estes elementos consistem do primeiro nível da árvore, não há arestas originadas deles. O nível 1 também está sempre presente na estrutura hierárquica do grafo WSDL.

Além das diferentes cores, tamanhos e formas, os elementos do grafo são também identificados por textos (estereótipos) posicionados no canto superior esquerdo em relação ao nó do grafo. Para os elementos *wsdl:interface*, o estereótipo `«Interface»` é utilizado.

Para os elementos *wsdl:operation*, o estereótipo «Operation» é utilizado. Para os elementos *wsdl:input*, o estereótipo «Input» é utilizado. Para os elementos *wsdl:output*, o estereótipo «Output» é utilizado. Para os elementos *wsdl:infault*, o estereótipo «InFault» é utilizado. Para os elementos *wsdl:outfault*, o estereótipo «OutFault» é utilizado. Para os elementos *wsdl:fault*, o estereótipo «Fault» é utilizado. Para os elementos *xs:complexType*, o estereótipo «Complex-Type» é utilizado. Por fim, para os elementos *xs:simpleType*, o estereótipo «Simple-Type» é utilizado.

A Figura 14 ilustra o uso da notação proposta para a representação de diferentes elementos da especificação WSDL. Uma inferface `BookStoreService` é composta pela operação `GetBook`. Esta operação, por sua vez, é composta pelas mensagens `GetBookRequest` (entrada), `GetBookResponse` (saída) e `BookNotFoundException` (falha de saída). A mensagem de entrada, `GetBookRequest`, utiliza o tipo de dado complexo `Book`, que por sua vez é composto pelo tipos simples `BookName` e `BookId`. A mensagem de saída, `GetBookResponse`, utiliza o tipo de dado complexo `ArrayOfBooks`. `ArrayOfBooks` por sua vez utiliza o tipo complexo `Book`, o qual é composto pelos tipos simples `BookName` e `BookId`. Por fim, a mensagem de falha de saída `BookNotFoundException` é composta pela falha `BookNotFoundException`. Esta falha utiliza o tipo complexo `BookNotFound`, o qual é composto pelo tipo simples `BookId`.

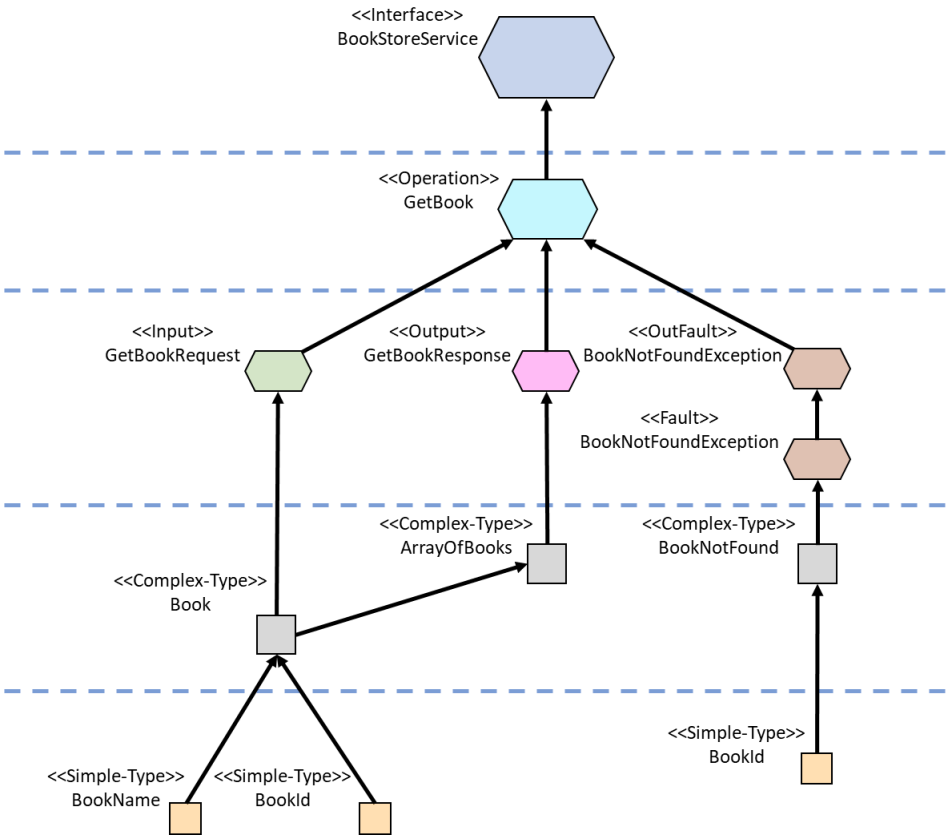
3.2 Notação Visual para OWL

As classes que compõem uma ontologia OWL também podem ser representadas graficamente por meio de uma estrutura de grafo. Esta estrutura facilita a compreensão de diferentes classes de uma ontologia, bem como suas relações hierárquicas.

Uma classe OWL é representada por meio de um círculo na cor vermelha no contexto deste trabalho. Dado o intuito de representar visualmente anotações semânticas segundo o padrão SAWSDL, nem todas as classes de uma ontologia OWL precisam ser graficamente representadas em um grafo, apenas as classes diretamente utilizadas em uma anotação (atributo do tipo *sawsdl:modelReference*).

Uma classe OWL pode especializar uma classe base. A especialização entre classes pode ser visualmente representada por meio de uma seta. Uma seta origina-se sempre de uma classe mais específica e destina-se a uma classe mais genérica (classe base). Os elementos do grafo da ontologia são também identificados por textos (estereótipos) posicionados no canto superior esquerdo em relação ao nó do grafo. Estes estereótipos contém o nome da ontologia da qual o conceito (classe OWL) em uso faz parte. A Figura 15 ilustra um grafo OWL representando três classes. A classe `Classe_C` é uma especialização da classe `Classe_B`, que por sua vez é uma especialização da classe `Classe_A`. Todas estas

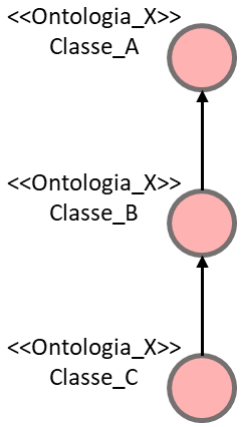
Figura 14 – Representações visuais de elementos WSDL juntamente com seus estereótipos.



Fonte: Autoria própria.

classes fazem parte de uma mesma ontologia `Ontologia_X`.

Figura 15 – Representação visual em formato de grafo para classes OWL.



Fonte: Autoria própria.

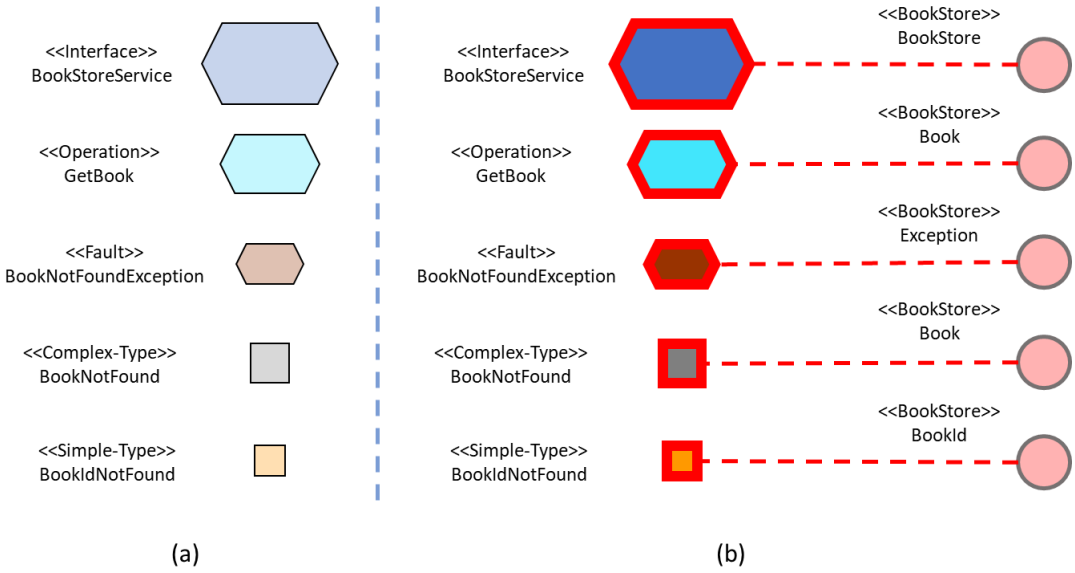
Por fim, caso haja classes hierarquicamente dispostas entre duas outras classes OWL que são utilizadas em anotações semânticas e, consequentemente, já representadas visualmente no grafo, estas classes intermediárias são também visualmente representadas.

Por exemplo, na Figura 15, caso as classes utilizadas em anotações semânticas sejam apenas as classes `Classe_A` e `Classe_C`, a classe `Classe_B` será automaticamente representada visualmente, visto que ela encontra-se disposta hierarquicamente entre as duas outras classes. A representação visual de classes intermediárias independe da quantidade das mesmas.

3.3 Notação Visual para Atributos SAWSDL

Uma especificação SAWSDL é essencialmente obtida por meio da junção dos grafos WSDL e OWL. Elementos WSDL/XSD anotados, independentemente do tipo de atributo SAWSDL (*Model Reference* ou *Schema Mapping*), possuem um brilho mais forte em relação aos elementos não anotados. Entretanto, quando anotados com o atributo *Model Reference*, os elementos possuem uma borda grossa e vermelha. Tal característica facilita a identificação de elementos já anotados com *Model Reference* em comparação a elementos não anotados. Adicionalmente, os elementos anotados com *Model Reference* possuem uma aresta (linha tracejada na cor vermelha) ligando-os a uma ou mais classes OWL. A Figura 16a ilustra elementos WSDL/XSD não anotados com *Model Reference*, enquanto que a Figura 16b ilustra elementos WSDL/XSD anotados.

Figura 16 – Notação visual para elementos WSDL/XSD anotados com *Model Reference*. (a) Elementos WSDL/XSD não anotados. (b) Elementos WSDL/XSD anotados com classes OWL.



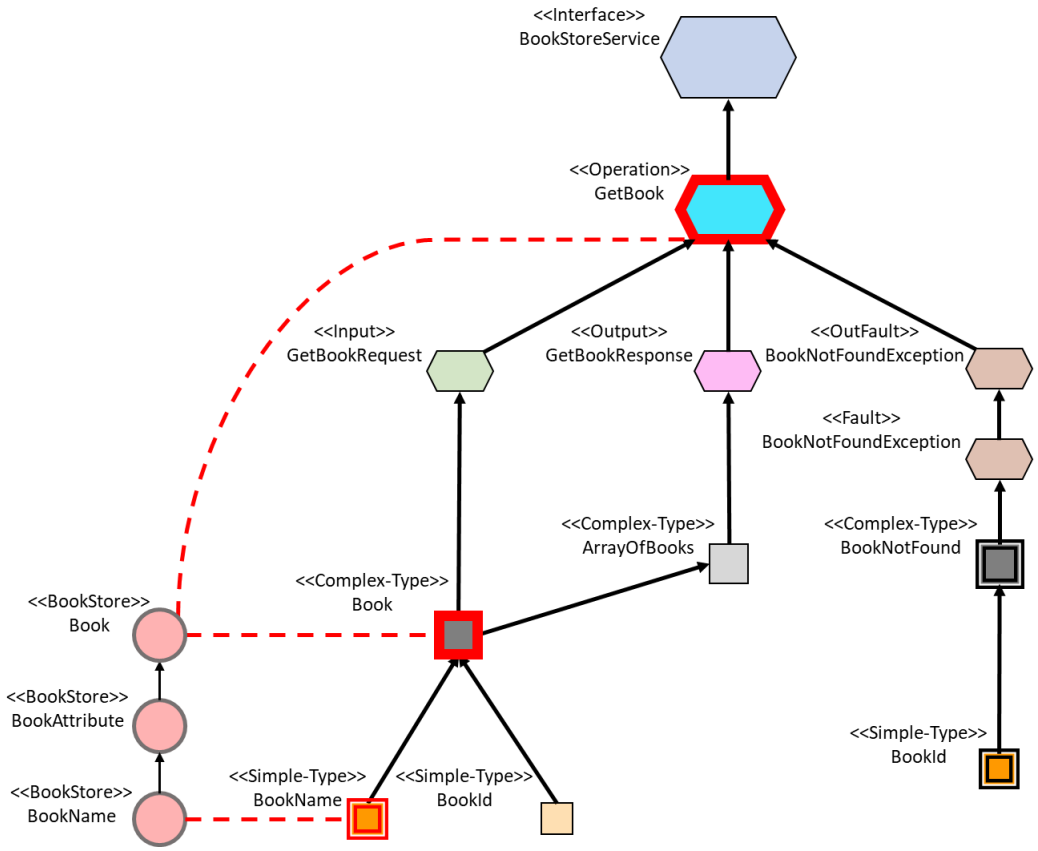
Fonte: Autoria própria.

Quando os elementos WSDL/XSD são anotados com o atributo *Schema Mapping*, estes elementos possuem uma borda dupla. Tal característica facilita a identificação de

elementos já anotados com *Schema Mapping* em comparação a elementos não anotados.

A Figura 17 ilustra um grafo WSDL com diferentes anotações SAWSDL. O tipo complexo `Book` e a operação `GetBook` ilustram elementos WSDL anotados apenas com o atributo *Model Reference*. O tipo complexo `BookNotFound` e o tipo simples `BookId` ilustram elementos WSDL anotados apenas com um atributo *Schema Mapping*. Finalmente, o tipo simples `BookName` ilustra um elemento WSDL anotado tanto com o atributo *Model Reference* quanto com um atributo *Schema Mapping*. Este elemento possui uma borda dupla na cor vermelha.

Figura 17 – Exemplo de grafo WSDL com anotações utilizando tanto *Model Reference* quanto *Schema Mapping*.



Fonte: Autoria própria.

3.4 Adequação aos princípios da Física das Notações

Dentre os nove princípios da Física das Notações, os princípios Clareza Semiótica, Diferenciação Perceptiva, Expressividade Visual, Codificação Dupla e Economia Gráfica foram

aplicados ao desenvolvimento da notação visual para SAWSDL. Os demais princípios não possuem aplicação para o propósito deste trabalho.

O princípio da Clareza Semiótica foi aplicado de modo que elementos de uma especificação WSDL e classes de uma ontologia possuam uma representação visual clara. Todos os elementos de uma especificação WSDL são visualmente representados com o propósito de facilitar o processo de anotação semântica. Os elementos *wsdl:input*, *wsdl:output*, *wsdl:infault* e *wsdl:outfault*, apesar de não serem passíveis de anotação semântica, são visualmente representados de modo a facilitar a compreensão da especificação WSDL. Em relação às classes OWL, classes que não são utilizadas em uma anotação semântica ou que não sejam hierarquicamente dispostas entre classes que são utilizadas em uma anotação semântica não são visualmente representadas.

O princípio da Diferenciação Perceptiva foi aplicado de modo a tornar as notações visuais propostas por este trabalho claramente distinguíveis umas das outras. Esta distinção ocorre por meio do uso (combinado) dos princípios da Expressividade Visual e da Codificação Dupla, ambos explicados a seguir. Desta forma, entende-se que os diferentes elementos visuais representados no grafo são facilmente identificados pelo usuário.

O princípio da Expressividade Visual foi satisfeito visto que das sete variáveis propostas por este princípio, seis foram utilizadas neste trabalho. A variável posição é aplicada entre os diferentes tipos de elementos WSDL e XSD, de modo a representar quais elementos são compostos por outros elementos da especificação WSDL. A variável tamanho é aplicada entre os diferentes tipos de elementos WSDL conforme seus níveis. A variável brilho foi utilizada a fim de diferenciar elementos WSDL/XSD anotados semanticamente dos elementos não anotados. A variável cor foi utilizada na representação de todos elementos do grafo, a fim de facilitar a diferenciação entre diferentes tipos de elementos da especificação WSDL, de elementos de classes OWL e, por fim, de anotações semânticas (arestas entre elementos WSDL/XSD e classes OWL). A variável orientação foi utilizada nos sentidos das arestas (setas). Entre elementos WSDL/XSD, a orientação da seta diferencia elementos que são compostos por outros elementos. Já entre elementos OWL, a orientação da seta diferencia elementos que especializam outros elementos. Finalmente, o uso da variável forma contribui para a diferenciação entre elementos WSDL (interfaces, operações e mensagens) e elementos XSD (tipos complexos e simples) de uma especificação WSDL e elementos OWL (classes de uma ontologia OWL). Apenas a variável textura não foi utilizada no contexto deste trabalho.

O princípio da Codificação Dupla foi aplicado por meio do uso de diferentes estereótipos juntamente com as representações visuais. Desta forma, a compreensão dos diferentes tipos de elementos visualmente representados no grafo é facilitada.

Finalmente, o princípio da Economia Gráfica foi satisfeito, pois utilizamos uma quantidade mínima de elementos visuais de modo a facilitar a compreensão de uma

especificação WSDL e de elementos envolvidos na anotação semântica segundo o padrão SAWSDL. Os tipos elementos que possuem representações visuais variam entre: *wsdl:interface*, *wsdl:operation*, *wsdl:input*, *wsdl:output*, *wsdl:infault*, *wsdl:outfault*, *wsdl:fault*, *xs:complexType* e *xs:simpleType* para elementos WSDL/XSD; e classes para elementos de uma ontologia OWL.

4

Ferramenta Grasews

Este capítulo apresenta a ferramenta *Graphical Annotation of Semantic Web Services* (Grasews) de suporte gráfico e colaborativo à anotação semântica de serviços web. O capítulo apresenta uma visão geral de sua arquitetura, principais tecnologias utilizadas em sua implementação e principais aspectos associados ao suporte à anotação semântica colaborativa de serviços web.

Este capítulo está estruturado da seguinte forma: a seção 4.1 apresenta a arquitetura da ferramenta Grasews; a seção 4.2 apresenta uma visão geral de sua implementação; a seção 4.3 descreve o suporte à anotação semântica provido pela ferramenta; e, por fim, a seção 4.4 apresenta o suporte à anotação semântica colaborativa.

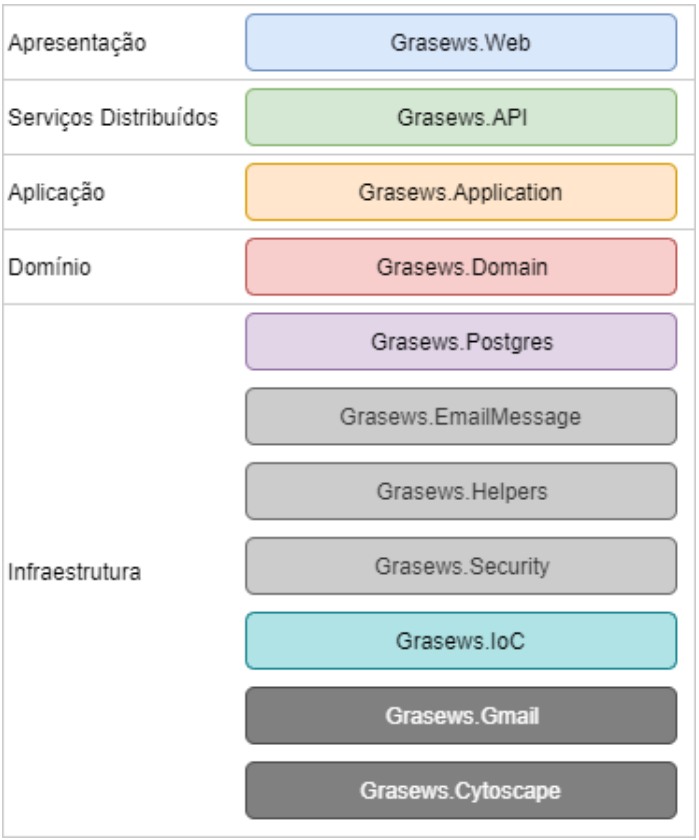
4.1 Arquitetura de Grasews

Esta seção apresenta uma visão geral acerca da arquitetura de desenvolvimento de Grasews, juntamente com uma visão detalhada dos módulos da arquitetura e as integrações destes módulos.

4.1.1 Visão Geral

Grasews foi desenvolvida segundo a abordagem *Domain-Driven Design* (DDD) (EVANS; FOWLER, 2004). De acordo com esta abordagem, Grasews foi estruturada em cinco (5) camadas: `Apresentação`, `Serviços Distribuídos`, `Aplicação`, `Domínio` e `Infraestrutura`. A camada de `Apresentação` consiste da camada de *front-end* de Grasews, enquanto que as demais camadas, `Serviços Distribuídos`, `Aplicação`, `Domínio` e `Infraestrutura`, consistem de camadas de *back-end* de Grasews. A Figura 18 apresenta os módulos existentes na arquitetura de Grasews distribuídos nas camadas da abordagem DDD.

Figura 18 – Camadas e módulos da arquitetura da ferramenta Grasews.



Fonte: Autoria própria.

A camada de `Apresentação` é composta pelo módulo `Grasews.Web`, que provê a interface gráfica de usuário (UI) da ferramenta. A camada de `Serviços Distribuídos` é composta pelo módulo `Grasews.API`, que provê uma API para expor as funcionalidades do *back-end* de Grasews. Esta API consiste de um conjunto de operações disponibilizadas por meio de serviços web RESTful. Todas as ações executadas por um usuário que necessitam interação com os demais módulos do lado servidor (*back-end*) da aplicação são exclusivamente realizadas através da API. A camada de `Aplicação` é composta pelo módulo `Grasews.Application`, responsável por orquestrar as regras de negócio da aplicação.

A camada de `Domínio` é composta pelo módulo `Grasews.Domain`, responsável por definir as regras de negócio da aplicação e as entidades de domínio do negócio. As regras de negócio são providas por meio de interfaces de desenvolvimento contidas neste módulo. Os demais módulos da aplicação seguem regras ditadas pelo módulo `Grasews.Domain`. A arquitetura de Grasews possui o seu desenvolvimento orientado a interfaces, facilitando a substituição de um módulo da aplicação por outro módulo que, por exemplo, utiliza outra tecnologia diferente. Por exemplo, caso Grasews utilize o SGBD *Postgres* e optemos por substituí-lo pelo SGBD *Microsoft SQL Server*(MICROSOFT, 2019i), esta substituição é facilitada, visto que existirá um módulo responsável pela conexão com o *Postgres* e

outro módulo responsável pela conexão com o *SQL Server*. Neste sentido, as classes dos demais módulos de Grasews que dependam do módulo do *Postgres* na verdade utilizam suas funcionalidades providas por meio de interfaces e não da implementação em si destas interfaces. Adicionalmente, por meio das interfaces de `Grasews.Domain`, garantimos que toda a aplicação funcione seguindo regras definidas pelo domínio (núcleo da aplicação).

A camada de `Infraestrutura` é composta pelos módulos `Grasews.Postgres`, `Grasews.EmailMessage`, `Grasews.Helpers`, `Grasews.Security`, `Grasews.IoC`, `Grasews.Gmail` e, por fim, `Grasews.Cytoscape`. O módulo `Grasews.Postgres` é responsável por conectar a aplicação ao banco de dados da aplicação. Grasews utiliza o SGBD Postgres (POSTGRES, 2019) para a persistência de dados. O módulo `Grasews.EmailMessage` provê suporte à construção de mensagens de *e-mail*. O módulo `Grasews.Helpers` provê suporte funcional a todas as demais camadas da aplicação. Tal suporte inclui funcionalidades de leitura de XML, funcionalidades de manipulação de enumeradores e funcionalidades auxiliares para a construção e manipulação de respostas de requisições HTTP, realizadas entre o lado cliente (*front-end*) e o lado servidor (*back-end*) por meio da API. O módulo `Grasews.Security` provê suporte às funcionalidades de segurança da aplicação, incluindo suporte à autenticação e ao controle de permissões de um usuário. Finalmente, o módulo `Grasews.IoC` provê suporte funcional à inversão de controle (*inversion of control* - IoC) e à injeção de dependências (*dependency injection* - DI).

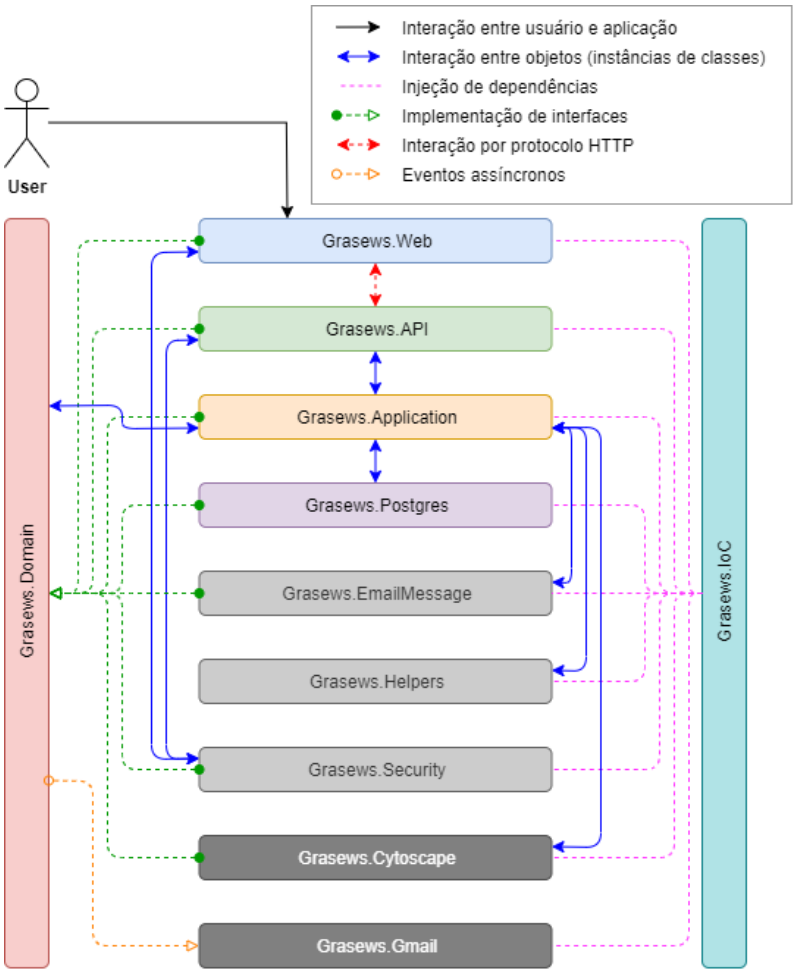
A inversão de controle é uma técnica utilizada para reduzir o acoplamento entre objetos (classes) de uma arquitetura de *software*. Ao invés de criar dependências entre classes por meio de suas implementações concretas, a inversão de controle quebra este acoplamento substituindo dependências de classes concretas por dependências de interfaces ou de classes abstratas. Assim, ao invés de depender de uma classe concreta, passamos a depender de uma abstração, contribuindo desta forma para o desacoplamento do código. A injeção de dependências representa uma abordagem para suporte à inversão de controle, garantindo então um baixo acoplamento em um conjunto de classes. O padrão de injeção de dependência consiste em uma maneira de informar a uma classe quais são as demais classes que serão consumidas por ela. A injeção de dependências de Grasews é feita por meio da injeção de objetos (dependências) por meio de construtores. Com isso, uma classe que necessita de instâncias de outras classes (dependências) obtém tais instâncias por meio da injeção destes objetos no construtor desta classe dependente.

Finalmente, o módulo `Grasews.Cytoscape` provê suporte à construção dos grafos WSDL e OWL da ferramenta. Este módulo utiliza a biblioteca *Cytoscape.js* (FRANZ et al., 2015), enquanto que o módulo `Grasews.Gmail` provê suporte ao envio de mensagens de *e-mail* aos usuários por meio do serviço do *Gmail* (GOOGLE, 2019).

4.1.2 Interações entre Módulos Grasews

A Figura 19 apresenta uma visão geral das principais interações existentes entre os módulos Grasews. Uma seta preta e sólida representa as interações de um usuário com a aplicação. Uma linha azul, sólida e com setas sólidas nas duas extremidades representa interações entre os módulos utilizando objetos (instâncias de classes). Uma linha rosa e tracejada representa injeções de dependências. Uma linha verde e tracejada representa implementações de interfaces. A extremidade com a seta verde e vazia indica a definição das interfaces contidas em `Grasews.Domain`, enquanto que a extremidade com o círculo verde e sólido indica um módulo que implementa uma interface definida em `Grasews.Domain`. Uma seta vermelha e tracejada indica interações realizadas por meio do protocolo HTTP entre os módulos da aplicação. Por fim, uma seta laranja e tracejada representa eventos assíncronos. A extremidade com o círculo laranja e vazio representa a origem dos eventos assíncronos, enquanto que a extremidade com a seta laranja e vazia indica o módulo onde os eventos assíncronos são notificados.

Figura 19 – Arquitetura da ferramenta Grasews.



Fonte: Autoria própria.

O fluxo resumido da aplicação inicia-se por meio da interação do usuário com o módulo `Grasews.Web`, o qual se comunica com `Grasews.API` por meio de requisições HTTP. Tanto `Grasews.Web` quanto `Grasews.API` utilizam funcionalidades providas por objetos de `Grasews.Security`. `Grasews.API` consome serviços providos por `Grasews.Application`, o qual orquestra o fluxo de trabalho conforme as regras de negócio da aplicação. Neste sentido, `Grasews.Application` é responsável por manipular objetos das entidades de domínio da aplicação, juntamente com funcionalidades de apoio providas por `Grasews.Helpers`. Adicionalmente, `Grasews.Application` utiliza funcionalidades providas por `Grasews.Cytoscape`, para a manipulação de elementos do grafo, e funcionalidades providas por `Grasews.EmailMessage`, para a construção de mensagens de *e-mail*. Juntamente com `Grasews.Gmail`, `Grasews.Application` invoca assíncronamente `Grasews.Domain` para que a mensagem de *e-mail*, previamente construída por `Grasews.EmailMessage`, seja enviada para um usuário. Finalmente, `Grasews.IoC` provê a injeção de dependências para todos os demais módulos da ferramenta.

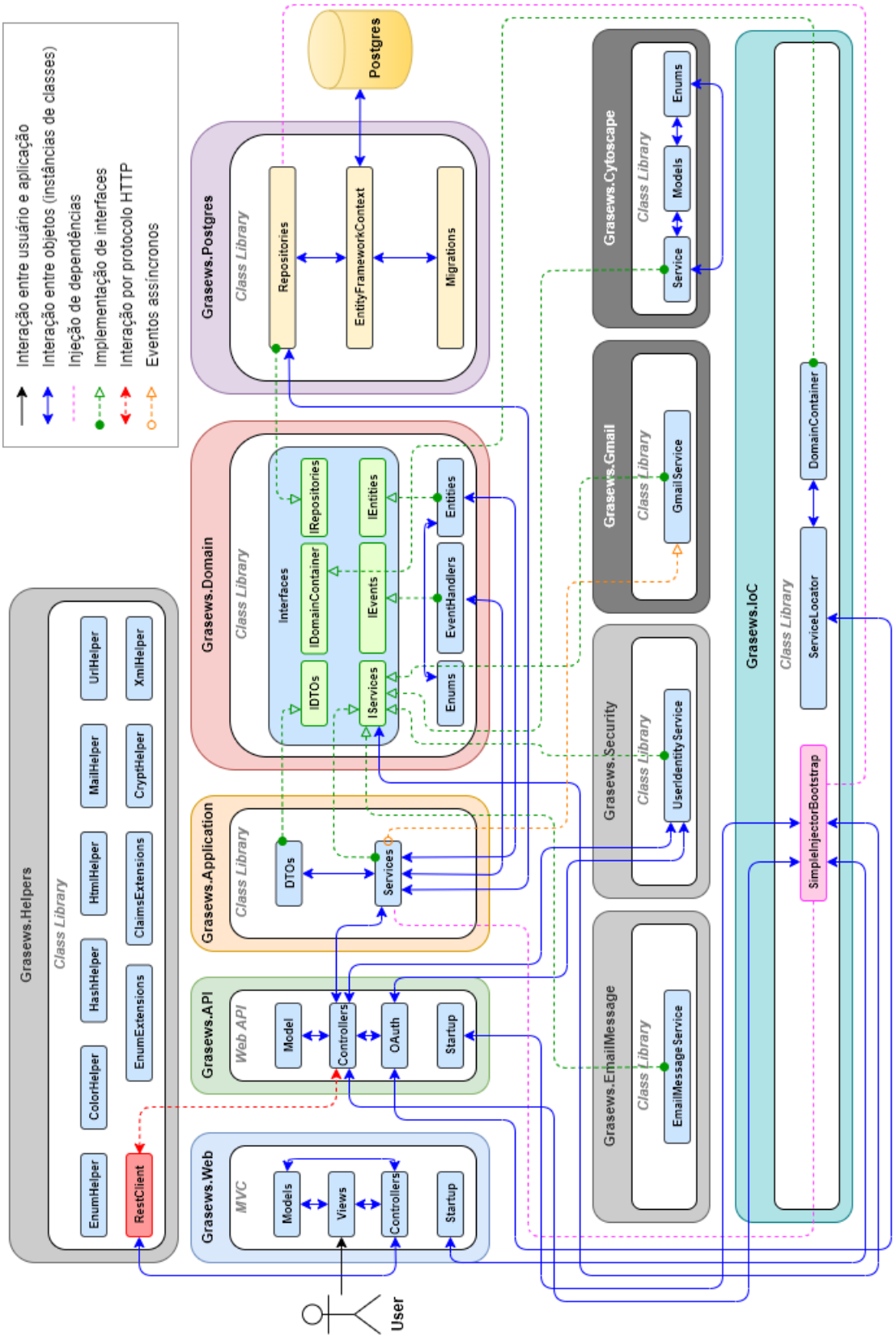
4.1.3 Interações Detalhadas entre Módulos Grasews

A Figura 20 detalha as principais interações dos módulos da arquitetura de Grasews. O fluxo da aplicação inicia-se com um usuário interagindo com a interface gráfica da aplicação por meio do módulo `Grasews.Web` (à esquerda, na cor azul). `Grasews.Web` foi desenvolvido utilizando o padrão arquitetônico MVC e, portanto, seus principais componentes são representados por `Models`, `Views` e `Controllers`. Um usuário interage com um componente `View` que, por sua vez, utiliza um objeto modelo definido por um componente `Model`. As ações de um usuário resultam em eventos controlados por um componente `Controller`.

O módulo `Grasews.IoC` é responsável pela injeção de dependências para todos os módulos da aplicação. Este módulo é composto por três componentes principais: `ServiceLocator`, `DomainContainer` e `SimpleInjectorBootstrap`. O componente `ServiceLocator` é responsável por prover instâncias de classes que ainda não foram introduzidas no contexto da aplicação. `ServiceLocator` é auxiliado pelo componente `DomainContainer`, que atua como um repositório de objetos da aplicação. Por fim, o componente `SimpleInjectorBootstrap` é responsável por criar um repositório de referências a objetos (classes) que serão necessários e utilizados por toda a aplicação. Estes objetos tornam-se disponíveis aos demais componentes da aplicação por meio da injeção de dependências realizada nos construtores destes demais componentes.

O componente `Startup`, do módulo `Grasews.Web`, é responsável por requisitar ao componente `SimpleInjectorBootstrap` o controle de dependências da aplicação. Por fim, um componente de `Controller` utiliza o componente `RestClient`, do módulo `Grasews.`

Figura 20 – Arquitetura detalhada da ferramenta Grasews.



Fonte: Autoria própria.

`Helper`, para que uma nova requisição seja feita à API da aplicação.

A API da aplicação é provida pelo módulo `Grasews.API` (centro à esquerda, na cor verde). Os principais componentes deste módulo são os `Controllers` e os `Models`. As requisições originadas de `Grasews.Web` são gerenciadas por um `Controller` da API. Tais requisições são repassadas aos serviços do módulo `Grasews.Application`. Adicionalmente, `Grasews.API` contém o componente `Startup`, que assim como no módulo `Grasews.Web`, este componente é responsável por requisitar o registro de dependências ao módulo `Grasews.IoC`.

Todas as requisições realizadas para `Grasews.API` são protegidas. Assim, para realizar uma ação na ferramenta, o usuário deve ser autenticado pelo componente `OAuth`. Este componente é responsável por validar as credenciais de um usuário, permitindo o uso das funcionalidades da ferramenta, tanto por meio do módulo `Grasews.Web` quanto do módulo `Grasews.API`. O componente `OAuth` utiliza recursos providos por `UserIdentityService`, do módulo `Grasews.Security` (centro inferior, na cor cinza claro).

O módulo `Grasews.Application` é representado simplificadamente por meio dos componentes `Services` e `DTOs` (*Data Transfer Objects*). As classes que compõem os serviços (`Services`) são responsáveis por orquestrar todas as requisições originadas do módulo `Grasews.API`. Requisições realizadas ao módulo `Grasews.Application` resultam na utilização de objetos das camadas inferiores, como, por exemplo, a criação de instâncias de entidades de domínio, de `Grasews.Domain`, ou o consumo de métodos de classes de acesso a dados, denominados repositórios, de `Grasews.Postgres`. Os parâmetros de entrada e de saída de serviços do módulo `Grasews.Application` variam entre tipos simples, e.g., *int* e *string*, instâncias de entidades de domínio ou então instâncias de um `DTO`. Nenhum objeto de um tipo de `Model` de `Grasews.API` é transferido para `Grasews.Application`. Os `Controllers` de `Grasews.API` são responsáveis por converter seus objetos de tipos de `Models` para objetos de tipo de uma entidade de domínio ou então de tipo de um `DTO`. Desta forma, obtém-se uma melhor segregação do escopo de objetos de cada módulo e, consequentemente, um maior desacoplamento entre os módulos, tornando um módulo o mais autossuficiente possível.

O módulo `Grasews.Domain` (centro, na cor salmão) contém todas as interfaces (na cor verde claro) de desenvolvimento que são implementadas pelos demais módulos da aplicação. Adicionalmente, `Grasews.Domain` é responsável por definir as entidades do domínio de negócio da aplicação, representadas pelo componente `Entities` (na cor azul). Uma entidade geralmente representa uma tabela no banco de dados. Por meio do módulo `Grasews.Postgres`, estas entidades e suas propriedades são mapeadas para tabelas e colunas do banco de dados, respectivamente. Por fim, `Grasews.Domain` possui componentes responsáveis por controlar eventos assíncronos. Eventos assíncronos são

funcionalidades que podem ser executadas independentemente do fluxo da aplicação, pois não geram dependências para a aplicação. Por exemplo, o envio de um *e-mail* para um usuário pode ser executado de forma assíncrona. Os componentes responsáveis pela implementação do controle de eventos assíncronos são os `EventHandlers` e as interfaces de `IEvents`.

O módulo `Grasews.Postgres` (centro à direita, na cor lilás) é responsável por realizar o acesso da aplicação ao banco de dados. Seus principais componentes são os repositórios (`Repositories`) e o contexto do banco de dados representado por `EntityFrameworkContext`. Todas as ações executadas em Grasews que necessitam acesso ao banco de dados são realizadas pelos repositórios. Os repositórios implementam interfaces definidas no módulo `Grasews.Domain`. Grasews utiliza o SGBD *Postgres* (centro à direita, na cor amarelo). Adicionalmente, toda a estrutura de banco de dados de Grasews foi gerada automaticamente por meio do componente `Migrations`, com base nas classes de entidades de domínio existentes no módulo `Grasews.Domain`. `Migrations` permite que alterações no modelo do código sejam feitas e depois propagadas para o banco de dados.

O módulo `Grasews.Helpers` (canto superior, na cor cinza claro) possui componentes com funcionalidades que dão suporte aos demais módulos de Grasews. Por exemplo, `Grasews.Helpers` provê o componente `RestClient` para melhor controle e padronização de requisições HTTP entre `Grasews.Web` e `Grasews.API`. Por fim, o módulo `Grasews.Cytoscape` (canto inferior direito, na cor cinza escuro) é responsável por manipular nós e arestas do grafo de Grasews. Este módulo foi desenvolvido baseado nas funcionalidades e modelos necessários da biblioteca de desenvolvimento `Cytoscape.js`, utilizada pelo módulo `Grasews.Web` para o desenvolvimento dos grafos na interface gráfica de usuário da ferramenta. Os principais componentes de `Grasews.Cytoscape` são `Service`, contendo os métodos para manipulação dos elementos (modelos) do grafo, e `Models`, contendo os tipos de dados necessários para a representação de elementos no grafo, como, por exemplo, nós, arestas e propriedades visuais.

4.2 Visão Geral da Implementação

A interface gráfica do usuário (*front-end*) foi desenvolvida seguindo a arquitetura MVC provido pelo *ASP.NET MVC* (MICROSOFT, 2019e). Neste módulo, a biblioteca *jQuery* (THE JQUERY FOUNDATION, 2019) foi intensivamente utilizada em conjunto com a linguagem *JavaScript* para o desenvolvimento de funcionalidades do lado cliente. Adicionalmente, as linguagens HTML5 e CSS3 foram utilizadas para a formatação da interface gráfica de usuário de Grasews. Ambas linguagens são suportadas por *templates* providos por *Admin LTE* (COLORLIB, 2019), por funcionalidades providas por *Bootstrap* (BOOTSTRAP, 2019) e por ícones providos por *Font-Awesome* (FONT-AWESOME, 2019).

O formato padrão de comunicação entre o *front-end* e o *back-end* é o formato JSON (JSON.ORG, 2019). Neste sentido, todas as mensagens utilizadas nas interações entre a interface de usuário e a API da aplicação utilizam objetos JSON. Adicionalmente, o formato JSON é utilizado na criação dos grafos.

A principal linguagem de programação utilizada no desenvolvimento da ferramenta Grasews foi a linguagem C#, do *framework* .NET (MICROSOFT, 2019g). A linguage C# foi utilizada no desenvolvimento de funcionalidades do lado servidor providas pelos módulos de *back-end* da aplicação. A API da aplicação foi desenvolvida como serviços web RESTful, suportados pelo *ASP.NET Web API* (MICROSOFT, 2019f). Adicionalmente, utilizamos o ORM *.NET Entity Framework* (MICROSOFT, 2019d) para facilitar o mapeamento de classes .NET para tabelas do banco de dados, bem como a criação da estrutura do banco de dados (*migrations*) e a persistência de dados da aplicação no *Postgres*.

As seguintes ferramentas foram utilizadas no desenvolvimento de Grasews:

- **Microsoft Visual Studio 2019**. *Microsoft Visual Studio* é um ambiente (IDE) de desenvolvimento para linguagens de programação do *framework* .NET, entre outras. *Visual Studio* provê fácil integração com *Microsoft Azure* e com *Microsoft Azure DevOps*, para publicação de aplicações e controle de versão do código-fonte, respectivamente. Todos os módulos de Grasews foram desenvolvidos utilizando o *Visual Studio*;

- **pgAdmin**. *pgAdmin* (PGADMIN, 2019) é a plataforma de administração e desenvolvimento mais popular e com mais recursos para o banco de dados *PostgreSQL*. Assim como *PostgreSQL*, *pgAdmin* é desenvolvido com o código aberto;

- **Postman**. *Postman* (POSTMAN, 2019) é uma plataforma de colaboração para o desenvolvimento de APIs. Os recursos do *Postman* simplificam cada etapa da criação de uma API e agilizam a colaboração para que você possa criar APIs mais rapidamente. Utilizamos *Postman* para validar as funcionalidades disponíveis em `Grasews.API`. Com isso, podemos validar tanto os endereços dos *endpoints*, quanto os parâmetros de entrada e as respostas dos *endpoints*;

- **Microsoft Azure Hosting**. *Microsoft Azure* (MICROSOFT, 2019b) é um serviço de computação em nuvem criado por *Microsoft* para criar, testar, implantar e gerenciar aplicativos e serviços por meio de data centers gerenciados por *Microsoft*. Utilizamos *Microsoft Azure* para hospedar tanto `Grasews.Web`[1] quanto `Grasews.API`[2];

[1] Grasews Web disponível em <https://grasewsweb.azurewebsites.net/>.
[2] Grasews API disponível em <https://grasewsapi.azurewebsites.net/>.

- **Microsoft Azure DevOps**. *Microsoft Azure DevOps* (MICROSOFT, 2019c) é um repositório de código-fonte oferecido por *Microsoft*. Por meio de *Microsoft Azure DevOps*, desenvolvedores podem versionar o código-fonte, bem como controlar as entregas e automatizar tarefas como compilações e publicações de aplicações. O código-fonte de Grasews está disponível por meio da Documentação *Online* do Projeto[3].

A Figura 21 apresenta a pilha de tecnologias utilizada na arquitetura de desenvolvimento de Grasews. No topo, encontramos as tecnologias utilizadas para o desenvolvimento *front-end* de Grasews. Nesta camada, à esquerda encontramos as principais tecnologias utilizadas do *framework* .NET. Ao centro da camada de *front-end*, encontramos as principais tecnologias e bibliotecas utilizadas para o desenvolvimento com base na linguagem *JavaScript*. Por fim, acima das bibliotecas *JavaScript*, encontramos as principais tecnologias e bibliotecas utilizadas para formatação e apresentação de páginas web (HTML e CSS). Abaixo da camada de *front-end*, são representadas as principais tecnologias utilizadas para o desenvolvimento da camada de *back-end*. Nesta camada utilizamos exclusivamente tecnologias do *framework* .NET. A próxima camada é a camada de persistência de dados. Nesta camada, encontramos o uso do banco de dados Postgres. Por fim, a camada na base da pilha consiste da plataforma de hospedagem e controle de código-fonte, provida pela solução em nuvem *Microsoft Azure*.

O Apêndice A apresenta uma visão geral das principais tecnologias e bibliotecas de desenvolvimento utilizadas na implementação de Grasews.

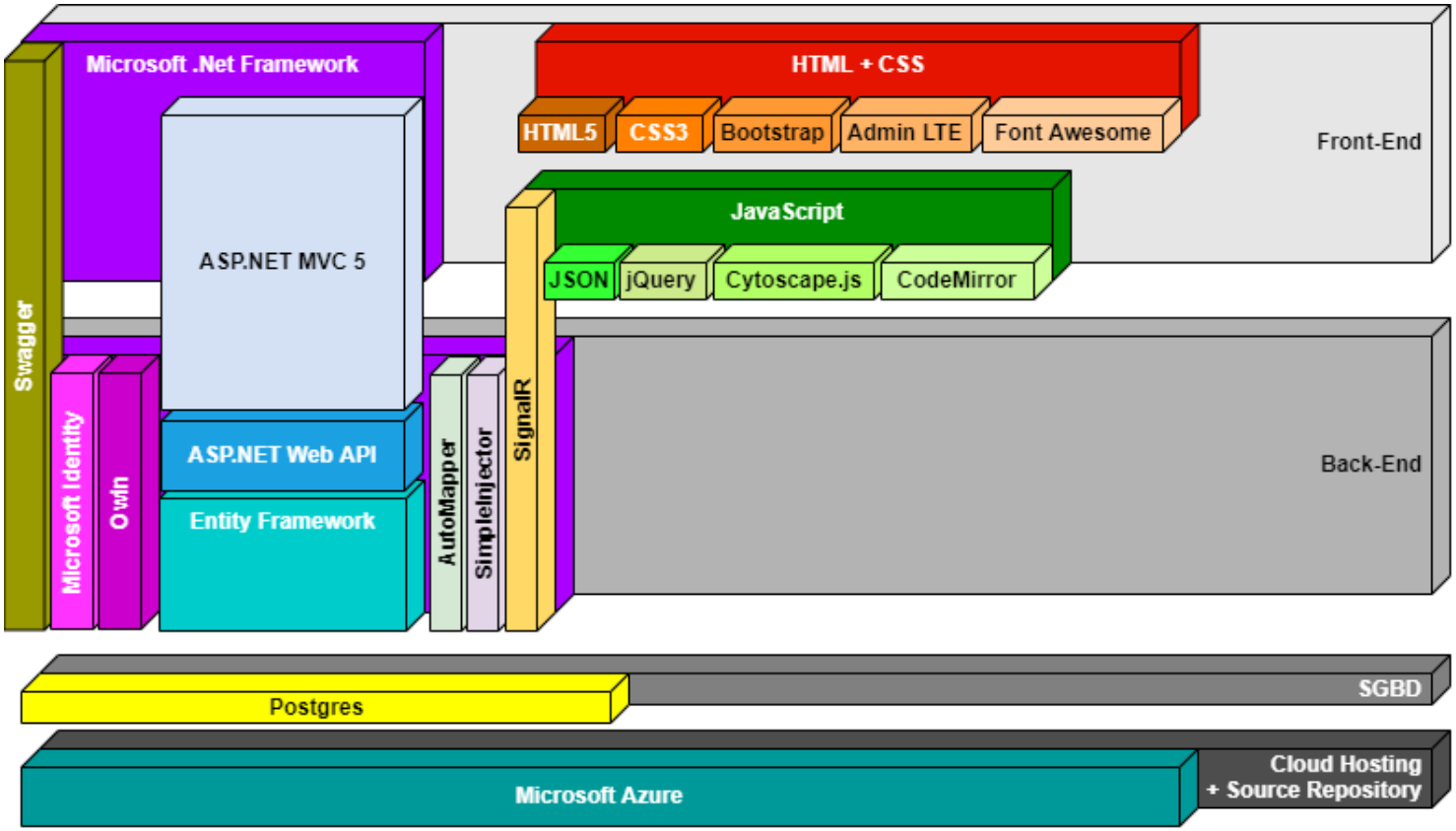
4.3 Suporte à Anotação Semântica

Grasews provê recursos para que o usuário seja capaz de anotar semanticamente uma especificação WSDL. Por meio de menus de contexto, o usuário é capaz de criar anotações utilizando tanto o atributo *Model Reference* quanto os atributos *Lifting Schema Mapping* e *Lowering Schema Mapping*, do padrão SAWSDL.

A Figura 22 ilustra a interface gráfica de Grasews, contendo os três principais componentes visuais de Grasews. Ao centro, encontra-se o painel de visualização tanto do grafo quanto do código WSDL/XML, apresentado na seção 4.3.1. À esquerda, encontra-se o painel de visualização de elementos de uma especificação WSDL no formato de menu de árvore (*tree-view*), apresentado na seção 4.3.2.1. Por fim, à direita, encontra-se o painel de visualização de classes de uma ontologia OWL também no formato de menu de árvore (*tree-view*), apresentado na seção 4.3.2.2.

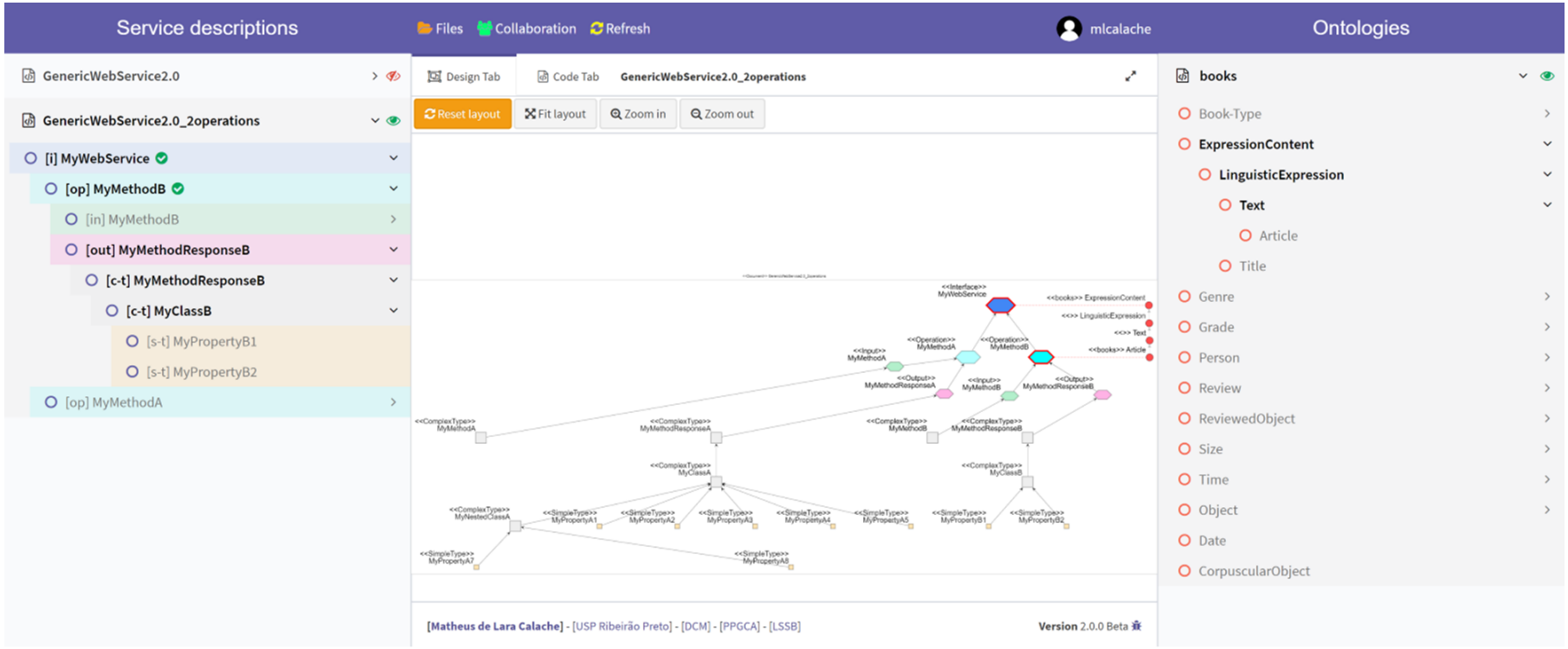
[3] Documentação *Online* do Projeto disponível em <https://grasewsweb.azurewebsites.net/projectdocumentation>.

Figura 21 – Pilha de tecnologias envolvidas no desenvolvimento de Grasews.



Fonte: Autoria própria.

Figura 22 – Interface gráfica de Grasews.



Fonte: Autoria própria.

4.3.1 Grafo

O principal componente visual de Grasews é o grafo. Grasews utiliza a notação visual proposta no Capítulo 3 para representar hierarquicamente elementos de uma especificação WSDL e classes de uma ontologia OWL, possibilitando que um usuário crie anotações semânticas segundo a abordagem SAWSDL. Para os elementos WSDL/XSD passíveis de serem anotados com *Model Reference*, Grasews disponibiliza um menu de contexto para adicionar e remover um URI de *Model Reference*. Para os elementos passíveis de serem anotados tanto com *Model Reference* quanto com *Lifting Schema Mapping* e *Lowering Schema Mapping*, Grasews estende o menu de contexto anterior com opções adicionais para permitir todos os tipos de anotações. A Figura 23 ilustra o menu de contexto disponível para um tipo complexo XSD `GetAllBooks` (parcialmente visível atrás do menu de contexto).

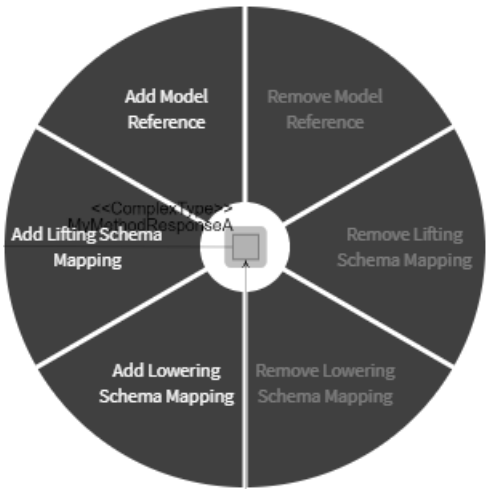
A Figura 24 apresenta parte do grafo de uma especificação WSDL anotada com *Model Reference*. Observa-se que a interface `MyWebService` está anotada com *Model Reference* utilizando a classe OWL `ExpressionContent`, pertencente à ontologia `books`. A operação `MyMethodB` está anotada com *Model Reference* utilizando a classe OWL `Article`, da mesma ontologia. Grasews constrói automaticamente a estrutura hierárquica entre as duas classes OWL em uso por anotações semânticas. Com isso, as classes OWL intermediárias `LinguisticExpression` e `Text` são automaticamente dispostas no grafo entre as duas classes OWL `ExpressionContent` e `Article`.

Adicionalmente, Grasews provê um painel para visualizar o código XML de uma especificação WSDL. A Figura 25 ilustra parte do painel de visualização do código WSDL/XML associado ao grafo representado na Figura 24. O painel é automaticamente atualizado conforme o trabalho (anotações semânticas) é realizado. A linha 70 representa a interface *MyWebService* anotada com o atributo *Model Reference* utilizando a classe OWL `ExpressionContent` da ontologia `books`. Por fim, a linha 75 representa a operação *MyMethodB* anotada com o atributo *Model Reference* utilizando a classe OWL `Article`, também da ontologia `books`. Com isso, caso o usuário tenha conhecimento técnico no código XML de uma especificação WSDL, este usuário pode verificar o painel de código a fim de validar as anotações semânticas criadas com o auxílio das notações visuais de Grasews. Entretanto, o painel de visualização de código não é editável.

4.3.2 Menu *tree-view*

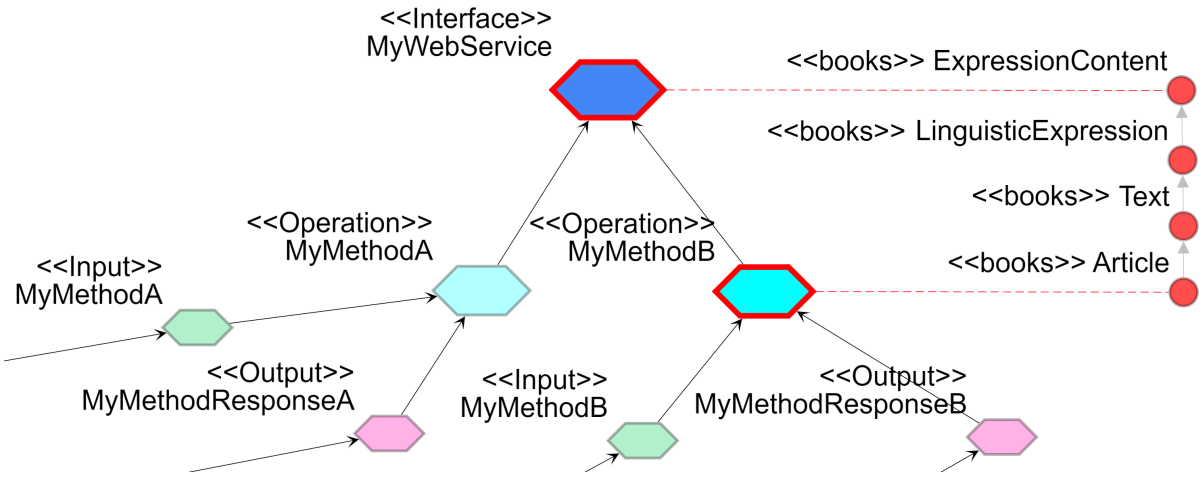
Grasews utiliza um menu no formato de árvore (*tree-view*) para auxiliar na compreensão tanto de uma especificação WSDL quanto de uma ontologia OWL, facilitando, consequentemente, o processo de anotação semântica.

Figura 23 – Menu de contexto para anotação semântica por meio de grafo do Grasews.



Fonte: Autoria própria.

Figura 24 – Elementos do grafo de Grasews anotados com *Model Reference*.



Fonte: Autoria própria.

4.3.2.1 Menu *tree-view* WSDL

De forma análoga ao grafo WSDL, o menu *tree-view* de uma especificação WSDL também é composto por cinco níveis. Cada nível está diretamente associado ao nível correspondente do grafo WSDL. A Figura 26 ilustra a estrutura hierárquica de elementos de uma especificação WSDL por meio do menu *tree-view*. Esta estrutura permite identificar as relações de composição entre os diferentes tipos de elementos.

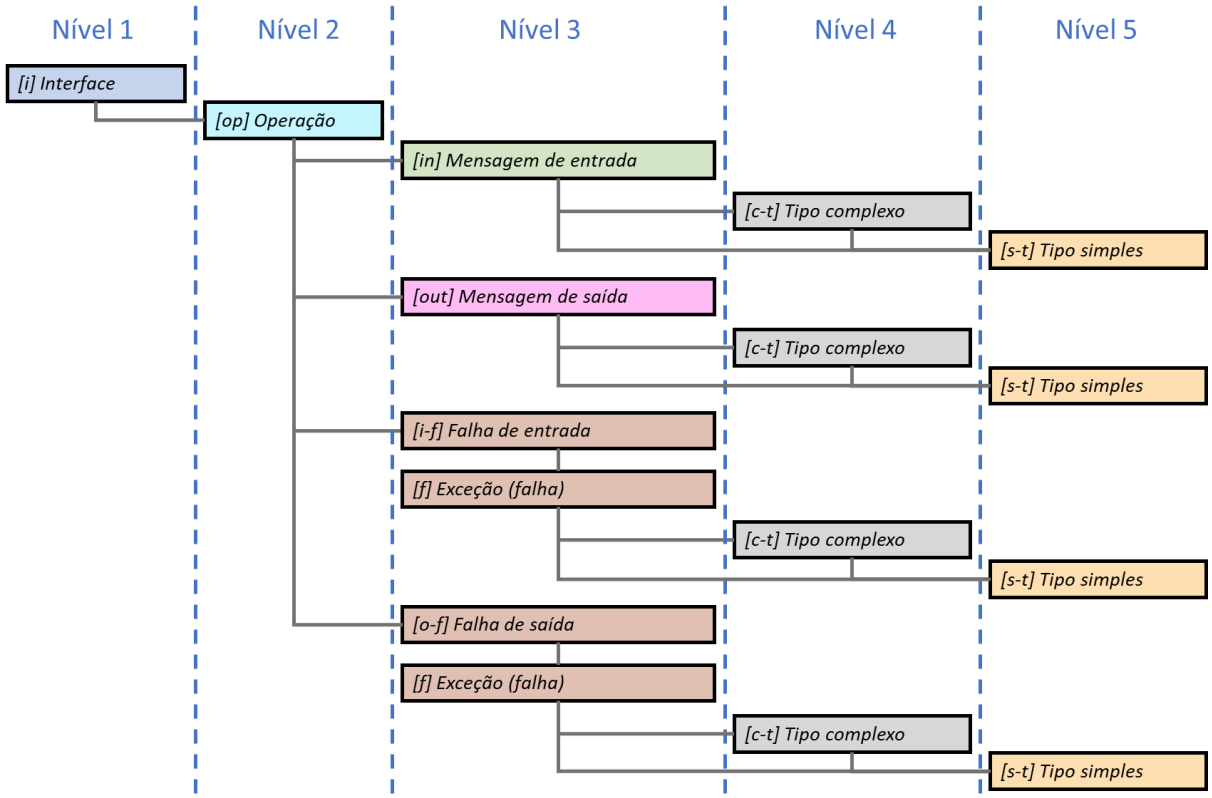
O primeiro nível (nível 1) consiste dos elementos do tipo *wsdl:interface*. Estes elementos estão representados na cor azul para facilitar a associação com elementos do grafo que representam uma interface, também na cor azul. Adicionalmente, um elemento

Figura 25 – Painel de Grasews para visualização do código WSDL/XML.

```
70   <interface name="MyWebService"
     sawsdl:modelReference="http://127.0.0.1:3001/ontology/books.owl#ExpressionContent">
71     <operation name="MyMethodA" pattern="http://www.w3.org/ns/wsdl/in-out">
72       <input element="tns:MyMethodA" />
73       <output element="tns:MyMethodResponseA" />
74     </operation>
75     <operation name="MyMethodB" pattern="http://www.w3.org/ns/wsdl/in-out"
     sawsdl:modelReference="http://127.0.0.1:3001/ontology/books.owl#Article">
76       <input element="tns:MyMethodB" />
77       <output element="tns:MyMethodResponseB" />
78     </operation>
79   </interface>
```

Fonte: Autoria própria.

Figura 26 – Estrutura da representação visual do menu *tree-view* para uma especificação WSDL.



Fonte: Autoria própria.

do tipo *wsdl:interface* possui o prefixo `[i]` no menu *tree-view*.

O segundo nível (nível 2) consiste dos elementos do tipo *wsdl:operation*. Estes elementos estão representados na cor ciano para facilitar a associação com elementos do grafo que representam uma operação, também na cor ciano. Adicionalmente, um elemento do tipo *wsdl:operation* possui o prefixo `[op]` no menu *tree-view*.

O terceiro nível (nível 3) consiste dos elementos dos tipos *wsdl:input*, *wsdl:output*, *wsdl:infault*, *wsdl:outfault* e, finalmente, *wsdl:fault*. Elementos *wsdl:input* são representados na cor verde para facilitar a associação com elementos do grafo que representam uma mensagem de entrada, também na cor verde. Adicionalmente, um elemento *wsdl:input* possui o prefixo `[in]` no menu. Elementos *wsdl:output* são representados na cor rosa para facilitar a associação com elementos do grafo que representam uma mensagem de saída, também na cor rosa. Adicionalmente, um elemento *wsdl:input* possui o prefixo `[out]` no menu. Por fim, elementos relacionados a falhas (exceções) são representados na cor marrom para facilitar a associação com elementos do grafo que representam uma mensagem de falha de entrada (*wsdl:infault*), uma mensagem de falha de saída (*wsdl:outfault*) ou, então, uma falha (*wsdl:fault*), todas na cor marrom. Adicionalmente, um elemento *wsdl:infault* possui o prefixo `[i-f]`, um elemento *wsdl:outfault* possui o prefixo `[o-f]` e, finalmente, um elemento *wsdl:fault* possui o prefixo `[f]`.

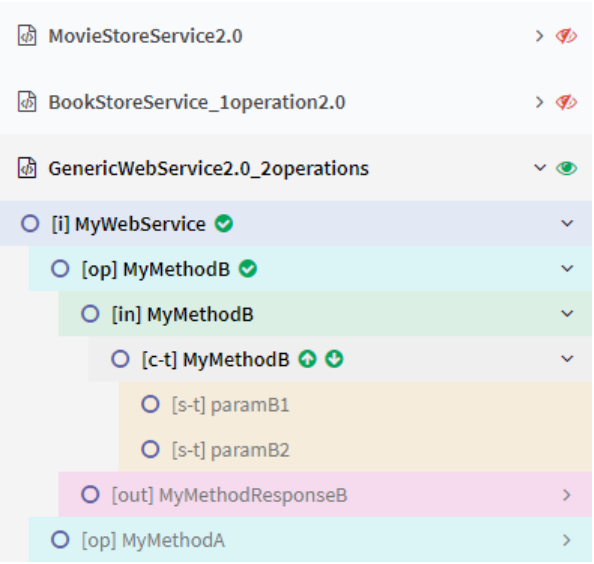
O quarto nível (nível 4) consiste dos elementos do tipo *xs:complexType*. Elementos *xs:complexType* são representados na cor cinza para facilitar a associação com elementos do grafo que representam um tipo complexo, também na cor cinza. Adicionalmente, um elemento *xs:complexType* possui o prefixo `[c-t]` no menu *tree-view*.

Finalmente, o quinto nível (nível 5) consiste dos elementos do tipo *xs:simpleType*. Elementos *xs:simpleType* são representados na cor laranja para facilitar a associação com elementos do grafo que representam um tipo simples, também na cor laranja. Adicionalmente, um elemento *xs:simpleType* possui o prefixo `[s-t]` no menu *tree-view*.

A Figura 27 ilustra o menu *tree-view* para uma especificação WSDL aberta na ferramenta Grasews. Inicialmente, ao abrir um documento WSDL na ferramenta, o menu *tree-view* possui apenas o elemento *wsdl:interface* visível. Por meio da navegação pelo menu *tree-view*, o usuário pode descobrir elementos da especificação WSDL. Por meio das setas na lateral direita do menu, o usuário pode visualizar sub-elementos do menu, i.e., visualizar os elementos WSDL/XSD que compõem hierarquicamente uma especificação WSDL. Uma seta orientada para a direita representa um item de menu que contém sub-itens, porém eles estão fechados, como, por exemplo, os itens de menu `[out] MyMethodResponseB` e `MyMethodA`. Uma seta orientada para baixo representa um item de menu que contém sub-itens abertos, como, por exemplo os itens de menu `[i] WebServiceService`, `[op] MyMethodB`, `[in] MyMethodB` e `[c-t] MyMethodB`. Por fim, quando um item de menu não possui sub-itens, não haverá seta para este item, como, por exemplo, todos os itens de menu que representam tipos simples XSD (prefixo `[s-t]`).

Grasews permite que múltiplos documentos WSDL sejam abertos simultaneamente. Porém, apenas uma especificação WSDL pode ser visualizada no grafo por vez. A Figura 27 ilustra os dois ícones utilizados pela ferramenta para indicar qual especificação WSDL está sendo vista ou não no grafo. O ícone com um olho verde indica que a

Figura 27 – Menu *tree-view* de Grasews para uma especificação WSDL.



Fonte: Autoria própria.

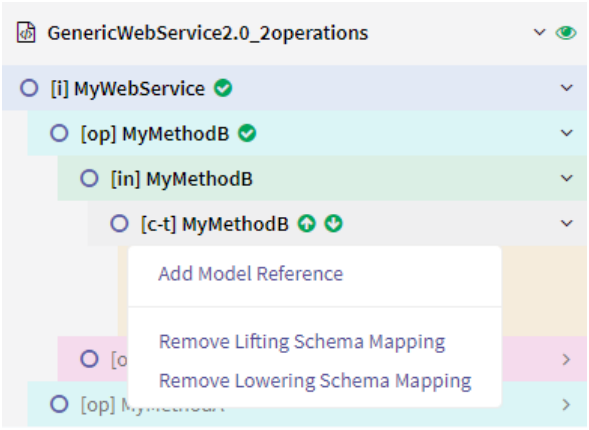
especificação `GenericWebService2.0_2operations` encontra-se visível no grafo para o usuário, enquanto que o ícone com o olho vermelho cortado indica que as especificações `MovieStoreService2.0` e `BookStoreService_1operation2.0` não estão visíveis no grafo para o usuário.

O menu *tree-view* de Grasews permite não apenas a visualização simplificada da estrutura de uma especificação WSDL mas também facilita o processo de anotação semântica. Por meio de um menu de contexto, acionado a partir de um elemento do menu *tree-view*, Grasews provê acesso às funcionalidades de anotação semântica. A Figura 28 ilustra o menu de contexto, acionado para o elemento `[c-t] MyMethodB`, juntamente com as notações visuais (ícones) que auxiliam na anotação semântica de uma especificação WSDL.

O menu *tree-view* WSDL utiliza três ícones que indicam com qual atributo da anotação semântica um elemento WSDL/XSD está anotado. Na Figura 28, o ícone verde com a marca de checagem, na lateral direita dos elementos `[i] MyWebService` e `MyMethodB`, indica o uso do atributo *Model Reference*. O ícone verde com a seta para cima, na lateral direita do elemento `[c-t] MyMethodB`, indica o uso do atributo *Lifting Schema Mapping*. Finalmente, o ícone verde com a seta para baixo, também na lateral direita do elemento `[c-t] MyMethodB`, indica o uso do atributo *Lowering Schema Mapping*.

Quando o elemento WSDL/XSD é passível de ser anotado apenas com *Model Reference*, o menu de contexto disponibiliza apenas a funcionalidade de adicionar *Model Reference*. Quando o elemento WSDL/XSD já está anotado com *Model Reference*, o menu de contexto de um elemento WSDL/XSD provê a funcionalidade de remoção da anotação

Figura 28 – Menu *tree-view* de Grasews para uma especificação WSDL.



Fonte: Autoria própria.

semântica. Note que, como é possível ter múltiplos URIs para *Model Reference*, a funcionalidade de adicionar *Model Reference* continuará disponível para o usuário. Adicionalmente, a funcionalidade de remover *Model Reference* também continuará disponível até que todos os URIs de *Model Reference* sejam removidos do elemento.

Quando um elemento WSDL/XSD também é passível de ser anotado com *Lifting Schema Mapping* e *Lowering Schema Mapping*, o menu de contexto disponibiliza funcionalidades tanto para adicionar um URI para *Lifting Schema Mapping* quanto para adicionar um URI para *Lowering Schema Mapping*. Quando já anotado com *Lifting Schema Mapping*, o menu de contexto de um elemento XSD provê a funcionalidade de remoção da anotação semântica. Como é possível ter apenas um atributo *Lifting Schema Mapping*, a funcionalidade de adicionar *Lifting Schema Mapping* está indisponível para o usuário quando o elemento já está anotado com *Lifting Schema Mapping*. O mesmo se aplica para *Lowering Schema Mapping*.

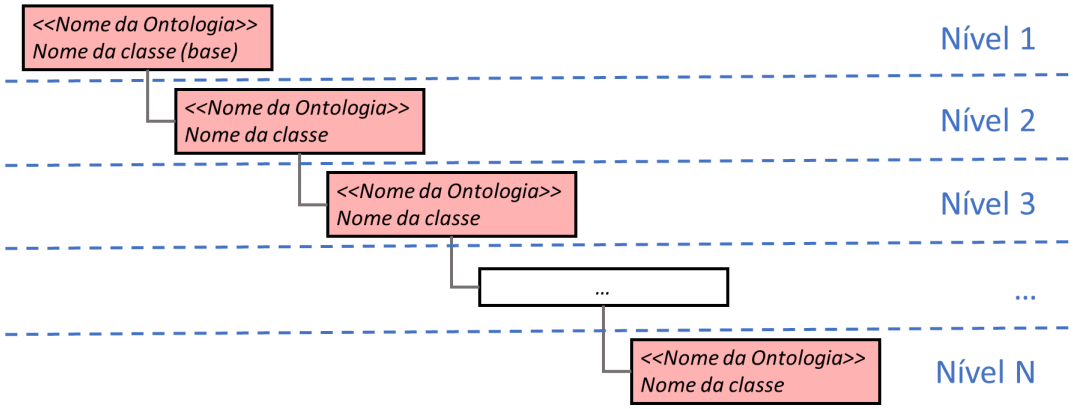
4.3.2.2 Menu *tree-view* OWL

Propusemos também o uso de um menu *tree-view* para auxiliar na compreensão das classes OWL passíveis de serem utilizadas na anotação semântica por meio do atributo *Model Reference* de SAWSDL. A representação do menu *tree-view* para classes de uma ontologia OWL facilita a identificação da hierarquia entre as classes OWL. Classes de conceitos mais genéricos encontram-se mais próximas a classes raiz da árvore, enquanto que classes de conceitos mais específicos encontram-se mais próximas das folhas da árvore. No menu *tree-view* OWL, todas as classes OWL são representadas, independentemente se são utilizadas em uma anotação semântica ou não.

A Figura 29 ilustra o uso de duas classes hierarquicamente dispostas no menu

tree-view. No primeiro nível (nível 1), encontramos a classe base com um conceito mais genérico, enquanto que nos níveis inferiores (de nível 2 a nível N), encontramos as classes de conceitos mais específicos. Esta estrutura pode conter N níveis, dependendo da ontologia e da quantidade de especializações que pode existir entre suas classes.

Figura 29 – Estrutura da representação visual do menu *tree-view* para uma ontologia OWL.



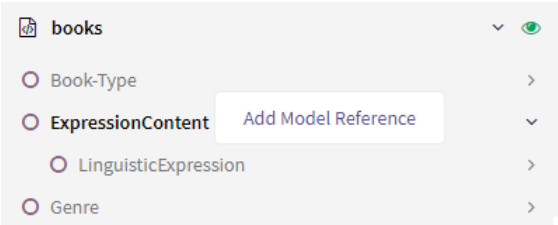
Fonte: Autoria própria.

O menu *tree-view* OWL é localizado no painel direito de Grasews (veja Figura 22). O menu *tree-view* de uma ontologia OWL também provê recursos para o processo de anotação semântica por meio de um menu de contexto. Assim como no menu *tree-view* WSDL, o menu de contexto do menu *tree-view* OWL pode ser acessado por meio do botão direito do *mouse* em uma classe OWL do menu. Por meio desta funcionalidade, a anotação semântica pode ser feita a partir de um conceito de uma ontologia ao invés de um elemento WSDL/XSD. Esta abordagem sugere que primeiro seja selecionada a classe OWL com a qual pretende-se anotar um elemento WSDL/XSD. Após selecionada a classe OWL, o elemento WSDL/XSD, com o qual pretende-se anotar utilizando a classe OWL, deve ser selecionado. O resultado (anotação semântica) obtido por meio desta abordagem é o mesmo que obtido seguindo a abordagem de seleção do elemento WSDL/XSD e posteriormente a classe OWL, proposta pelo grafo WSDL (subseção 4.3.1) e pelo menu *tree-view* WSDL (subseção 4.3.2.1). A Figura 30 ilustra o menu de contexto associado à classe OWL `ExpressionContent` para a anotação utilizando o atributo *Model Reference*. Neste exemplo, Grasews possibilita que um usuário utilize o URI de `ExpressionContent` na anotação de um elemento WSDL/XSD por meio do atributo *Model Reference*.

4.4 Suporte à Edição Colaborativa

Grasews provê suporte para a anotação semântica de forma colaborativa, ou seja, múltiplos usuários podem simultaneamente anotar semanticamente uma especificação WSDL. Esta

Figura 30 – Menu de contexto para uma classe OWL no menu *tree-view*.



Fonte: Autoria própria.

seção apresenta uma visão geral do suporte provido para a anotação semântica de forma colaborativa.

4.4.1 Anotação Semântica Colaborativa

Um projeto consiste de uma especificação WSDL juntamente com as ontologias OWL associadas. Uma ontologia associada consiste de uma ontologia que já foi utilizada em alguma anotação semântica de algum elemento da especificação WSDL (vínculo explícito) ou então de ontologias disponíveis na aplicação durante o compartilhamento (vínculo implícito).

O primeiro passo para o trabalho colaborativo é compartilhar o projeto com outro usuário. Grasews disponibiliza um menu com funcionalidades relacionadas à edição colaborativa. Um usuário de Grasews pode convidar outro usuário para anotar colaborativamente uma especificação WSDL. O usuário convidado pode ser tanto um novo usuário quanto um usuário já existente. O usuário convidado receberá um *e-mail* que permitirá que ele aceite ou rejeite o convite para a edição colaborativa. Caso seja um novo usuário e aceite o convite, Grasews solicitará que realize o cadastro na ferramenta. Tendo aceito o convite e após autenticar-se na ferramenta, o usuário poderá abrir a especificação WSDL e as ontologias OWL compartilhadas com ele, da mesma forma como se tivesse sido este usuário que houvesse carregado estes documentos na ferramenta.

Múltiplos usuários podem anotar semanticamente uma especificação WSDL. Caso dois ou mais usuários estejam anotando uma mesma especificação WSDL simultaneamente, todo o trabalho realizado por qualquer usuário é compartilhado e disponibilizado aos demais. Desta forma, o grafo e o menu *tree-view* são automaticamente atualizados de forma a representar a última versão da especificação WSDL compartilhada. De modo a facilitar o trabalho colaborativo e remoto, Grasews provê um conjunto de notificações que são disparadas aos usuários conforme o trabalho é realizado. Tais notificações são exibidas no canto inferior direito da ferramenta.

A Figura 31 ilustra um caso de uso durante o processo de anotação semântica de

forma colaborativa. À esquerda, encontram-se os estados do ambiente de trabalho de um `Usuário A`, enquanto que à direita encontram-se os ambientes de trabalho de um `Usuário B`. A Figura 31a e a Figura 31b ilustram os ambientes de trabalho do `Usuário A` e do `Usuário B`, respectivamente, em seus estados iniciais. A Figura 31c ilustra o ambiente de trabalho do `Usuário A` após este ter adicionado um *Model Reference* ao elemento tipo complexo `MyMethodResponseB`, na lateral direita do grafo. Simultaneamente, a Figura 31d ilustra o ambiente de trabalho do `Usuário B` com o mesmo estado inicial, pois a nova anotação semântica ainda não foi notificada ao `Usuário B`. A Figura 31e ilustra o ambiente de trabalho do `Usuário A` com a anotação semântica feita anteriormente. Finalmente, a Figura 31f ilustra o ambiente de trabalho do `Usuário B` após receber a notificação e a atualização com a anotação semântica criada pelo `Usuário A` anteriormente.

Grasews controla o trabalho colaborativo por meio dos módulos `Grasews.Application` e `Grasews.Postgres`. `Grasews.Application` garante que requisições de anotações semânticas, criadas pelo módulo `Grasews.API`, sejam verificadas e validadas juntamente com o `Grasews.Postgres` a fim de que um usuário não sobrescreva o trabalho de outro usuário.

Na anotação semântica por meio de *Model Reference*, Grasews verifica se o mesmo conceito de uma mesma ontologia já encontra-se anotado para um dado elemento WSDL/XSD. Caso este elemento já esteja anotado com esse termo, Grasews impede que uma anotação seja feita de forma duplicada e notificará seus usuários, informando-os que a anotação semântica (associação) já existe entre o elemento WSDL/XSD e a classe OWL. Caso sejam conceitos distintos, Grasews permite que a anotação seja feita, visto que *Model Reference* permite múltiplos URIs.

Já na anotação de um dado elemento WSDL/XSD com os atributos do tipo *Lifting Schema Mapping* e *Lowering Schema Mapping*, Grasews impedirá que uma nova anotação seja feita sempre que estes tipos de atributos já tiverem sido utilizados em um dado elemento. Assim, ao tentar anotar um elemento WSDL/XSD com *Lifting Schema Mapping*, Grasews verifica se este elemento já está anotado com este tipo de atributo ou não. Caso esteja, a nova anotação será descartada e uma mensagem de erro será informada ao usuário. O mesmo ocorre para a anotação com *Lowering Schema Mapping*. Ao tentar anotar um dado elemento WSDL/XSD com *Lowering Schema Mapping*, Grasews verifica se este elemento já está anotado com este tipo de atributo ou não. Caso esteja, a nova anotação será descartada e informada ao usuário. Deste modo, Grasews garante que uma anotação não sobrescreva outra por engano. Caso um usuário queira sobrescrever a anotação, é necessário remover a anotação existente e criar uma nova.

De forma a evitar inconsistências nas edições colaborativas de uma especificação WSDL, Grasews enfileira todas as requisições de anotação, tratando cada uma delas individualmente, considerando as regras já descritas. Com isso, todas as requisições de anotação realizadas de forma concorrente são processadas na ordem em que essas são

recebidas. Assim, caso dois usuários tentem anotar um elemento XSD com o mesmo tipo de atributo (*Lifting Schema Mapping* e *Lowering Schema Mapping*), a primeira requisição que for processada será persistida, enquanto que as demais serão descartadas. A anotação processada será notificada ao usuário que a requisitou, informando-o acerca da confirmação de que a anotação foi realizada com sucesso. Simultaneamente, a ferramenta irá notificar os demais usuários acerca de que uma nova anotação semântica foi realizada na especificação WSDL e que, portanto, o projeto foi automaticamente atualizado. Adicionalmente, para os demais usuários que tiveram suas anotações descartadas, a ferramenta irá notificá-los acerca de suas anotações rejeitadas.

Por fim, cada usuário que possui uma especificação WSDL compartilhada com outros usuário pode reposicionar os nós do grafo individualmente. Grasews permite que cada usuário salve o seu próprio grafo com os ajustes (reposicionamentos) realizados. Este recurso também possibilita que cada usuário possa realizar a anotação semântica de forma personalizada, já que cada usuário pode compreender o grafo WSDL de forma diferente. Entretanto, as anotações realizadas na especificação e, consequentemente, as notações visuais do grafo são automaticamente compartilhadas, independentemente das posições dos nós do grafo. Note que na Figura 31, as posições dos nós do grafo do `Usuário A`, à esquerda, diferem das posições dos nós do grafo do `Usuário B`, à direita.

4.4.2 Questões & Tarefas

O processo de anotação semântica colaborativo é auxiliado por meio da definição de questões e tarefas. Quando um usuário possui algum dúvida específica em relação à anotação de um elemento de uma especificação WSDL, este usuário pode criar uma questão (*issue*) associada ao elemento. Uma questão não é direcionada a um usuário específico, i.e., qualquer usuário envolvido na edição colaborativa é capaz de visualizar as questões para uma especificação WSDL e respondê-las. É possível ter múltiplas respostas de múltiplos usuários. Quando uma questão é considerada devidamente respondida, o usuário que criou-a pode marcá-la como resolvida. A Figura 32 ilustra a interface gráfica de Grasews para a listagem de questões. Por meio desta interface gráfica, um usuário pode adicionar respostas às questões e, oportunamente, marcá-las como resolvidas.

Quando um elemento WSDL/XSD possui uma questão não resolvida, o grafo WSDL de Grasews representa visualmente este elemento na cor amarela, substituindo, portanto, a cor original deste elemento. Caso o elemento WSDL/XSD não possua mais questões relacionadas, este elemento é representado em sua cor original conforme a notação visual proposta por este trabalho.

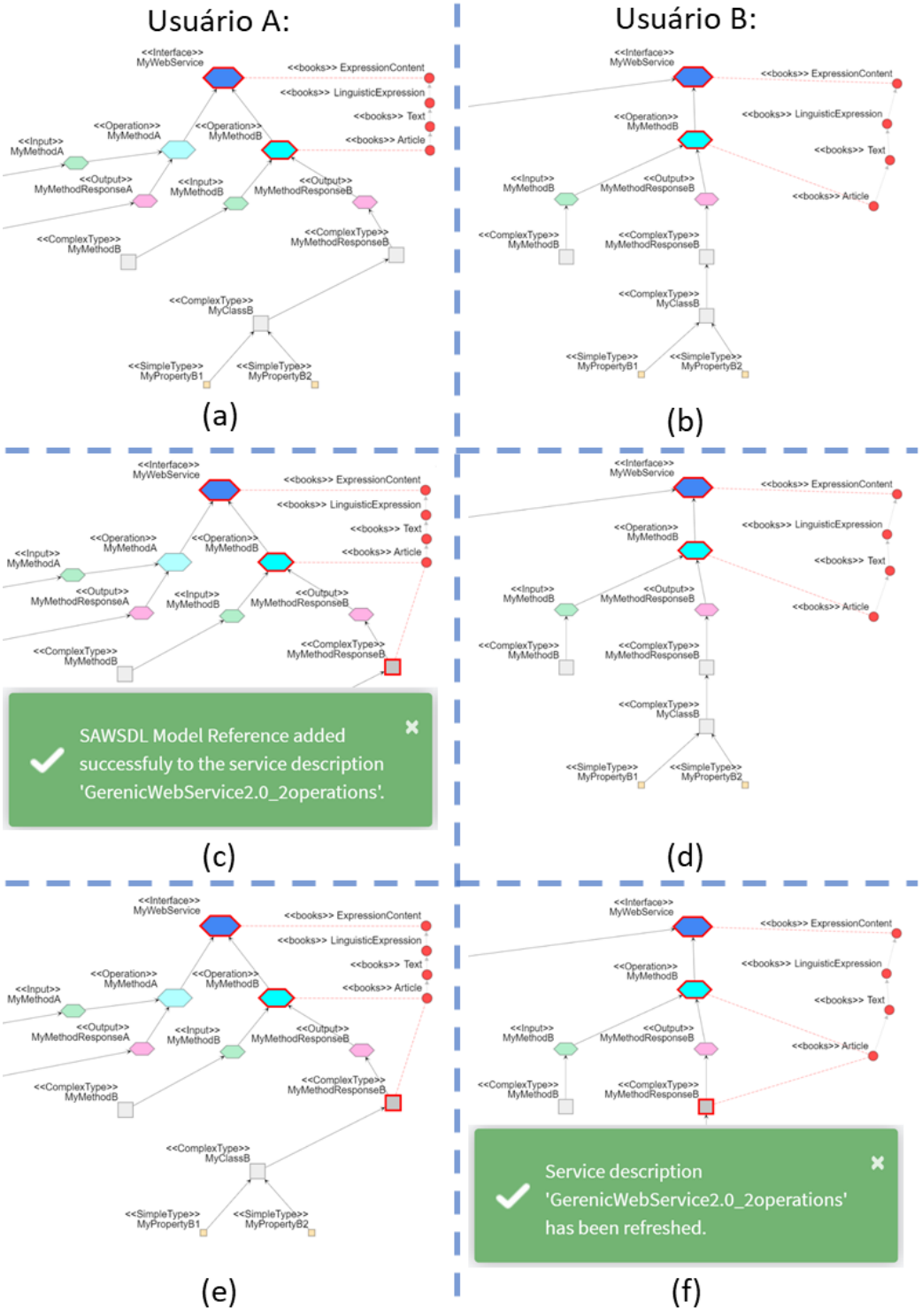
A Figura 33 ilustra diferentes notações visuais envolvendo a anotação semântica de forma colaborativa e auxiliada por questões. A especificação WSDL exemplificada pela

Figura 33 está compartilhada entre dois usuários: `Usuário A` e `Usuário B`. A Figura 33a ilustra a interface `MyWebService` para ambos os usuários em seu estado inicial, sem *Model Reference* e sem questão. A Figura 33b ilustra a interface `MyWebService` com uma questão adicionada pelo `Usuário A`. O `Usuário B` responde esta questão. Esta resposta possibilitou que o `Usuário A` anotasse este elemento. A Figura 33c ilustra a interface `MyWebService` anotada, porém ainda com a questão. Finalmente, após ter realizada a anotação semântica, o `Usuário A` marca a sua questão como resolvida. A Figura 33d ilustra, portanto, a notação visual deste elemento, agora com *Model Reference* e sem questão.

Já uma tarefa representa uma lista do trabalho necessário a ser realizado para concluir o processo de anotação semântica. A definição de uma ou mais tarefas pode contribuir para a coordenação do processo e anotação semântica entre múltiplos usuários envolvidos. Quando uma tarefa é concluída, esta tarefa pode ser marcada desta forma por qualquer usuário envolvido na edição colaborativa. Assim, os usuários mantém o controle das tarefas ainda pendentes de serem feitas.

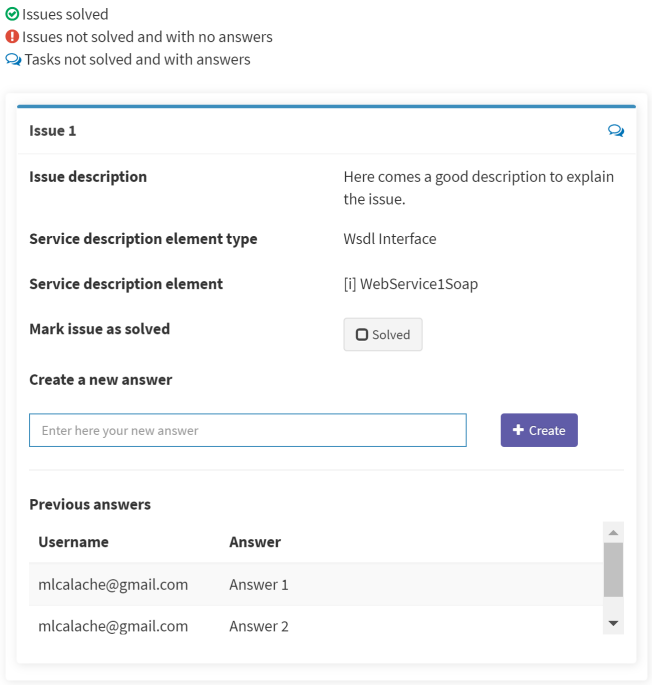
Uma tarefa não é direcionada a um usuário específico, i.e., qualquer usuário envolvido na edição colaborativa é capaz de visualizar as tarefas para uma especificação WSDL. Além de visualizar a lista de tarefas, os usuários podem adicionar comentários, podendo contribuir para a finalização de uma tarefa. É possível ter múltiplos comentários de múltiplos usuários. A Figura 34 ilustra a interface gráfica de Grasews para a listagem de tarefas. Por meio desta interface gráfica, um usuário pode adicionar comentários às tarefas e, oportunamente, marcá-las como concluídas.

Figura 31 – Processo de anotação semântica de forma colaborativa de Grasews. (a) Estado inicial de `Usuário A`. (b) Estado inicial de `Usuário B`. (c) `Usuário A` adiciona uma anotação semântica. (d) `Usuário B` ainda com estado inicial. (e) Estado final de `Usuário A` com a nova anotação semântica. (f) `Usuário B` recebe notificação e atualização com a nova anotação semântica.



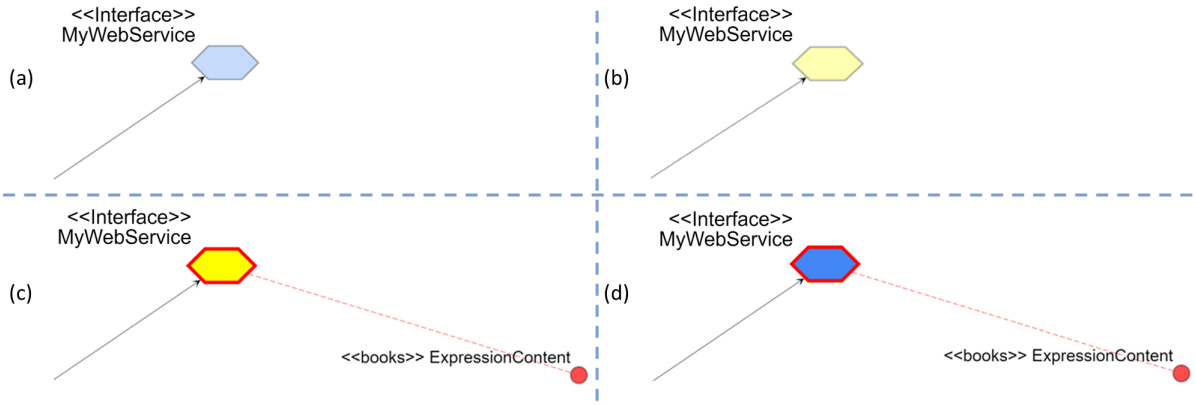
Fonte: Autoria própria.

Figura 32 – Interface gráfica para a listagem de questões.



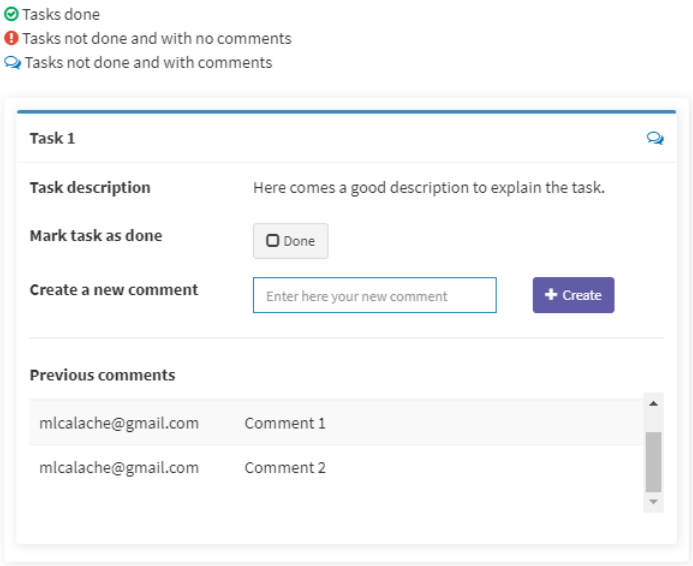
Fonte: Autoria própria.

Figura 33 – Uso de questões para a anotação semântica de forma compartilhada. (a) Estado inicial da interface WSDL. (b) Interface WSDL com questão associada. (c) Interface WSDL com *Model Reference* e com questão associada. (d) Interface WSDL com *Model Reference*.



Fonte: Autoria própria.

Figura 34 – Interface gráfica para a listagem de tarefas.



Fonte: Autoria própria.

5

Prova de Conceito

Este capítulo apresenta uma prova de conceito utilizando a ferramenta de anotação semântica colaborativa Grasews. Este capítulo tem por objetivo demonstrar o uso da ferramenta Grasews a fim de anotar semanticamente duas especificações WSDL por meio de notações visuais e de forma colaborativa.

Este capítulo está estruturado da seguinte forma: a seção 5.1 apresenta uma visão geral desta atividade de avaliação; a seção 5.2 descreve o uso da ferramenta para a anotação de um serviço do domínio biomédico; a seção 5.3 descreve o uso da ferramenta para a anotação colaborativa de um serviço do domínio cinematográfico; por fim, a seção 5.4 apresenta os comentários finais.

5.1 Visão Geral

Esta prova de conceito está divida em duas etapas. A primeira etapa consistiu da anotação semântica do serviço `MicroAffyNorm` 1(GUARDIA et al., 2015), desenvolvido para o domínio de genômica funcional, por meio da especificação WSDL e da ontologia OWL providas no repositório deste serviço. Esta etapa teve por objetivo avaliar o suporte provido pela ferramenta Grasews a fim de obter os resultados de anotação semântica similares quando comparados com a especificação WSDL originalmente anotada para este serviço.

`MicroAffyNorm` (GUARDIA et al., 2015) é um serviço web semântico RESTful que permite o pré-processamento (normalização) de dados de *microarray one-color* gerados a partir da plataforma *Affymetrix* (GAUTIER et al., 2004). Este serviço é implementado como um *wrapper* para o pacote R "affy". O pré-processamento dos dados do *microarray Affymetrix* é realizado usando o método *Robust Multi-array Average* (RMA) deste pacote. Este serviço está anotado com conceitos providos pela ontologia OWL `GEXPASO` (GUARDIA et al., 2015). A ontologia `GEXPASO` descreve diferentes processos de análise de expressão

gênica, como, por exemplo, o pré-processamento de dados de *microarray Affymetrix*, *Genepix* e *Agilent*, a identificação de genes diferencialmente expressos em dados de *microarray* e *RNA-Seq*, o agrupamento hierárquico e *k-means* de dados de *microarray*, a geração de dendrogramas de amostras e genes, a análise de enriquecimento simples e de grupos de genes em dados de expressão gênica e o mapeamento e visualização de dados de expressão gênica em processos biológicos *KEGG*.

O processo de anotação semântica do serviço `MicroAffyNorm` foi realizado individualmente por mim, com base nas anotações semânticas originais da especificação WSDL. Como a especificação deste serviço já estava originalmente anotada e a especificação WSDL original deste serviço não estava disponível, foi necessário remover todos os atributos SAWSDL e referências ao *namespace* de SAWSDL, de modo a obter uma especificação não anotada (limpa).

Tanto a especificação WSDL não anotada de `MicroAffyNorm` quanto a ontologia OWL `GEXPASO` foram abertas na ferramenta Grasews. O processo seguiu com a anotação semântica da especificação WSDL utilizando somente notações visuais e funcionalidades providas por Grasews. Tendo como base a especificação WSDL originalmente anotada, iniciei a anotação seguindo uma abordagem *top-down*. Segundo esta abordagem, termos mais genéricos são inicialmente anotados e posteriormente termos mais específicos são anotados, aumentando o nível de granularidade e especifidade da anotação semântica. Neste sentido, iniciei a anotação das operações, seguido dos tipos complexos e, finalmente, dos tipos simples de dados.

A segunda etapa consistiu da anotação semântica colaborativa do serviço `MovieStore`, desenvolvido para o domínio cinematográfico. Esta etapa teve por objetivo anotar a especificação WSDL deste serviço por meio da edição colaborativa, envolvendo um desenvolvedor de serviços e um especialista de domínio.

De modo a anotar semanticamente a especificação WSDL deste serviço, utilizamos a ontologia OWL `MovieOntology` (BOUZA, 2010), desenvolvida pelo Departamento de Informática da Universidade de Zurique. Esta ontologia contém hierarquias de conceitos para categorização de filmes que permitem a apresentação amigável de descrições de filmes. Como a ontologia `MovieOntology` representa uma ontologia de domínio e não de serviços (ações), focamos na anotação apenas dos tipos de dados. Entretanto, realizamos a anotação semântica apenas nos tipos complexos de dados XSD.

A anotação semântica colaborativa do serviço `MovieStore` foi realizada por dois indivíduos: eu, denominado `Desenvolvedor`, e um especialista de domínio, denominado `Especialista`. O `Desenvolvedor` consiste de um desenvolvedor de serviços web, possuindo, portanto, conhecimento especializado na tecnologia envolvida no desenvolvimento de serviços web semânticos e acerca de quais elementos WSDL/XSD devem ser anotados.

Embora o domínio cinematográfico possa ser considerado mais simples se comparado ao domínio de genômica funcional, o `Desenvolvedor` não possui, a princípio, o conhecimento especializado deste domínio e, consequentemente, acerca de quais termos da ontologia que devem ser utilizados efetivamente na anotação. Já o `Especialista` consiste de um especialista no domínio cinematográfico, dado que este possui um sistema web de busca de filmes e venda de ingressos de cinema desde 2012, ou seja, portanto, há mais de sete anos.

5.2 Anotação do serviço MicroAffyNorm

O serviço `MicroAffyNorm` (GUARDIA et al., 2015) é composto por um conjunto de sete (7) operações. A operação `generateID` é responsável por criar um contexto de execução de uma atividade de análise de expressão gênica responsável pela normalização de um conjunto de dados de *microarray Affymetrix*. A operação `sendFile` é responsável por enviar um arquivo de dados de *microarray Affymetrix* para o serviço de análise, o qual será amarmazenado no contexto de execução criado anteriormente. A operação `listFiles` é responsável por obter o nome de todos os arquivos de *microarray Affymetrix* do ambiente de trabalho. A operação `removeFile` é responsável por remover um arquivo *Affymetrix* do ambiente de trabalho. A operação `performAffyNorm` é responsável por normalizar os arquivos de *microarrays Affymetrix* do ambiente de trabalho. A operação `getStatus` é responsável por obter o estado de execução da atividade de normalização dos arquivos do contexto de execução. Por fim, a operação `getResult` é responsável por obter o arquivo de dados de *microarray* normalizado armazenado no contexto de execução.

O processo de anotação semântica do serviço `MicroAffyNorm` iniciou-se com a fase denominada *Limpeza da Especificação*. Nesta fase, a especificação WSDL anotada originalmente foi manualmente editada a fim de ser obter uma especificação WSDL não anotada (limpa). Adicionalmente, foram identificados os elementos que estavam anotados originalmente e, consequentemente, quais classes OWL estava sendo utilizadas nas anotações. A próxima fase do processo foi denominada *Criação do Projeto de Trabalho*. Nesta fase, tanto a especificação WSDL quanto a ontologia OWL foram abertas na ferramenta, compreendidas e validadas, comparando o código XML/WSDL com os elementos WSDL/XSD identificados e hierarquicamente estruturados pela ferramenta. O processo seguiu, por fim, com a fase denominada *Anotação Semântica*. Nesta fase, as anotações foram criadas com base na especificação WSDL anotada originalmente.

Tanto a especificação WSDL não anotada do serviço `MicroAffyNorm` quanto a ontologia OWL `GEXPASO` foram abertas na ferramenta Grasews e, portanto, salvas no contexto do usuário. Desta forma, foi possível melhor compreender os elementos da especificação e a sua estrutura hierárquica de elementos, visto que a especificação WSDL

foi visualmente representada pelo grafo e, adicionalmente, pela estrutura hierárquica do menu *tree-view*. Como o processo de anotação foi realizado apenas por uma pessoa, não foi necessário utilizar qualquer funcionalidade de suporte ao trabalho colaborativo.

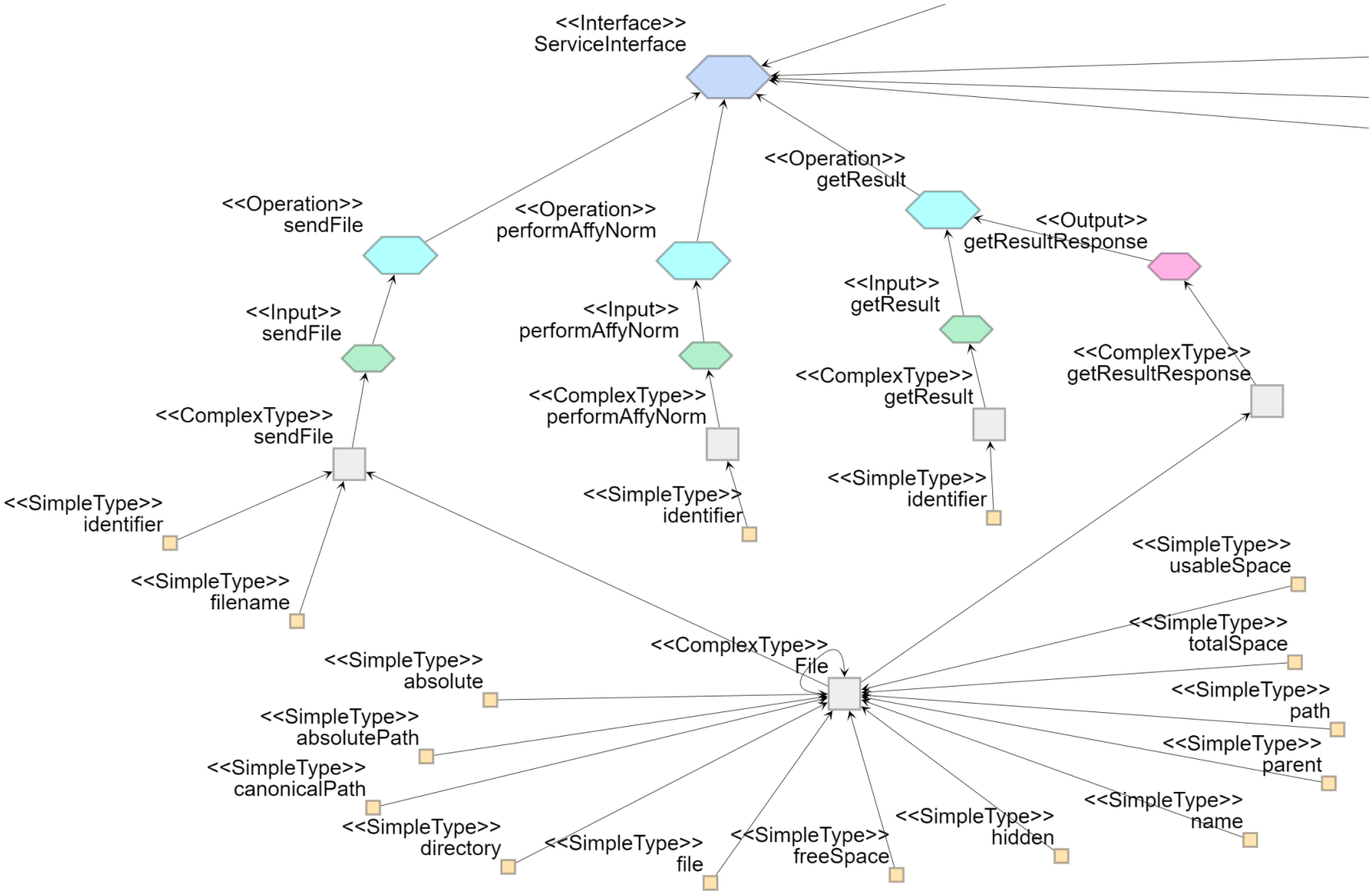
A Figura 35 ilustra parte do grafo da especificação WSDL do serviço `MicroAffyNorm`, sem anotação. O grafo ilustra a interface `ServiceInterface` composta pelas operações `sendFile`, `performAffyNorm` e `getResult`. A operação `sendFile` é composta pelo parâmetro de entrada `sendFile`. Este parâmetro utiliza o tipo complexo `sendFile`, que por sua vez é composto pelos tipos simples `identifier` e `fileName` e pelo tipo complexo `File`. O tipo complexo `File` é composto por ele mesmo e pelos tipos simples `absolute`, `absolutePath`, `canonicalPath`, `directory`, `file`, `freeSpace`, `hidden`, `name`, `parent`, `parent`, `path`, `totalSpace` e, finalmente, `usableSpace`. A operação `performAffyNorm` é composta pelo parâmetro de entrada `performAffyNorm`. Este parâmetro utiliza o tipo complexo `performAffyNorm`, que por sua vez é composto pelo tipo simples `identifier`. Finalmente, a operação `getResult` é composto pelo parâmetro de entrada `getResult` e pelo parâmetro de saída `getResultResponse`. O parâmetro de entrada `getResult` utiliza o tipo complexo `getResult`, que por sua vez é composto pelo tipo simples `identifier`. Já o parâmetro de saída `getResultResponse` utiliza o tipo complexo `getResultResponse`, que por sua vez é composto pelo tipo complexo `File`.

Seguindo a abordagem *top-down*, o primeiro passo desta etapa de prova de conceito foi anotar a operação `performAffyNorm`. As demais operações deste serviço não foram anotadas em razão da especificação WSDL anotada originalmente não conter tais anotações e por não haver conceitos adequados para este propósito na ontologia `GEXPASO`. Posteriormente, anotamos tipos complexos e simples tentando nos aproximar das anotações contidas na especificação WSDL anotada originalmente.

Ao final do processo, uma nova especificação WSDL foi gerada e salva, para que pudéssemos compará-la com a especificação WSDL anotada originalmente. A Figura 36 ilustra parte do grafo da especificação WSDL do serviço `MicroAffyNorm` após finalizada a anotação semântica por meio da ferramenta Grasews.

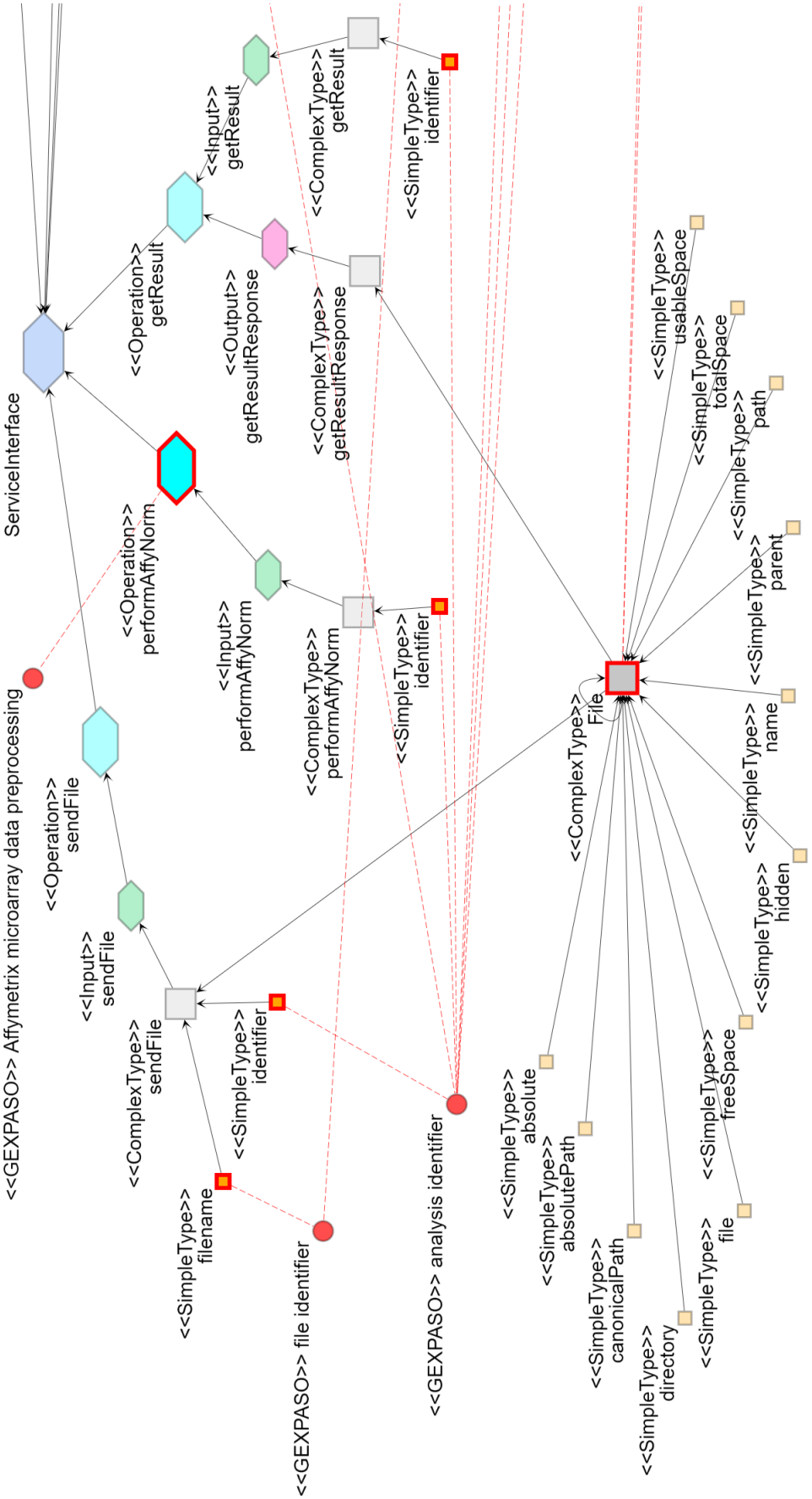
O resultado final da especificação WSDL do serviço `MicroAffyNorm` anotado utilizando Grasews ficou ligeiramente diferente quando comparado com a especificação WSDL anotada originalmente. Tal fato decorre em razão de uma decisão de projeto relacionada à forma com a qual Grasews anota semanticamente elementos XSD de uma especificação WSDL. Grasews foca em anotar a declaração do tipo XSD ao invés do uso do tipo XSD. Por exemplo, o tipo complexo `File` é utilizado tanto pelo elemento de tipo complexo `getResultResponse`, por meio de seu elemento interno `return`, do tipo `File`, quanto pelo elemento complexo `sendFile`, por meio de seu elemento interno `file`, do tipo `File`. Na especificação WSDL anotada originalmente, o elemento `return` encontra-se anotado com o URI da classe OWL `GEXPASO_0000098` (*log-transformed one-*

Figura 35 – Parte da especificação WSDL do serviço `MicroAffyNorm`, sem anotação.



Fonte: Autoria própria.

Figura 36 – Parte da especificação WSDL do serviço `MicroAffyNorm` após anotada semanticamente.



Fonte: Autoria própria.

color normalized microarray data)(veja Listagem 2, linha 5). Por sua vez, o elemento `file` encontra-se anotado com o URI da classe OWL `GEXPASO_0000021` (*Affymetrix raw microarray data*)(veja Listagem 2, linha 15). Finalmente, a declaração do tipo `File` (Listagem 2, linha 22), não possui qualquer tipo de anotação associada.

Como os tipos `return` de `getResultResponse` e `file` de `sendFile` são declarados como sendo ambos do tipo `File`, ao criar a anotação semântica para cada um, Grasews atribuiu o URI das duas classes OWL ao elemento XSD de tipo complexo `File`. Portanto, como resultado, `File` possui dois conceitos associados, `GEXPASO_0000098` e `GEXPASO_0000021` (veja Listagem 3, linha 22), enquanto que os tipos `getResultResponse` e `sendFile` não são anotados (veja Listagem 3, linhas 5 e 15).

```
1  <!-- Tipo complexo getResultResponse, composto pelo elemento return do tipo File -->
2  <xs:element name="getResultResponse">
3      <xs:complexType>
4          <xs:sequence>
5              <xs:element minOccurs="0" name="return" nillable="true" type="ax21:File"
       sawsdl:modelReference="http://dcm.ffclrp.usp.br/lssb/geas/ontologies/GEXPASO.owl#
       GEXPASO_0000098"/>
6          </xs:sequence>
7      </xs:complexType>
8  </xs:element>
9  ...
10 <!-- Tipo complexo sendFile, composto pelo elemento file do tipo File -->
11 <xs:element name="sendFile">
12     <xs:complexType>
13         <xs:sequence>
14         ...
15             <xs:element minOccurs="0" name="file" nillable="true" type="ax21:File"
       sawsdl:modelReference="http://dcm.ffclrp.usp.br/lssb/geas/ontologies/GEXPASO.owl#
       GEXPASO_0000021"/>
16         </xs:sequence>
17     </xs:complexType>
18 </xs:element>
19 ...
20 <!-- Tipo complexo File -->
21 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="qualified"
       elementFormDefault="qualified" targetNamespace="http://io.java/xsd">
22     <xs:complexType name="File">
23         <xs:sequence>
24             ...
25         </xs:sequence>
26     </xs:complexType>
27 </xs:schema>
```

Listagem 2 – Elementos XSD do serviço MicroAffyNorm anotados originalmente.

```
1  <!-- Tipo complexo getResultResponse, composto pelo elemento return do tipo File -->
2  <xs:element name="getResultResponse">
3      <xs:complexType>
4          <xs:sequence>
```

```
5             <xs:element minOccurs="0" name="return" nillable="true" type="ax21:File" />
6         </xs:sequence>
7     </xs:complexType>
8 </xs:element>
9 ...
10 <!-- Tipo complexo sendFile, composto pelo elemento file do tipo File -->
11 <xs:element name="sendFile">
12     <xs:complexType>
13         <xs:sequence>
14             ...
15             <xs:element minOccurs="0" name="file" nillable="true" type="ax21:File" />
16         </xs:sequence>
17     </xs:complexType>
18 </xs:element>
19 ...
20 <!-- Tipo complexo File -->
21 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="qualified"
       elementFormDefault="qualified" targetNamespace="http://io.java/xsd">
22     <xs:complexType name="File" sawsdl:modelReference="http://dcm.ffclrp.usp.br/lssb/geas
       /ontologies/GEXPASO.owl#GEXPASO_0000021 http://dcm.ffclrp.usp.br/lssb/geas/ontologies
       /GEXPASO.owl#GEXPASO_0000098">
23         <xs:sequence>
24             ...
25         </xs:sequence>
26     </xs:complexType>
27 </xs:schema>
```

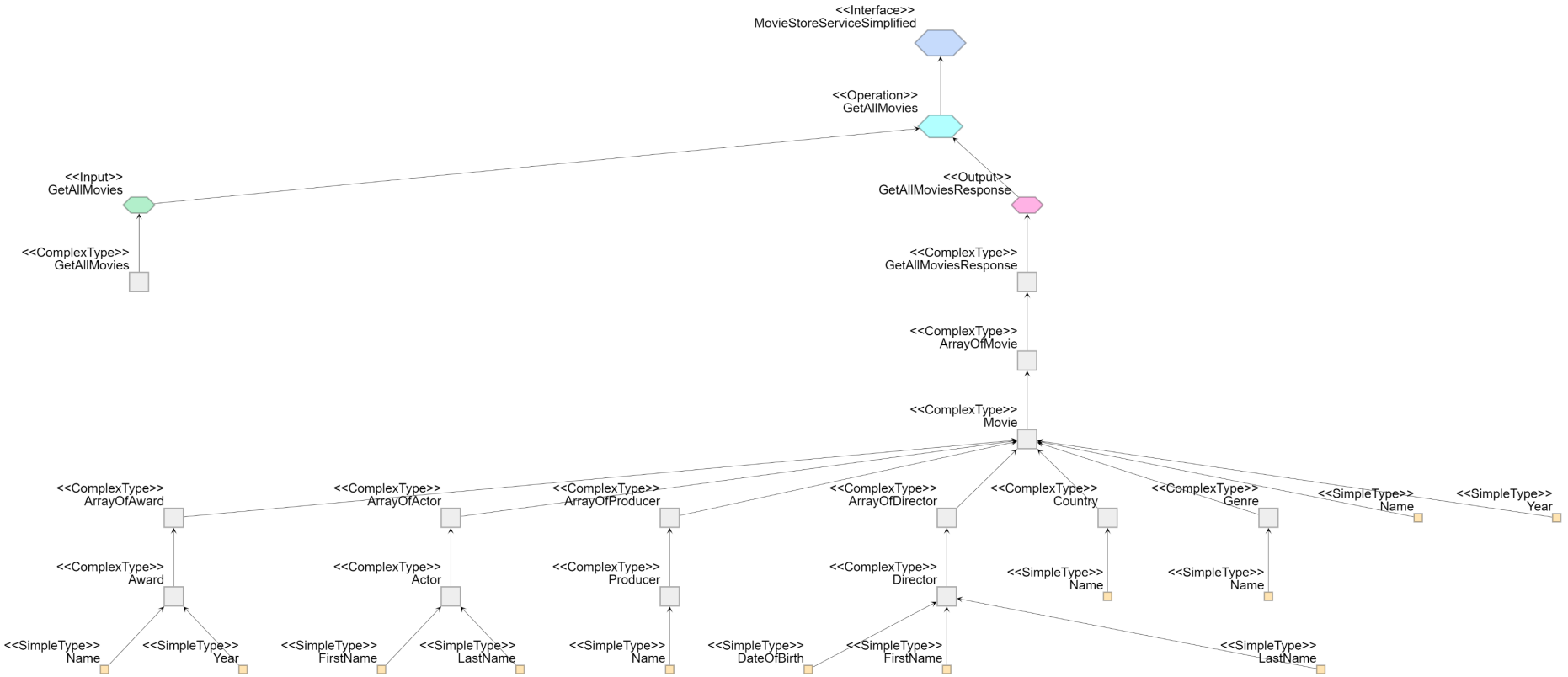
Listagem 3 – Elementos XSD anotados utilizando Grasews e diferentes dos elementos originais.

5.3 Anotação do serviço MovieStore

O serviço `MovieStore` é composto apenas pela operação `GetAllMovies`, responsável por buscar todos os filmes em um dado repositório. Esta operação tem como retorno uma coleção de filmes. Um filme é composto por uma lista de premiações (`Awards`), uma lista de atores (`Actor`), uma lista de produtoras (`Producer`), uma lista de diretores (`Director`), um país (`Country`), um gênero (`Genre`), um nome (`Name`) e um ano de lançamento (`Year`). Uma premiação (`Award`) é composta pelo nome (`Name`) e pelo ano da premiação (`Year`). Um ator (`Actor`) é composto pelo primeiro nome (`FirstName`) e pelo sobrenome (`LastName`). Uma produtora (`Producer`) é composta pelo nome (`Name`). Um diretor (`Director`) é composto pela data de nascimento (`DateOfBirth`), pelo primeiro nome (`FirstName`) e pelo sobenome (`LastName`). Um país (`Country`) é composto pelo nome (`Name`). Por fim, um gênero (`Genre`) é composto pelo nome (`Name`). A Figura 37 ilustra o grafo para a especificação WSDL do serviço `MovieStore`.

O processo de anotação semântica colaborativo do serviço `MovieStore` iniciou-se a fase denominada *Compartilhamento do Projeto*. Nesta fase, a especificação WSDL e a ontologia OWL foram abertas pelo `Desenvolvedor` na ferramenta e, posteriormente,

Figura 37 – Grafo da especificação WSDL do serviço `MovieStore`, sem anotação.



Fonte: Autoria própria.

disponibilizadas ao `Especialista`, por meio do compartilhamento do projeto. A próxima fase do processo foi denominada *Definição do Escopo de Trabalho*. Nesta fase, um conjunto de tarefas foi criado na ferramenta como uma forma de restringir e guiar o trabalho necessário a ser realizado para a conclusão da anotação semântica. O processo seguiu com a fase denominada *Alinhamento de Conhecimento*. Nesta fase os usuários tentaram entender o que precisava ser feito e à medida que dúvidas surgiam, questões eram criadas e solucionadas por meio da ferramenta.

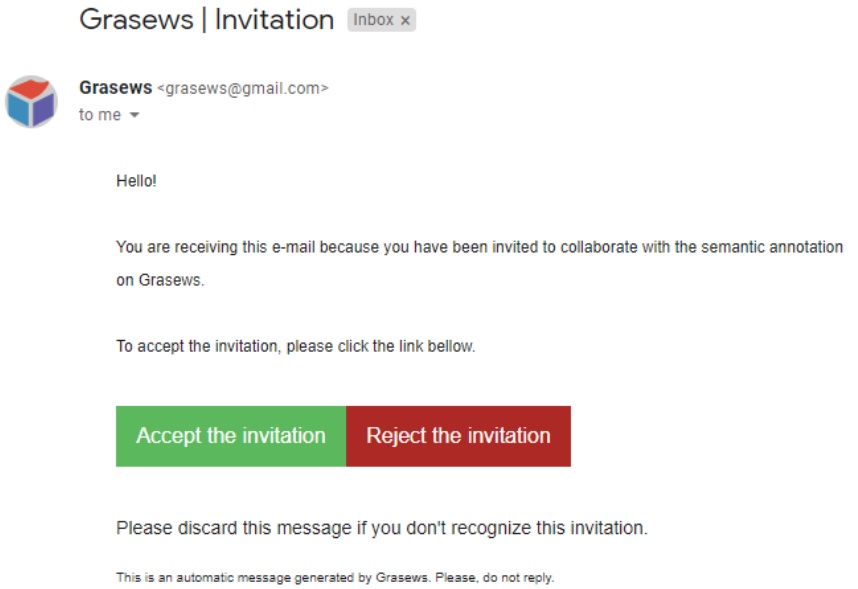
Por meio de um conjunto de questões criado na ferramenta, ambos usuários puderam trabalhar colaborativamente a fim de solucionar questões (dúvidas) que contribuíram para um melhor entendimento da especificação WSDL e, consequentemente, para anotar semanticamente este serviço de forma mais eficaz. Por fim, na fase denominada *Anotação Semântica*, as anotações foram criadas seguindo o escopo de trabalho definido por meio das tarefas (*Definição do Escopo de Trabalho*) e apenas após o conhecimento ter sido alinhado por meio da resolução das questões (*Alinhamento de Conhecimento*).

O processo de anotação semântica poderia ter seguido um outro fluxo de trabalho, visto que não há dependência explícita de uma fase com a outra. Por exemplo, o processo poderia ter sido feito da seguinte forma: i) primeiro criaríamos uma única tarefa; ii) depois, eventualmente, criaríamos uma ou mais questões; iii) as eventuais questões criadas seriam respondidas ao mesmo tempo que a anotação seria feita. Este ciclo poderia então iniciar-se novamente até a conclusão de todo processo. Note que a ferramenta Grasews não restringe a ordem de cada atividade, o que possibilita que cada grupo de trabalho utilize a metodologia que for mais conveniente.

Na fase de *Compartilhamento do Projeto*, o `Desenvolvedor` já encontrava-se cadastrado na ferramenta e era identificado, na ferramenta, pelo seu nome de usuário `mlcalache@gmail.com`. O `Desenvolvedor` foi responsável por abrir a especificação WSDL e a ontologia OWL na ferramenta. O `Desenvolvedor` também foi responsável por compartilhar o projeto, i.e., a especificação WSDL e a ontologia OWL, com o `Especialista` por meio do endereço de *e-mail* `usuariograsews@gmail.com`. Como `Especialista` ainda não estava cadastrado na ferramenta, este recebeu, então, uma mensagem de *e-mail* com um convite para o trabalho colaborativo (ver Figura 38) e teve, portanto, que cadastrar-se na ferramenta. Após realizado o cadastrado e a autenticação na ferramenta, o `Especialista` pôde abrir a mesma especificação WSDL compartilhada pelo `Desenvolvedor`.

Na fase de *Definição do Escopo de Trabalho*, o `Desenvolvedor` criou uma lista de tarefas que foi compartilhada com o `Especialista`. A Figura 39a ilustra a interface de usuário para a criação de uma tarefa. Já a Figura 39b ilustra a interface gráfica de usuário para a listagem de tarefas criadas para o serviço, igualmente disponível para ambos usuários. O Quadro 5.1 apresenta as tarefas criadas nesta etapa. Foi criada uma tarefa para cada tipo complexo a ser anotado.

Figura 38 – Mensagem de *e-mail* contendo o convite para a edição colaborativa.



Fonte: Autoria própria.

Quadro 5.1 - Tarefas do serviço MovieStore

(T1) **Título**: Annotate Award Complex Type
Tarefa: Annotate Award Complex Type with a Model Reference

(T2) **Título**: Annotate Actor Complex Type
Tarefa: Annotate Actor Complex Type with a Model Reference

(T3) **Título**: Annotate Producer Complex Type
Tarefa: Annotate Producer Complex Type with a Model Reference

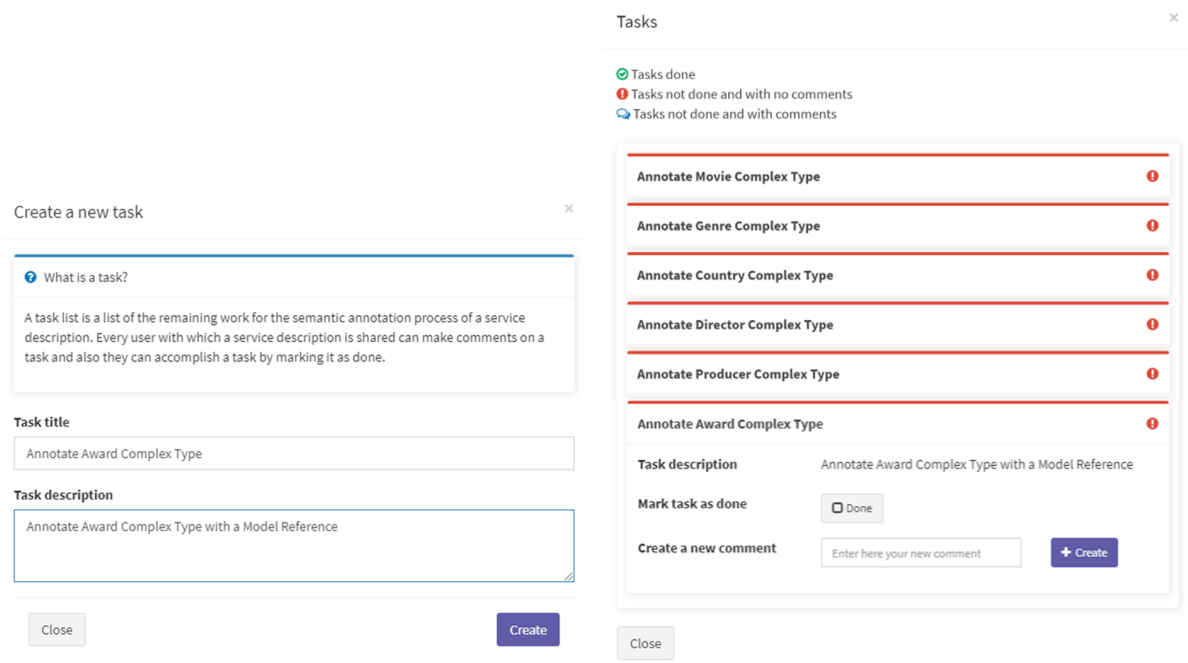
(T4) **Título**: Annotate Director Complex Type
Tarefa: Annotate Director Complex Type with a Model Reference

(T5) **Título**: Annotate Country Complex Type
Tarefa: Annotate Country Complex Type with a Model Reference

(T6) **Título**: Annotate Genre Complex Type
Tarefa: Annotate Genre Complex Type with a Model Reference

(T7) **Título**: Annotate Movie Complex Type
Tarefa: Annotate Movie Complex Type with a Model Reference

Figura 39 – Interface gráfica de usuário para a criação e gerenciamento de tarefas. (a) Criação de uma nova tarefa (b) Listagem de tarefas



Fonte: Autoria própria.

Após os dois usuários terem acesso à mesma especificação WSDL de forma compartilhada, diferentes questões relacionadas aos elementos da especificação WSDL foram criadas pelos dois usuários na fase de *Alinhamento de Conhecimento*. A Figura 40a ilustra a interface gráfica de usuário para a criação de uma questão. Ao todo, cinco questões foram criadas. Já a Figura 40b ilustra a interface gráfica para a listagem de questões criadas para o serviço. O Quadro 5.2 apresenta as questões criadas nesta etapa.

Quadro 5.2 - Questões do serviço MovieStore

(Q1) **Título da questão**: *Array of Awards?*
Questão: *Does it mean it is possible to have multiple awards for a single movie?*
Elemento WSDL/XSD associado à questão: [c-t] ArrayOfAward
Usuário da questão: mlcalache@gmail.com

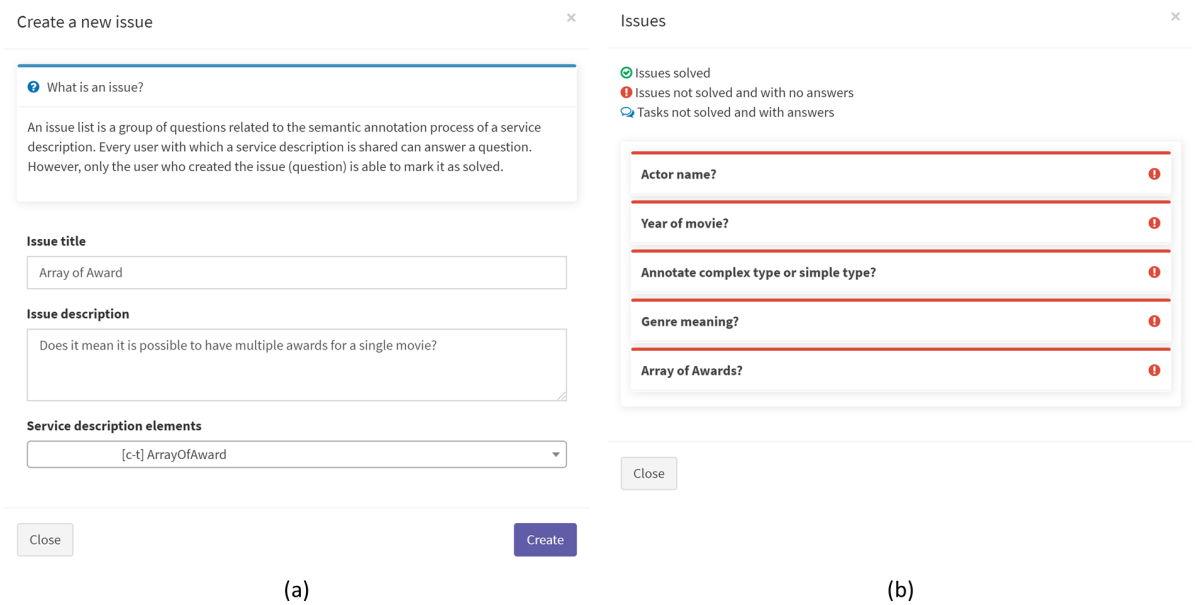
(Q2) **Título da questão**: *Genre meaning?*
Questão: *What genre stands for?*
Elemento WSDL/XSD associado à questão: [c-t] Genre
Usuário da questão: mlcalache@gmail.com

(Q3) **Título da questão**: *Annotate the complex type of the simple type?*
Questão: *Shoud I annotate the country complex type or the country name simple type?*
Elemento WSDL/XSD associado à questão: [c-t] Country
Usuário da questão: usuariograsews@gmail.com

(Q4) **Título da questão**: *Year of movie?*
Questão: *Is there any problem if we don't have an ontology term for the year of the movie? And what about other elements which we also don't have an ontology term?*
Elemento WSDL/XSD associado à questão: [s-t] Year
Usuário da questão: usuariograsews@gmail.com

(Q5) **Título da questão**: *Actor name?*
Questão: *Since we don't have a distinct ontology term for the first name and the last name, I suggest we annotate only the Actor element and leave the first and the last name with no annotation. What do you think?*
Elemento WSDL/XSD associado à questão: [s-t] FirstName
Usuário da questão: usuariograsews@gmail.com

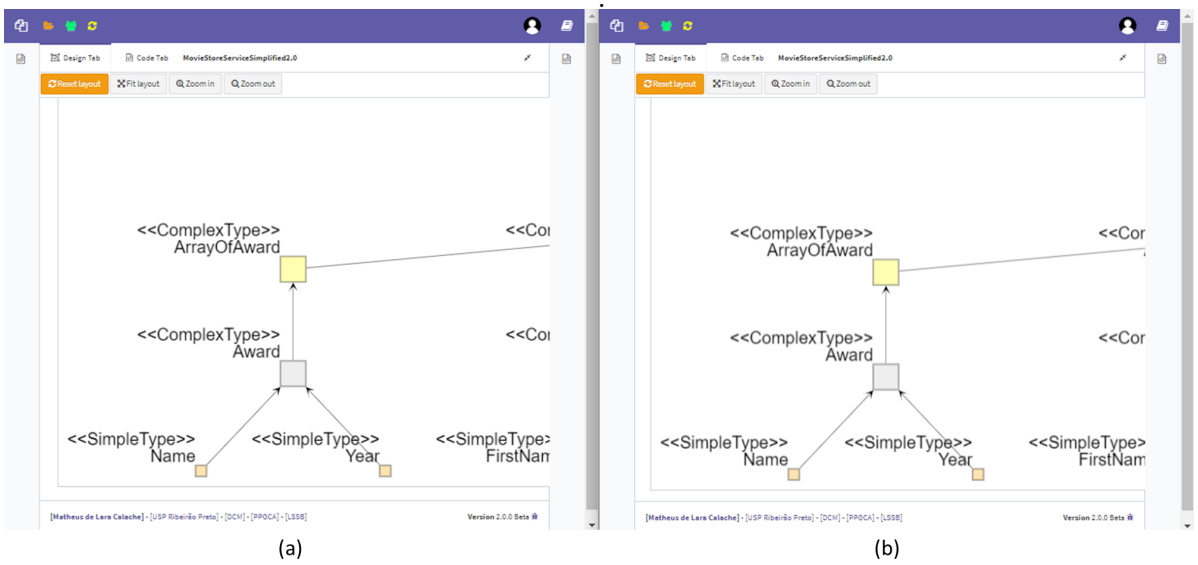
Figura 40 – Interface gráfica de usuário para a criação e gerenciamento de questões. (a) Criação de uma nova questão (b) Listagem de questões



Fonte: Autoria própria.

Logo que uma nova questão era criada por um usuário, o outro usuário recebia notificações acerca da nova questão. Adicionalmente, o grafo foi automaticamente atualizado a cada questão criada, contendo, portanto, a representação visual para os elementos WSDL/XSD com questões associadas. A Figura 41 ilustra o elemento `ArrayOfAward` com uma questão em aberto para o `Desenvolvedor`, no lado esquerdo, e para o `Especialista`, no lado direito. Ambos usuários possuem o mesmo estado da especificação WSDL e, simultaneamente, a mesma representação visual para o elemento `ArrayOfAward` que possui a questão relacionada. Adicionalmente, a figura também ilustra a notação visual para um elemento WSDL/XSD com uma questão (em amarelo).

Figura 41 – Representação visual do elemento `ArrayOfAward`, com questão, para ambos usuários (a) `Desenvolvedor` (b) `Especialista`



Fonte: Autoria própria.

Após a criação do conjunto de questões, ambos usuários adicionaram respostas a elas. O `Especialista` adicionou pelo menos uma resposta para cada questão criada pelo `Desenvolvedor`, enquanto que o `Desenvolvedor` adicionou pelo menos uma resposta para cada questão criada pelo `Especialista`. À medida em que as questões eram respondidas e consideradas resolvidas pelo usuário que as criou, estas eram marcadas como resolvidas. Ao fim do processo, todas as questões foram marcadas como resolvidas. O Quadro 5.3 as respostas dadas às questões criadas anteriormente nesta etapa.

Quadro 5.3 - Respostas das questões do serviço MovieStore

(R1) **Questão Q1**:
Resposta: *Yes. A movie can have multiple awards, such as Oscar and Golden Globe awards.*
Usuário da resposta: *usuariograsews@gmail.com*

(R2) **Questão Q2**:
Resposta: *This is the movie category, such as Drama, Comedy, SciFi, etc.*
Usuário da resposta: *usuariograsews@gmail.com*

(R3) **Questão Q3**:
Resposta: *Let's focus on the complex types.*
Usuário da resposta: *mlcalache@gmail.com*

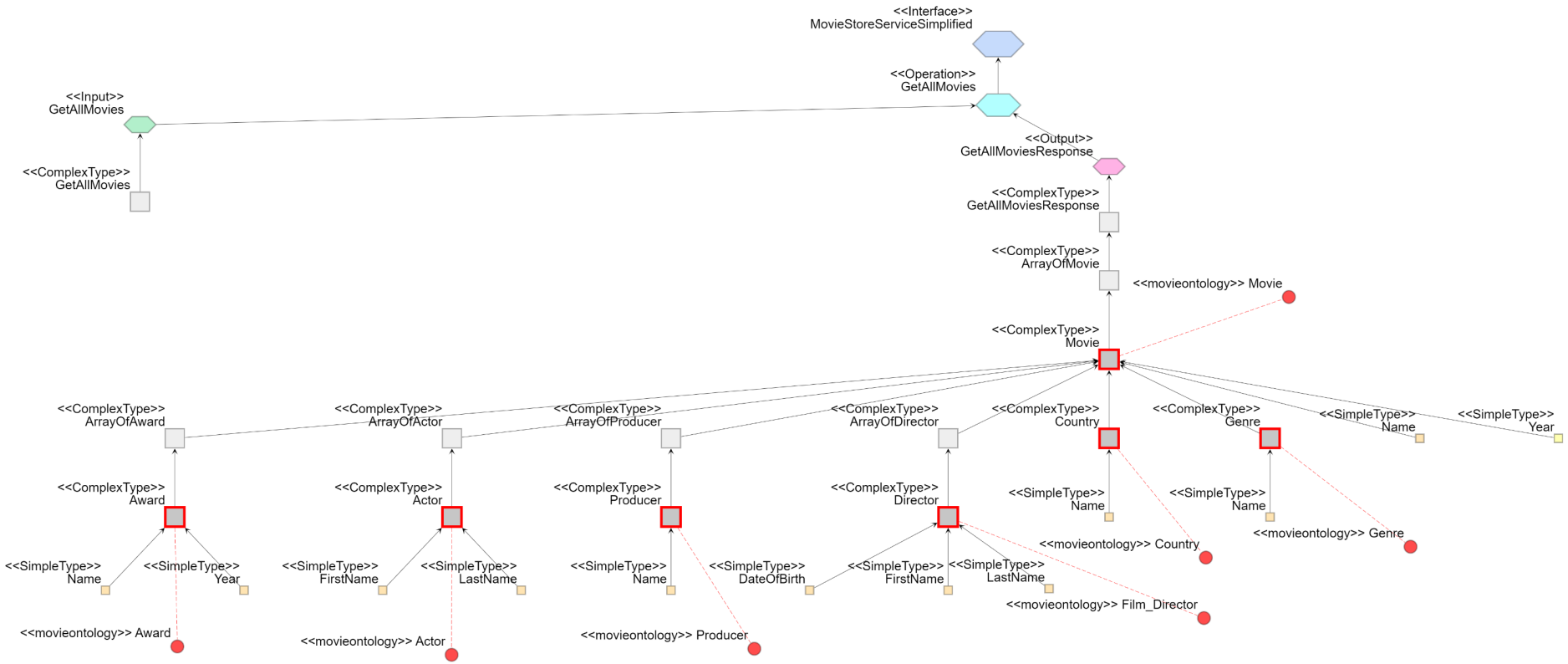
(R4) **Questão Q4**:
Resposta: *Let's skip the simple types for now. Let's focus on the complex types. I've created a list of tasks, which does not contemplate the simple types.*
Usuário da resposta: *mlcalache@gmail.com*

(R5) **Questão Q5**:
Resposta: *Let's skip the simple types for now. Let's focus on the complex types. I've created a list of tasks, which does not contemplate the simple types.*
Usuário da resposta: *mlcalache@gmail.com*

O último passo deste processo consiste da fase de *Anotação Semântica* da especificação WSDL. Ambos usuários trabalharam simultaneamente na anotação semântica conforme a lista de tarefas criada pelo `Desenvolvedor`. A anotação semântica foi feita tanto por meio do menu de contexto do grafo, quanto por meio do menu de contexto do menu *tree-view* WSDL e do menu *tree-view* OWL. À medida que as anotações eram feitas e validadas, cada tarefa era marcada como finalizada. Ao final do processo, todas as tarefas encontravam-se marcadas como finalizadas.

Ambos usuários foram capazes de anotar semanticamente a especificação WSDL. Dado que o domínio cinematográfico pode ser considerado relativamente simples, o `Desenvolvedor` foi também capaz de realizar anotações semânticas, com base na lista de tarefas previamente criada. Porém, isto não descartou a importância do `Especialista` durante o processo, pois este pôde contribuir efetivamente no processo de anotação semântica, também revisando e validando as anotações feitas pelo `Desenvolvedor`. A Figura 42

Figura 42 – Grafo da especificação WSDL anotada do serviço `MovieStore`.



Fonte: Autoria própria.

ilustra parte do grafo WSDL após o processo de anotação semântica.

5.4 Comentários Finais

Nesta prova de conceito, demonstramos o uso da ferramenta Grasews para a anotação semântica de duas especificações WSDL. A primeira especificação WSDL descreve o serviço `MicroAffyNorm`, enquanto que a segunda especificação descreve o serviço `MovieStore`. O processo foi inteiramente realizado por meio das notações visuais e funcionalidades providas por Grasews. Adicionalmente, trabalhamos de forma colaborativa para a anotação do serviço `MovieStore`, envolvendo um desenvolvedor de *software* e um especialista de domínio.

A utilização da ferramenta facilitou a anotação semântica da especificação WSDL utilizada nos dois casos. Na anotação do serviço `MicroAffyNorm`, algumas anotações semânticas criadas em elementos XSD diferiram quando comparadas com a especificação WSDL originalmente anotada, visto que Grasews foca na anotação da definição do tipo a ser anotado e não na utilização deste tipo. Na anotação do serviço `MovieStore`, como a especificação WSDL foi criada para o propósito deste trabalho, não foi possível realizar qualquer tipo de comparação dos resultados obtidos. Porém, ressaltamos que o foco do processo de anotação semântica deste serviço não foi de validar os conceitos ontológicos utilizados na anotação semântica, mas sim validar o processo de anotação semântica auxiliado por notações visuais e pelo trabalho colaborativo por meio da ferramenta Grasews. Ressaltamos que nenhum URI de transformação para os atributos *Lifting Schema Mapping* e *Lowering Schema Mapping* foi utilizado durante as duas etapas desta prova de conceito. Toda a documentação associada a esta prova de conceito encontra-se disponível *online*[1].

A utilização de notações visuais e o suporte à edição colaborativa foram diferenciais para o processo de anotação semântica realizado nesta prova de conceito. Por meio de notações visuais, tanto para a anotação do serviço `MicroAffyNorm` quanto para o serviço `MovieStore`, os usuários não tiveram que lidar diretamente com dados sintáticos provindos de padrões e tecnologias envolvidas na anotação semântica segundo a abordagem SAWSDL. O uso da notação visual proposta abstrai detalhes técnicos, permitindo que seus usuários trabalhem mais facilmente na anotação semântica de um serviço web. Já o suporte à edição colaborativa permitiu que dois usuários trabalhassem simultaneamente na anotação semântica do serviço `MovieStore`. Não só a carga de trabalho pôde ser distribuída entre os dois usuários, mas também contribuiu para que usuários com especializações complementares anotassem semanticamente o serviço web de foma mais efetiva. Acreditamos

[1] Documentação *online* do projeto. Disponível em https://grasewsweb.azurewebsites.net/projectdocumentation.

também que, em cenários mais complexos, a anotação semântica possa ser realizada mais rapidamente, com o possível envolvimento de um número maior de desenvolvedores e especialistas de domínio.

6

Conclusão

Este trabalho teve por objetivo a investigação do suporte ferramental para o desenvolvimento colaborativo de serviços web semânticos por meio de notações gráficas, segundo a abordagem SAWSDL. Este capítulo apresenta as principais contribuições deste trabalho, sua relevância quando comparado com outros trabalhos da área, suas principais limitações e, por fim, as possíveis direções futuras para esta pesquisa.

O capítulo está estruturado da seguinte forma: a seção 6.1 apresenta as principais contribuições deste trabalho; a seção 6.2 apresenta uma discussão acerca da relevância deste trabalho e suas limitações; e, por fim, a seção 6.3 apresenta os trabalhos futuros.

6.1 Principais Contribuições

Neste trabalho, propusemos uma (nova) notação visual para representar o padrão SAWSDL, fortemente aderente aos Princípios da Física das Notações. Propusemos um conjunto de elementos gráficos capazes de representar todos os elementos de uma especificação WSDL 2.0 passíveis de serem anotados com atributos SAWSDL. Propusemos também elementos gráficos para representar hierarquias de classes OWL, bem como elementos gráficos para representar os diferentes tipos de atributo SAWSDL.

Por meio desta notação visual, o processo de anotação semântica pode ser realizado de uma forma mais abstrata, sem se preocupar com detalhes sintáticos associados aos padrões e tecnologias relacionados ao processo de anotação semântica de serviços web segundo a abordagem SAWSDL. Neste sentido, o processo torna-se mais inclusivo, possibilitando que pessoas com diferentes especializações possam anotar semanticamente um serviço web, visto que não se faz mais necessário o conhecimento técnico acerca da abordagem de anotação semântica utilizada em questão.

De modo a prover suporte à anotação gráfica (visual) de um serviço web utilizando a anotação proposta por este trabalho, desenvolvemos a ferramenta *Graphical Annotation*

of Semantic Web Services (Grasews). A ferramenta Grasews suporta completamente a notação visual proposta para a abordagem SAWSDL. Esta ferramenta também provê suporte à edição colaborativa, permitindo que dois ou mais usuários, possivelmente com especializações complementares, possam colaborar para anotar semanticamente serviços web de forma simultânea e remota. Adicionalmente, Grasews é uma ferramenta web, o que facilita a sua distribuição.

Finalmente, uma prova de conceito envolvendo a anotação semântica de dois serviços web distintos foi realizada para validar tanto a notação visual proposta quanto o suporte à anotação gráfica colaborativa implementados por Grasews. O primeiro serviço web anotado foi o serviço `MicroAffyNorm` (GUARDIA et al., 2015), desenvolvido para a normalização de dados de *microarray one-color Affymetrix*. Para anotar semanticamente este serviço, utilizamos a ontologia OWL `GEXPASO`. Já o segundo serviço web anotado foi o serviço `MovieStore`, desenvolvido para o domínio cinematográfico. Para anotar semanticamente este serviço, utilizamos a ontologia OWL `MovieOntology` (BOUZA, 2010).

6.2 Discussão

Com o intuito de investigar o conhecimento e a utilização de tecnologias de serviços web semânticos entre profissionais de TI, realizamos uma pesquisa por meio da aplicação de um questionário simples. Esta pesquisa demonstrou que a web semântica de maneira geral e a abordagem SAWSDL em particular são ainda pouco conhecidas por este público. Embora não possamos atribuir este baixo conhecimento à falta de suporte ferramental adequado, o fato é que, atualmente, existe um número bastante limitado de ferramentas de suporte à anotação semântica segundo o padrão SAWSDL. O apêndice B apresenta o questionário e os resultados da pesquisa realizada.

As ferramentas atualmente disponíveis são geralmente utilizadas para anotar diretamente o código WSDL/XML das descrições de serviços web. A anotação direta de uma especificação WSDL pressupõe que seus usuários tenham conhecimento técnico acerca das tecnologias e padrões associados. Estas ferramentas também possuem limitações de distribuição, visto que nenhuma é disponibilizada como uma ferramenta web. Radiant apresenta problemas quanto à sua disponibilidade, haja vista a sua dependência da ferramenta Eclipse. Além disso, as distribuições de Radiant, atualmente encontradas na Internet, não são compatíveis com as versões mais recentes de Eclipse. Iridescent apresenta uma maior disponibilidade, porém apresentou erros (exceções não tratadas) durante o seu uso. Já EasySAWSDL, por se tratar de uma biblioteca associada a uma linguagem de programação específica (*Java*), possui usabilidade e aplicabilidade restritas quando comparada às demais soluções para SAWSDL. Finalmente, nenhuma destas ferramentas suporta o trabalho colaborativo.

Neste sentido, Grasews destaca-se devido à sua capacidade de abstrair detalhes técnicos do processo de anotação semântica, por meio do uso de notações visuais. Adicionalmente, o suporte para a anotação semântica colaborativa deve ser considerado um fator relevante, visto que o processo de anotação semântica, assim como o processo de desenvolvimento de um *software*, tem tornado-se cada vez mais complexo e, consequentemente, exigindo a atuação conjunta de equipes de profissionais para produzir soluções com qualidade, eficiência e eficácia. Por fim, por se tratar de uma ferramenta web, Grasews possui mais fácil distribuição quando comparado com as demais ferramentas existentes.

A forma com a qual a ferramenta Grasews anota semanticamente os elementos XSD de uma especificação WSDL, na declaração do tipo XSD ao invés de sua utilização, pode eventualmente dificultar a utilização do serviço web semântico criado. Porém, tal aparente limitação não invalida nossa contribuição uma vez não existe uma padronização acerca da anotação de um serviço e a utilização exata de um conjunto de anotações semânticas é específica ao propósito de uma determinada aplicação. Grasews pode ser futuramente adaptada para que contemple tanto a anotação na declaração de um tipo (atual abordagem) quanto na utilização deste tipo (adaptação).

A utilização de notações visuais e a edição colaborativa para o processo de anotação semântica foram vantagens consideradas relevantes durante a prova de conceito realizada neste trabalho. Porém, uma avaliação de usabilidade não foi realizada. Tal avaliação nos permitirá identificar possíveis limitações, bem como novas funcionalidades a serem incorporadas à ferramenta. Entretanto, a ferramenta foi testada e utilizada com sucesso na anotação semântica de diferentes serviços web, incluindo os exemplos reportados na prova de conceito.

6.3 Trabalhos Futuros

Futuramente, pretendemos realizar uma avaliação de usabilidade da ferramenta Grasews, envolvendo pelo menos dois grupos distintos de usuários para a anotação semântica de um ou mais serviços web. Neste processo, um grupo deverá anotar um dado serviço web utilizando a ferramenta Grasews, enquanto um segundo grupo deverá utilizar outra ferramenta para a anotação semântica deste mesmo serviço. Tal avaliação servirá para identificarmos os pontos negativos e positivos de Grasews em relação a outras ferramentas, seja pelas funcionalidades providas (notação visual e edição colaborativa), seja pela interface gráfica de usuário.

Diferentes funcionalidades podem também ser incorporadas futuramente na ferramenta Grasews. Inicialmente, pretendemos flexibilizar a anotação semântica de um tipo XSD, de modo a permitir que o usuário escolha se a anotação semântica será aplicada à

definição do tipo XSD ou diretamente à sua utilização. A importação de ontologias OWL também pode ser aprimorada. Grasews necessita realizar o *upload* de um arquivo OWL para que os conceitos definidos nesta ontologia sejam utilizados na anotação semântica de um dado serviço. Grasews não suporta o *upload* de ontologias extensas, como, por exemplo, a ontologia GO (Gene Ontology) (CONSORTIUM; ACENCIO, 2018). Adicionalmente, Grasews pode ser estendido de forma a permitir que os usuários customizassem as cores dos elementos representados no grafo. Por meio de uma paleta de cores, os usuários poderiam alterar as cores pré-definidas pela ferramenta, conforme a notação visual, para os elementos de uma especificação WSDL representados no grafo. Também, cada ontologia aberta e utilizada em anotações semânticas poderia ser representada por uma cor diferente, facilitando também a identificação de conceitos semânticos providos por diferentes ontologias no grafo. Por fim, a ferramenta Grasews poderia ser aprimorada por meio do suporte ao controle de versão das especificações WSDL anotadas. O controle de versão poderia permitir que seus usuários pudessem revisar as anotações semânticas criadas mais recentemente, comparando a versão mais recente com as versões anteriores da especificação WSDL (revisão de código). O controle de versão também poderia contribuir para que a ferramenta Grasews pudesse reverter a especificação WSDL para uma dada versão anterior (*rollback*), facilitando o descarte de anotações semânticas que possivelmente possam ser consideradas incorretas.

Finalmente, podemos investigar o suporte à anotação semântica utilizando outras abordagens, como, por exemplo, as abordagens WSMO (W3C, 2005a) e WSMO-Lite (W3C, 2010), contribuindo para tornar Grasews uma ferramenta mais completa e abrangente. Adicionalmente, podemos investigar o suporte a outros formatos de descrições de serviços, juntamente com o suporte a outras abordagens de anotação semântica para tais formatos. Além do formato WSDL, diferentes padrões de descrições de serviço podem ser utilizados para descrever um serviço web, como, por exemplo, o padrão OpenAPI (SMARTBEAR, 2017). Alguns trabalhos com o foco na anotação semântica de descrições deste formato têm sido propostos, como, por exemplo, Cha e Lee (CHA; LEE, 2013), Schwichtenberg, Gerth e Engels (SCHWICHTENBERG; GERTH; ENGELS, 2017) e Peng e Bai (PENG; BAI, 2018).

Referências

AUTO MAPPER. *Auto Mapper*. 2019. Disponível em: <https://automapper.org/>.

BELHAJJAME, K.; EMBURY, S. Verification of semantic web service annotations using ontology-based partitioning. IEEE Transactions on Services Computing, v. 7, p. 515–528, 2014.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. Scientific American: Feature Article, 2001.

BOOTSTRAP. *Bootstrap*. 2019. Disponível em: <https://getbootstrap.com/>.

BOUZA, A. *MO - The Movie Ontology*. 2010. [Online; 26. Jan. 2010]. Disponível em: <http://www.movieontology.org>.

CARDOSO, J.; MILLER, J.; EMANI, S. Web services discovery utilizing semantically annotated wsdl. In: . [S.l.: s.n.], 2008. p. 240–268.

CARDOSO, J.; SHETH, A. *Semantic Web Services, Processes and Applications*. [S.l.]: Springer, 2006.

CHA, S.-J.; LEE, K.-C. Advanced tagging and semantic-annotation methods for the semantic-based openapi retrieval system. In: . [s.n.], 2013. Disponível em: <https://www.semanticscholar.org/paper/Advanced-Tagging-and-Semantic-Annotation-Methods-Cha-Lee/652418e97c259ffb263e3c577271542e8057273e>.

CODEMIRROR. *CodeMirror*. 2019. Disponível em: <https://codemirror.net/>.

COLORLIB. *Admin LTE*. 2019. Disponível em: <https://adminlte.io/>.

CONSORTIUM, T.; ACENCIO, M. The gene ontology resource: 20 years and still going strong. *Nucleic Acids Research*, v. 49, p. gky1055, 11 2018.

DACONTA, M.; OBRST, L.; SMITH, K. The semantic web: A guide to the future of xml, web services, and knowledge management. Wiley Publishing Inc, 2003.

DAPPER. *Dapper ORM*. 2019. Disponível em: <https://dapper-tutorial.net/>.

DAVIES, W. V. Egyptian hieroglyphs. *Reading the past: ancient writing from cuneiform to the alphabet*, Univ of California Press, p. 75–135, 1990.

DENKER, G.; ELENIUS, D.; MARTIN, D. *OWL-S Editor*. 2016. Disponível em: <http://www.csl.sri.com/papers/owl-s-editor-demo/owlseditor-demo.pdf>.

DILLON, A. How collaborative is collaborative writing? an analysis of the production of two technical reports. In: . [S.l.: s.n.], 1993.

DOMAIN-DRIVEN DESIGN COMMUNITY. *Domain-Driven Design Community*. 2019. Disponível em: <https://dddcommunity.org/>.

DUTTA, A.; DEVI, M. S.; ARORA, M. Online census-based information sharing for delivery of e-governance services. In: *Proceedings of the Special Collection on eGovernment Innovations in India*. New York, NY, USA: ACM, 2017. (ICEGOV '17), p. 45–51. ISBN 978-1-4503-4930-7. Disponível em: <http://doi.acm.org/10.1145/3055219.3055226>.

EASYSAWSDL. *EasySAWSDL*. 2016. Disponível em: <http://easywsdl.ow2.org/extensions-sawsdl.html>.

EASYWSDL. *EasyWSDL*. 2016. Disponível em: <http://easywsdl.ow2.org/>.

ELENIUS, D. *The OWL-S Editor – A Domain-Specific Extension to Protégé*. 2005. Disponível em: <http://protege.stanford.edu/conference/2005/submissions/abstracts/accepted-abstract-elenius.pdf>.

ELLIS, C. A.; GIBBS, S. J.; REIN, G. L. Groupware: Some issues and experiences. *Commun. ACM*, v. 34, p. 39–58, 1991.

EVANS, E.; FOWLER, M. *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004. ISBN 9780321125217. Disponível em: <https://books.google.nl/books?id=7dlaMs0SECsC>.

FIELDING, R. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, University of California, 2000.

FONT-AWESOME. *Font-Awesome*. 2019. Disponível em: <https://fontawesome.com/v4.7.0/>.

FRANZ, M. et al. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, v. 32, n. 2, p. 309–311, 09 2015. ISSN 1367-4803. Disponível em: <https://doi.org/10.1093/bioinformatics/btv557>.

GAUTIER, L. et al. affyâ"analysis of Affymetrix GeneChip data at the probe level. *Bioinformatics*, v. 20, n. 3, p. 307–315, 02 2004. ISSN 1367-4803. Disponível em: <https://doi.org/10.1093/bioinformatics/btg405>.

GOOGLE. *Gmail*. 2019. Disponível em: <https://www.google.com/gmail/>.

GRUBER, T. A translation approach to portable ontology specifications. Journal Knowledge Acquisition – Special issue: Current issues in knowledge modeling, v. 5, p. 199–220, 1993.

GRUBER, T. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, v. 43, p. 907–928, 1995.

GUARDIA, G. D. et al. Semanticsco: A platform to support the semantic composition of services for gene expression analysis. *Journal of Biomedical Informatics*, v. 66, p. 116–128, 2017. ISSN 1532-0464. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1532046416301885>.

Referências

GUARDIA, G. D. A. et al. A methodology for the development of restful semantic web services for gene expression analysis. *PLOS ONE*, Public Library of Science, v. 10, n. 7, p. 1–28, 07 2015. Disponível em: <https://doi.org/10.1371/journal.pone.0134011>.

GUARINO, N. Formal ontology in information systems. Association for Computing Machinery – Digital Library, Proceedings of the 1st International Conference, IOS Press, 1998.

HEIMERL, F. et al. Word cloud explorer: Text analytics based on word clouds. In: *2014 47th Hawaii International Conference on System Sciences*. [S.l.: s.n.], 2014. p. 1833–1842.

HUANG, W. et al. Sens-e-motion: Capturing and visualising emotional status of computer users in real time. In: *2019 23rd International Conference in Information Visualization – Part II*. [S.l.: s.n.], 2019. p. 96–99.

IBM. *Service-oriented architecture (SOA)*. 2019. Disponível em: <https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html>.

IEEE. Internation standard iso/iec/ieee 42010: Systems and software engineering - architecture description. Switzerland. Primeira Edição, 2011.

JSON.ORG. *JSON*. 2019. Disponível em: <https://www.json.org/json-pt.html>.

KACI, A.; NACEF, A.; HENNI, A. Mobile cloud system for road safety. In: *Proceedings of the International Conference on Geoinformatics and Data Analysis*. New York, NY, USA: ACM, 2018. (ICGDA '18), p. 132–136. ISBN 978-1-4503-6445-4. Disponível em: <http://doi.acm.org/10.1145/3220228.3220233>.

KOPECKý, J. et al. Sawsdl: semantic annotations for wsdl and xml schema. *Internet Computing, IEEE*, v. 11, p. 60–67, 12 2007.

LOWRY, P. B.; CURTIS, A. M.; LOWRY, M. R. A taxonomy of collaborative writing to improve empirical research, writing practice, and tool development. In: . [S.l.: s.n.], 2004. v. 41, n. 1, p. 66–99.

MATHEWS, P. Classic maya emblem glyphs. *Classic Maya political history: Hieroglyphic and archaeological evidence*, Cambridge University Press: Cambridge, UK, p. 19–29, 1991.

MICROSOFT. *ASP.NET Identity*. 2019. Disponível em: <https://docs.microsoft.com/en-us/aspnet/identity/>.

MICROSOFT. *Microsoft Azure Cloud Computing Platform & Services*. 2019. Disponível em: <https://azure.microsoft.com/>.

MICROSOFT. *Microsoft Azure DevOps*. 2019. Disponível em: <https://azure.microsoft.com/en-us/services/devops/>.

MICROSOFT. *Microsoft Entity Framework*. 2019. Disponível em: <https://docs.microsoft.com/en-us/ef/>.

MICROSOFT. *Micrsoft ASP.NET MVC*. 2019. Disponível em: <https://dotnet.microsoft.com/apps/aspnet/mvc>.

MICROSOFT. *Micrsoft ASP.NET Web API*. 2019. Disponível em: <https://dotnet.microsoft.com/apps/aspnet/apis>.

MICROSOFT. *Micrsoft .NET Framework*. 2019. Disponível em: <https://dotnet.microsoft.com/download/dotnet-framework/net472>.

MICROSOFT. *Real-time ASP.NET with SignalR*. 2019. Disponível em: <https://dotnet.microsoft.com/apps/aspnet/signalr>.

MICROSOFT. *SQL Server*. 2019. Disponível em: <https://www.microsoft.com/en-us/sql-server/sql-server-2017>.

MILLER, J. et al. Radiant: A tool for semantic annotation of web services. In: *Radiant: A tool for semantic annotation of Web Services*. [S.l.: s.n.], 2005.

MOODY, D. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 35, n. 6, p. 756–779, nov. 2009. ISSN 0098-5589. Disponível em: <https://doi.org/10.1109/TSE.2009.67>.

NOVèRE, N. L. et al. The systems biology graphical notation. In: . [S.l.: s.n.], 2009. v. 27, p. 735–741.

OASIS. *UDDI Version 3.0.2*. 2004. Disponível em: <http://www.uddi.org/pubs/uddi-v3.0.2-20041019.pdf>.

OAUTH. *OAuth 2.0*. 2019. Disponível em: <https://oauth.net/>.

OMG. *Business Process Model and Notation (BPMN), Version 2.0*. 2011. Disponível em: <http://www.omg.org/spec/BPMN/2.0>.

OMG. *UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1*. 2017. Disponível em: <https://www.omg.org/spec/UML/2.5.1/>.

ORACLE. *Oracle*. 2019. Disponível em: <https://www.oracle.com/index.html>.

OW2. *OW2 Consortium*. 2016. Disponível em: <https://www.ow2.org>.

OWIN. *OWIN*. 2019. Disponível em: <http://owin.org/>.

OYUCU, S.; POLAT, H. Online video content analysis system. In: *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. [S.l.: s.n.], 2018. p. 1–5.

PAPAZOGLOU, M.; GEORGAKOPOULOS, D. Service oriented computing. Communications of the ACM, No. 10, v. 46, 2003.

PENG, C.; BAI, G. Using tag based semantic annotation to empower client and rest service interaction. In: *COMPLEXIS 2018 - Proceedings of the 3rd International Conference on Complexity, Future Information Systems and Risk :*. [s.n.], 2018. p. 64–71. ISBN 978-989-758-297-4. Disponível em: <http://www.scitepress.org/PublicationsDetail.aspx?ID=qDeSBDbcnqI%3d>.

PGADMIN. *pgAdmin*. 2019. Disponível em: <https://www.pgadmin.org/>.

Referências

PMI. *A Guide to the Project Management Body of Knowledge (PMBOK Guide), 6th Edition*. [S.l.]: Project Management Institute, 2017.

POPESCU, G.; WEGMANN, A. Using the physics of notations theory to evaluate the visual notation of seam. In: *2014 IEEE 16th Conference on Business Informatics*. [S.l.: s.n.], 2014. v. 2, p. 166–173.

POSTGRES. *Postgres*. 2019. Disponível em: <https://www.postgresql.org/>.

POSTMAN. *Postman*. 2019. Disponível em: <https://www.getpostman.com/>.

PROTéGé. *Protégé*. 2016. Disponível em: <http://protege.stanford.edu/>.

QUINN, J. Y. et al. Sbol visual: A graphical language for genetic designs. *PLOS Biology*, Public Library of Science, v. 13, n. 12, p. 1–9, 12 2015. Disponível em: <https://doi.org/10.1371/journal.pbio.1002310>.

RAO, N. K. Aspects of prehistoric astronomy in india. Astronomical Society of India, 2005.

SAADATI, S.; DENKER, G. *An OWL-S Editor Tutorial – Version 1.1*. 2016. Disponível em: <http://owlseditor.semwebcentral.org/documents/tutorial.pdf>.

SCHWICHTENBERG, S.; GERTH, C.; ENGELS, G. From open api to semantic specifications and code adapters. In: . [s.n.], 2017. Disponível em: <https://www.researchgate.net/publication/316653033_From_Open_API_to_Semantic_Specifications_and_Code_Adapters>.

SCICLUNA, J. *OWL-S Editor to Semantically Annotate Web-Services*. 2016. Disponível em: <http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html>.

SENTRY. *Sentry*. 2019. Disponível em: <https://sentry.io/>.

SHETH, A. *SAWSDL: Tools and Applications*. 2007. Disponível em: <https://www.w3.org/2007/Talks/www2007-SAWSDL-AmitSheth.pdf>.

SIMPLE INJECTOR. *Simple Injector*. 2019. Disponível em: <https://simpleinjector.org/>.

SMARTBEAR. *OpenAPI Specification (OAS) Version 3.0.2*. 2017. Disponível em: <https://swagger.io/specification/>.

SMARTBEAR. *Swagger*. 2019. Disponível em: <https://swagger.io/>.

SMITH, K. L. et al. *Handbook of visual communication: Theory, methods, and media*. [S.l.]: Routledge, 2004.

SOA Manifesto. *SOA Manifesto*. 2013. Disponível em: <http://www.soa-manifesto.org>.

SOUZA, C. R. B. de; MARCZAK, S.; PRIKLADNICKI, R. Desenvolvimento colaborativo de software. In: *Sistemas Colaborativos*. [S.l.]: Elsevier, 2012. cap. 8, p. 122–134.

STAVROPOULOS, T.; VRAKAS, D.; VLAHAVAS, I. Iridescent: a tool for rapid semantic annotation of web service descriptions. *ACM International Conference Proceeding Series*, 06 2013.

THE JQUERY FOUNDATION. *jQuery*. 2019. Disponível em: <https://jquery.com/>.

TOURé, V. et al. Quick tips for creating effective and impactful biological pathways using the systems biology graphical notation. *PLOS Computational Biology*, Public Library of Science, v. 14, n. 2, p. 1–6, 02 2018. Disponível em: <https://doi.org/10.1371/journal.pcbi.1005740>.

VALIPOUR, M. et al. A brief survey of software architecture concepts and service oriented architecture. *2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, p. 34–38, 2009.

VITVAR, T. et al. Wsmo-lite annotations for web services. The Semantic Web: Research and Applications. 5th European Semantic Web Conference, ESWC 2008, p. 674–689, 2008.

W3C. *XSL Transformations (XSLT)*. 1999. Disponível em: <https://www.w3.org/TR/xslt>.

W3C. *Web Services Description Language (WSDL) 1.1*. 2001. Disponível em: <https://www.w3.org/TR/wsdl.html>.

W3C. *OWL Web Ontology Language for Services (OWL-S)*. 2004. Disponível em: <https://www.w3.org/Submission/OWL-S/>.

W3C. *RDFS*. 2004. Disponível em: <https://www.w3.org/2001/sw/wiki/RDFS>.

W3C. *Web Service Modeling Ontology (WSMO)*. 2005. Disponível em: <https://www.w3.org/Submission/WSMO/>.

W3C. *Web Service Semantics - WSDL-S*. 2005. Disponível em: <https://www.w3.org/Submission/WSDL-S/>.

W3C. *Latest SOAP versions*. 2007. Disponível em: <https://www.w3.org/TR/soap/>.

W3C. *Semantic Annotations for WSDL and XML Schema*. 2007. Disponível em: <https://www.w3.org/TR/sawsdl/>.

W3C. *Web Services Description Language (WSDL) Version 2.0*. 2007. Disponível em: <https://www.w3.org/TR/wsdl20/>.

W3C. *SPARQL Query Language for RDF*. 2008. Disponível em: <https://www.w3.org/TR/rdf-sparql-query/>.

W3C. *Web Application Description Language*. 2009. Disponível em: <https://www.w3.org/Submission/wadl/>.

W3C. *WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web*. 2010. Disponível em: <https://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/>.

W3C. *OWL*. 2012. Disponível em: <https://www.w3.org/OWL/>.

W3C. *OWL 2.0 Web Ontology Language Primer (Second Edition)*. 2012. Disponível em: <https://www.w3.org/TR/2012/REC-owl2-primer-20121211/>.

W3C. *RDF*. 2014. Disponível em: <https://www.w3.org/2001/sw/wiki/RDF>.

W3C. *XQuery 3.1: An XML Query Language*. 2017. Disponível em: <https://www.w3.org/TR/2017/REC-xquery-31-20170321/>.

WALI, B.; GIBAUD, B. Extending owl-s for the composition of web services generated with a legacy application wrapper. ICIW 2012, The Seventh International Conference on Internet and Web Applications and Services, p. 97–105, 5 2012.

Apêndices

A

Tecnologias e Bibliotecas de Desenvolvimento

As principais tecnologias utilizadas no desenvolvimento de Grasews incluem:

1. **Microsoft .NET Framework**

   .NET *framework* (MICROSOFT, 2019g) é um *framework* de desenvolvimento provido pela *Microsoft*. Todos os módulos e componentes da ferramenta Grasews foram desenvolvidas utilizando a linguagem de programação C# da versão 4.7 do *framework* .NET. Os tipos de projetos .NET variam entre *Asp.NET MVC*, para o módulo `Grasews.Web`, *Asp.Net Web API*, para o módulo `Grasews.API`, e *Class Library*, para os demais módulos da ferramenta.

2. **Microsoft ASP.NET MVC**

   ASP.NET (MICROSOFT, 2019e) fornece recursos tecnológicos baseados em padrões para criar *websites* dinâmicos usando o padrão *Model-View-Controller* (MVC). MVC é um padrão de projeto usado para desacoplar a interface do usuário (visualização), os dados (modelo) e a lógica de aplicativo (controlador). Esse padrão ajuda a alcançar a separação de responsabilidades de forma mais clara.

   Ao usar o padrão MVC para o desenvolvimento de sites, as solicitações são encaminhadas para um *controller* responsável por trabalhar com um *model* para executar ações e/ou recuperar dados. Um *controller* escolhe a *view* para exibir e fornece o modelo (*model*) para a página. Por fim, uma *view* renderiza a página final com base nos dados de um *model*. O módulo `Grasews.Web` foi desenvolvido utilizando este recurso.

3. **Microsoft ASP.NET Web API**

   ASP.NET Web API (MICROSOFT, 2019f) é o formato mais recente utilizado para o desenvolvimento de serviços web RESTful do *framework* .NET. O módulo *Grasews.API* foi desenvolvido utilizado este recurso.

4. **Microsoft .NET Entity Framework**

   Entity Framework (MICROSOFT, 2019d) é um *Object Relational Mapper* (ORM) que permite que desenvolvedores trabalhem com um banco de dados por meio de objetos (instâncias de classes). Isso elimina a necessidade da maior parte do código de acesso a dados que os desenvolvedores geralmente precisam escrever, como, por exemplo, sentenças SQL (*queries*). O módulo `Grasews.Postgres` utiliza *.NET Entity Framework* para o acesso a dados existentes no SGBD *Postgres*.

5. **Simple Injector**

   Simple Injector (SIMPLE INJECTOR, 2019) é uma biblioteca de injeção de dependência de código aberto para o *framework* NET. O módulo `Grasews.IoC` é responsável por controlar as injeções de dependências, suportado pela biblioteca *Simple Injector*.

6. **AutoMapper**

   AutoMapper (AUTO MAPPER, 2019) é uma biblioteca criada para mapear um objeto para outro. Ao invés de criar um método responsável por mapear propriedades de um objeto para propriedades de outro objeto, uma a uma, *AutoMapper* simplifica o código, dado que esta biblioteca o provê o padrão de projeto *Adapter*. *AutoMapper* é utilizado na conversão entre objetos `Models` dos módulos `Grasews.Web` e `Grasews.API`, objetos `DTOs` do módulo `Grasews.Application` e entidades de domínio do módulo `Grasews.Domain`.

7. **OWIN**

   OWIN (OWIN, 2019) é uma uma interface padrão de desenvolvimento entre o lado servidor e o lado cliente de uma aplicação desenvolvida com o *framework* .NET. *OWIN* tem o propósito de desacoplar serviços do lado servidor e aplicativos Web desenvolvidos utilizando o *framework* .NET, permitindo uma melhor comunicação entre os módulos `Grasews.Web` e `Grasews.API`.

8. **OAuth 2.0**

   OAuth 2.0 (OAUTH, 2019) é o protocolo padrão para autorização de aplicações. *OAuth* 2.0 fornece fluxos de autorização específicos para aplicativos da web, dispositivos móveis e computadores. Os módulos `Grasews.Web` e `Grasews.API` utilizam *OAuth* 2.0 para o controle de autenticação e autorização de um usuário de Grasews.

9. **Microsoft ASP.NET Identity**

   Microsoft ASP.NET Identity (MICROSOFT, 2019a) é uma biblioteca de desenvolvimento do *framework* .NET que provê suporte a controles de segurança de uma aplicação. Por meio de *Microsoft ASP.NET Identity*, Grasews possui integração com o protocolo *OAuth* e a interface *OWIN*. Os módulos `Grasews.Web`, `Grasews.API` e `Grasews.Security` utilizam a biblioteca *Microsoft ASP.NET Identity*.

10. **Postgres**

    PostgreSQL (POSTGRES, 2019) é um sistema banco de dados relacional de objetos de código aberto. O módulo `Grasews.Postgres` utiliza bibliotecas de desenvolvimento que proveem conectividade entre *Entity Framework* e *Postgres*.

11. **Sentry**

    Sentry (SENTRY, 2019) fornece uma plataforma de monitoramento de erros baseada na nuvem. *Sentry* possui um conjunto de bibliotecas disponíveis para o *framework* .NET e outras linguagens o qual torna possível capturar erros de uma aplicação e enviar as especificações dos erros para a ferramenta da web. Por meio da ferramenta da web de *Sentry*, o monitoramento e gerenciamento de erros é facilitado. Equipes de desenvolvimento podem mais facilmente descobrir, filtrar e priorizar erros em tempo real. Todos os módulos de Grasews utilizam *Sentry* para a captura de exceções.

12. **Cytoscape.js**

    Cytoscape.js (FRANZ et al., 2015) é uma biblioteca *JavaScript* para a visualização e análise de grafos. O módulo `Grasews.Web` utiliza esta biblioteca para a criacão do grafo na interface de usuário. Adicionalmente, o módulo `Grasews.Cytoscape` é responsável por manipular e construir objetos (modelos), por meio da linguagem de programação C#, utilizados pela biblioteca *Cytoscape.js*.

13. **Admin LTE**

    Admin LTE (COLORLIB, 2019) é um modelo de interface de usuário utilizado para a construção de painéis de administração. *Admin LTE* possui o código aberto e disponibiliza diversas variações, customizações e tema de painel de controle. *Admin LTE* fornece uma variedade de componentes responsivos, reutilizáveis e comumente usados em sistemas web. *Admin LTE* é construído sobre o *Bootstrap*. O módulo `Grasews.Web` utiliza *Admin LTE* em sua interface gráfica de usuário.

14. **Bootstrap**

    Bootstrap (BOOTSTRAP, 2019) é uma biblioteca de código aberto para o desenvolvimento de aplicações web com HTML, CSS e *JavaScript*. *Bootstrap* fornece um conjunto de componentes responsivos para diversos dispositivos, extensos componentes pré-construídos e *plugins* criados no *jQuery*. O módulo `Grasews.Web` utiliza *Bootstrap*, em conjunto com *Admin LTE* na implementação da interface gráfica de usuário.

15. **jQuery**

    jQuery (THE JQUERY FOUNDATION, 2019) é uma biblioteca *JavaScript* que simplifica a manipulação de documentos HTML, a manipulação de eventos, animações e o gerenciamento de requisições AJAX. O módulo `Grasews.Web` utiliza *jQuery* para a construção de funcionalidades da interface gráfica de usuário de Grasews.

16. **Font-Awesome**

    Font-Awesome (FONT-AWESOME, 2019) é uma biblioteca de ícones vetoriais e escaláveis que podem ser personalizados instantaneamente. Tamanho, cor, sombreamento e outras propriedades dos ícones podem ser facilmente modificadas por meio de CSS. O módulo `Grasews.Web` utiliza ícones providos pela versão 4.7 de *Font-Awesome* para a construção da interface gráfica de usuário de Grasews.

17. **CodeMirror**

    CodeMirror (CODEMIRROR, 2019) é um editor de texto implementado utilizando a linguagem *JavaScript*. *CodeMirror* possui vários modos que dão suporte à manipulação de diversas linguagens de programação. Adicionalmente, *CodeMirror* possui um conjunto de temas CSS que possibilitam customizar o editor de texto. O módulo `Grasews.Web` utiliza a biblioteca *CodeMirror* no painel de exibição do código WSDL de uma descrição de serviço web na interface gráfica de usuário de Grasews.

18. **Microsoft SignalR**

    Microsoft SignalR (MICROSOFT, 2019h) é parte do *framework* .NET que tem como foco a implementação de *web-sockets*. Por meio de *Microsoft SignalR*, aplicações web desenvolvidas em .NET (lado cliente) podem receber atualizações do lado servidor de forma passiva. *Microsoft SignalR* implementa o padrão de projeto *Observer*. Com isso, o lado cliente permanece observando mensagens que são registradas no lado servidor por meio de canais de comunicação denominados *hubs*. Quando uma nova mensagem é registrada em um *hubs*, o lado cliente responde à essa mensagem. *Microsoft SignalR* permite que o módulo `Grasews.Web` atualize os ambientes de trabalho dos usuários que estão trabalhando de forma colaborativa em uma especificação WSDL aberta na ferramenta. Adicionalmente, além das atualizações dos ambientes de trabalho, os

usuários recebem notificações informando-os sobre novas atualizações em uma dada especificação WSDL.

19. **Swagger OpenAPI Specification**

    Swagger (SMARTBEAR, 2019) é uma estrutura de software de código aberto que apoia desenvolvedores a projetar, criar, documentar e consumir serviços da Web RESTful. O módulo `Grasews.API` provê uma documentação web (*help pages*) com a descrição de suas funcionalidades web por meio do formato *OpenAPI*, automaticamente criado pelo *framework* .NET.

20. **JSON**

    JavaScript Object Notation (JSON) (JSON.ORG, 2019) é um formato padrão para troca de dados. JSON é de fácil compreensão tanto para humanos quanto para máquinas. Este formato é completamente independente de linguagens de programação. Toda comunicação realizada entre os módulos `Grasews.Web` e `Grasews.API` é realizada por meio de mensagens no formato JSON. Adicionalmente, a construção do grafo por meio da biblioteca *Cytoscape.js* também é feita por meio de objetos no formato JSON.

B

Pesquisa de Conhecimento Geral de Web Semântica

Realizamos uma pesquisa com o intuito de investigar o conhecimento e a utilização de tecnologias de serviços web semânticos entre profissionais de TI. Por meio desta pesquisa, pudemos ter uma visão mais clara acerca da baixa adoção do desenvolvimento de serviços web semânticos entre estes profissionais.

Este apêndice está estruturado da seguinte forma: a seção B.1 lista as questões do formulário desta pesquisa e a seção B.2 apresenta os resultados do questionário desta pesquisa.

B.1 Questionário

Q1 **Qual é a sua formação acadêmica?**

.....

Q2 **Qual é a sua profissão?**

.....

Q3 **Como você classifica o seu conhecimento acerca de desenvolvimento de serviços web?**

( ) Muito pouco ( ) Pouco ( ) Intermediário ( ) Avançado ( ) Muito avançado

Q4 **Como você classifica o seu conhecimento acerca de desenvolvimento de serviços web semânticos?**

( ) Muito pouco ( ) Pouco ( ) Intermediário ( ) Avançado ( ) Muito avançado

Q5 **Como você classifica o seu conhecimento acerca de descrições de serviços web utilizando a linguagem WSDL 2.0?**

( ) Muito pouco ( ) Pouco ( ) Intermediário ( ) Avançado ( ) Muito avançado

Q6 **Como você classifica o seu conhecimento acerca de ontologias OWL?**
( ) Muito pouco ( ) Pouco ( ) Intermediário ( ) Avançado ( ) Muito avançado

Q7 **Como você classifica o seu conhecimento acerca de anotações semânticas utilizando o padrão SAWSDL?**
( ) Muito pouco ( ) Pouco ( ) Intermediário ( ) Avançado ( ) Muito avançado

Q8 **Você já utilizou alguma ferramenta de suporte à anotação semântica de serviços web?**
( ) Sim ( ) Não

Q9 **Se já utilizou alguma ferramenta de suporte à anotação semântica de serviços web, qual ferramenta utilizou?**
( ) Não se aplica ( ) Iridescent ( ) EasySAWSDL ( ) Radiant
( ) Outra. Especifique: .....

B.2 Resultados

O questionário criado por meio da plataforma web *SurveyMonkey*[1], em 26 de Novembro de 2019, foi enviado para diversos grupos da área de TI, bem como individualmente para profissionais de TI (aproximadamente quatrocentos e cinquenta (450) pessoas ao todo). O questionário foi fechado na data de 23 de Janeiro de 2020. Até o seu fechamento, foram coletadas respostas de quarenta e quatro (44) pessoas.

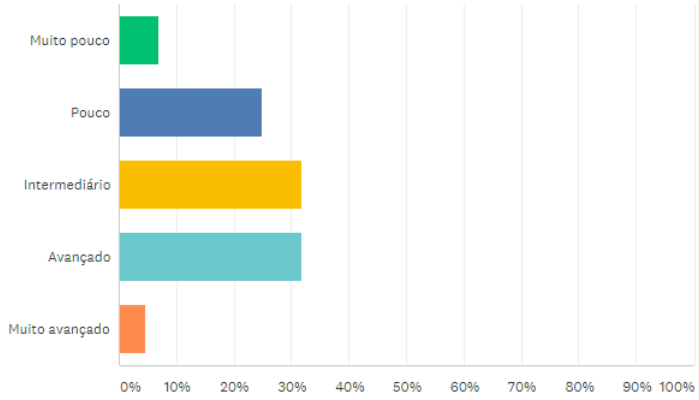
Dentre as quarenta e quatro (44) pessoas, todas possuem ensino superior, variando entre tecnólogos, bacharéis e pós-graduados. Em relação às profissões exercidas por estas pessoas, os cargos variam entre engenheiros de *software*, empresários, servidores públicos, professores, gerentes de TI, analistas de sistemas, analistas de implantação, gerentes de projetos, analistas de requisitos, analistas de testes, analistas de infraestrutura e cientistas de dados.

A Figura 43 ilustra as respostas coletadas para a questão Q3: *"Como você classifica o seu conhecimento acerca de desenvolvimento de serviços web?"*. Dentre as respostas, dezesseis (16) pessoas responderam que possuem conhecimento avançado ou muito avançado, correspondendo a 36,37% das respostas. Apenas três (3) pessoas possuem muito pouco conhecimento (apenas 6,82%).

A Figura 44 ilustra as respostas coletadas para a questão Q4: *"Como você classifica o seu conhecimento acerca de desenvolvimento de serviços web semânticos?"*. Dentre as

[1] Questionário online realizado por meio da plataforma SurveyMonkey. Disponível em https://pt.surveymonkey.com/

Figura 43 – Respostas para a questão Q3 do questionário B.1. "Como você classifica o seu conhecimento acerca de desenvolvimento de serviços web?"



ANSWER CHOICES	RESPONSES	
Muito pouco	6.82%	3
Pouco	25.00%	11
Intermediário	31.82%	14
Avançado	31.82%	14
Muito avançado	4.55%	2
TOTAL		44

Fonte: Autoria própria.

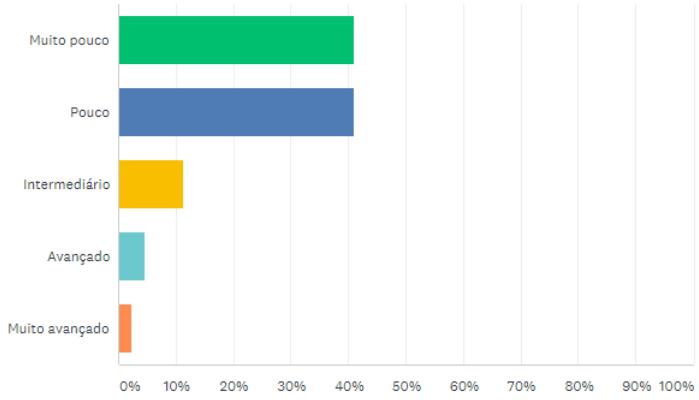
respostas, trinta e seis (36) pessoas responderam que possuem pouco ou muito pouco conhecimento, o que corresponde a 80,38% das respostas. Apenas três (3) pessoas responderam que possuem conhecimento avançado ou muito avançado (apenas 6,82%).

A Figura 45 ilustra as respostas coletadas para a questão Q5: *"Como você classifica o seu conhecimento acerca de descrições de serviços web utilizando a linguagem WSDL 2.0?"*. Dentre as respostas, nenhuma pessoa possui conhecimento muito avançado. Apenas uma (1) pessoa possui conhecimento avançado (2,27%).

A Figura 46 ilustra as respostas coletadas para a questão Q6: *"Como você classifica o seu conhecimento acerca de ontologias OWL?"*. Dentre as respostas, nenhuma pessoa possui conhecimento avançado ou muito avançado acerca de ontologias OWL. Trinta e uma (31) respostas foram que possuem muito pouco conhecimento (70,45%).

A Figura 47 ilustra as respostas coletadas para a questão Q7: *"Como você classifica o seu conhecimento acerca de anotações semânticas utilizando o padrão SAWSDL?"*. Dentre as respostas, nenhuma pessoa possui conhecimento muito avançado acerca do padrão SAWSDL. Quarenta e uma (41) pessoas responderam que possuem pouco ou muito pouco conhecimento (95,34%). Com isso também podemos concluir o quanto a adoção do desenvolvimento de serviços web semânticos ainda encontra-se baixa pela comunidade de desenvolvedores de *software*.

Figura 44 – Respostas para a questão Q4 do questionário B.1. "Como você classifica o seu conhecimento acerca de desenvolvimento de serviços web semânticos?"



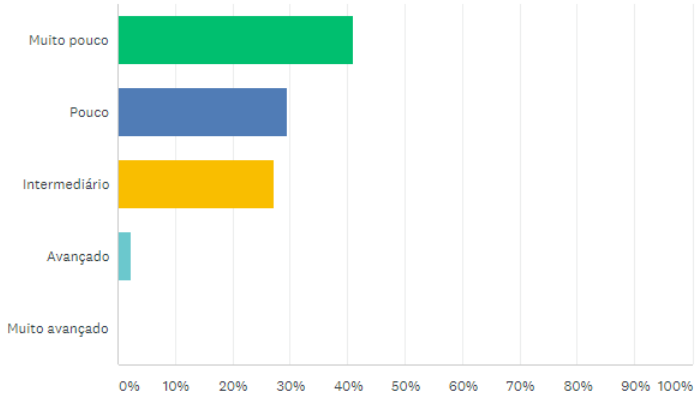
ANSWER CHOICES	RESPONSES	
Muito pouco	40.91%	18
Pouco	40.91%	18
Intermediário	11.36%	5
Avançado	4.55%	2
Muito avançado	2.27%	1
TOTAL		44

Fonte: Autoria própria.

A Figura 48 ilustra as respostas coletadas para a questão Q8: *"Você já utilizou alguma ferramenta de suporte à anotação semântica de serviços web?"*. Dentre as respostas, trinta e nove (39) responderam que nunca utilizaram alguma ferramenta de suporte à anotação semântica (88,64%). Este resultado já era esperado, visto que os desenvolvedores de *software* pouco tem utilizado descrições de serviços e pouco conhecem acerca do desenvolvimento de serviços web semânticos.

Por fim, a Figura 49 ilustra as respostas coletadas para a questão Q9: *"Se já utilizou alguma ferramenta de suporte à anotação semântica de serviços web, qual ferramenta utilizou?"*. Como 88,64% das respostas da questão Q8 disseram que as pessoas nunca utilizaram uma ferramenta de suporte à anotação semântica, o resultado para esta questão Q9 é de 93.18% das respostas não se aplicarem. Entretando, observa-se que uma (1) pessoa já utilizou a ferramenta Iridescent (STAVROPOULOS; VRAKAS; VLAHAVAS, 2013) e duas (2) pessoas já utilizaram a biblioteca EasySAWSDL (EASYSAWSDL, 2016).

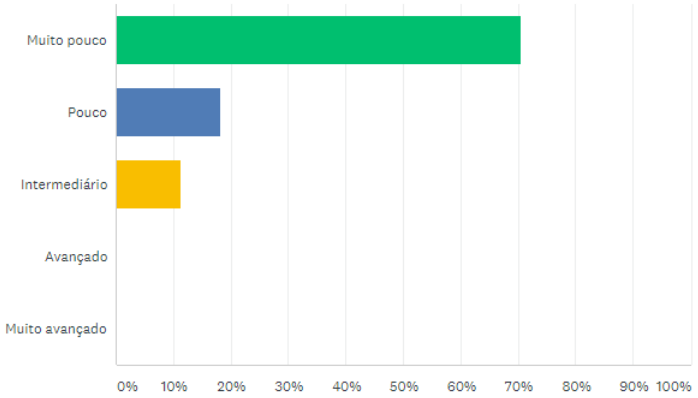
Figura 45 – Respostas para a questão Q5 do questionário B.1. "Como você classifica o seu conhecimento acerca de descrições de serviços web utilizando a linguagem WSDL 2.0?"



ANSWER CHOICES	RESPONSES	
Muito pouco	40.91%	18
Pouco	29.55%	13
Intermediário	27.27%	12
Avançado	2.27%	1
Muito avançado	0.00%	0
TOTAL		44

Fonte: Autoria própria.

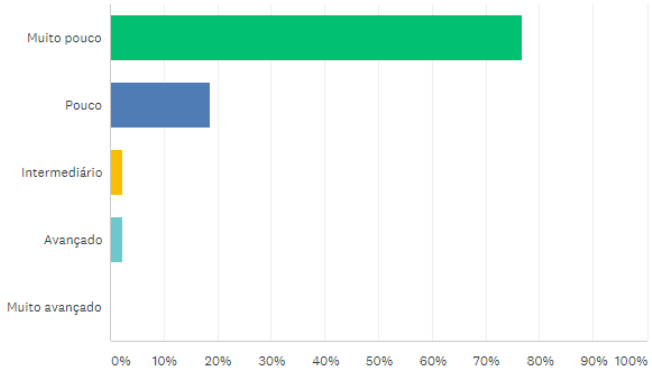
Figura 46 – Respostas para a questão Q6 do questionário B.1. "Como você classifica o seu conhecimento acerca de ontologias OWL?"



ANSWER CHOICES	RESPONSES	
Muito pouco	70.45%	31
Pouco	18.18%	8
Intermediário	11.36%	5
Avançado	0.00%	0
Muito avançado	0.00%	0
TOTAL		44

Fonte: Autoria própria.

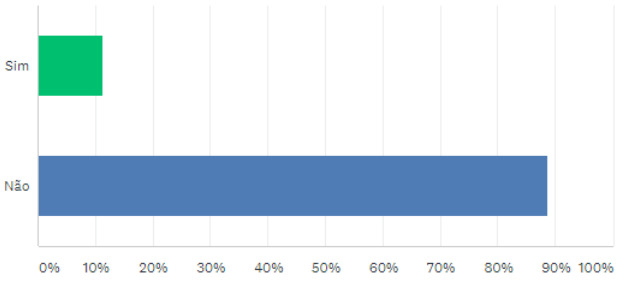
Figura 47 – Respostas para a questão Q7 do questionário B.1. "Como você classifica o seu conhecimento acerca de anotações semânticas utilizando o padrão SAWSDL?"



ANSWER CHOICES	RESPONSES	
Muito pouco	76.74%	33
Pouco	18.60%	8
Intermediário	2.33%	1
Avançado	2.33%	1
Muito avançado	0.00%	0
TOTAL		43

Fonte: Autoria própria.

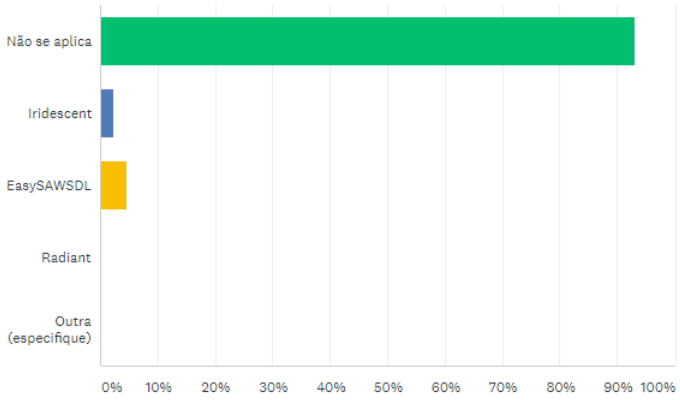
Figura 48 – Respostas para a questão Q8 do questionário B.1. "Você já utilizou alguma ferramenta de suporte à anotação semântica de serviços web?"



ANSWER CHOICES	RESPONSES	
Sim	11.36%	5
Não	88.64%	39
TOTAL		44

Fonte: Autoria própria.

Figura 49 – Respostas para a questão Q9 do questionário B.1. "Se já utilizou alguma ferramenta de suporte à anotação semântica de serviços web, qual ferramenta utilizou?"



ANSWER CHOICES		RESPONSES	
Não se aplica		93.18%	41
Iridescent		2.27%	1
EasySAWSDL		4.55%	2
Radiant		0.00%	0
Outra (especifique)	Responses	0.00%	0
TOTAL			44

Fonte: Autoria própria.