Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Adicionando temporalidade à linguagem OWL 2: um estudo a partir da linguagem tOWL e sua decibilidade

Déborah Mendes Ferreira

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Flávio de Barros Vidal

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenadora: Prof.ª Dr.ª Célia Ghedini Ralha

Banca examinadora composta por:

Prof. Dr. Flávio de Barros Vidal (Orientador) — CIC/UnB
Prof. Dr. André Santanchè — DSI/IC/Unicamp
Prof. Dr. Li Weigang — CIC/UnB
Prof. Dr. Eduardo Adilio Pelinson Alchieri - Suplente — CIC/UnB

CIP — Catalogação Internacional na Publicação

Ferreira, Déborah Mendes.

Adicionando temporalidade à linguagem OWL 2: um estudo a partir da linguagem tOWL e sua decibilidade / Déborah Mendes Ferreira. Brasília : UnB, 2016.
92 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2016.

1. ontologia, 2. tOWL, 3. temporalidade, 4. decidibilidade,
5. raciocínio automático

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Adicionando temporalidade à linguagem OWL 2: um estudo a partir da linguagem tOWL e sua decibilidade

Déborah Mendes Ferreira

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Flávio de Barros Vidal (Orientador)
CIC/UnB

Prof. Dr. André Santanchè
DSI/IC/Unicamp

Prof. Dr. Li Weigang
CIC/UnB

Prof. Dr. Eduardo Adilio Pelinson Alchieri - Suplente
CIC/UnB

Prof.ª Dr.ª Célia Ghedini Ralha
Coordenadora do Mestrado em Informática

Brasília, 01 de março de 2016

Dedicatória

Dedico este trabalho ao meu pai por ter sido sempre meu herói.

"Time is
Too slow for those who Wait,
Too swift for those who Fear,
Too long for those who Grieve,
Too short for those who Rejoice,
But for those who Love,
Time is not." - Henry van Dyke

Agradecimentos

"The heart may be weak. And sometimes it may even give in. But I've learned... that deep down there's a light that never goes out!"

– Sora - *Kingdom Hearts*

Agradeço ao professor Flávio Vidal, por me orientar e confiar na minha capacidade de realizar este trabalho. Professor, obrigada não só pela sua orientação na tese, mas também pelos bons conselhos.

Agradeço à minha família por terem me apoiado durante estes dois anos de estudo. Sem vocês eu não teria forças para enfrentar todas as dificuldades deste período. Pai, obrigada por todas as vezes que me salvou, você me entende melhor que ninguém. Mãe, obrigada por curar todas minhas feridas com seu carinho, você é minha inspiração. Brendha, obrigada por ser minha melhor amiga e meu anjo. Michela, sei que posso contar com você sempre, obrigada por me escutar. Nícolas, obrigada por sempre me ajudar quando precisei. Tiago, obrigada por me ensinar que sou capaz de aprender qualquer coisa, isso me impediu de desistir nas horas difíceis. Mari e Eliza, vocês são maravilhosas, obrigada por estarem em nossas vidas. Pedro, obrigada por todas as risadas, você é demais. Lucas, obrigada pelo melhor presente de todos: meu afilhado.

Agradeço aos meus queridos amigos Geovanna, Luquinhas, Maysa, Pablo e Victor por estarem ao meu lado, por sempre me motivarem e por escutarem meus desabafos nos momentos difíceis. Não importa onde eu estiver, vocês sempre estarão no meu coração, contem comigo!

Alber, obrigada pelos momentos especiais. Du bist mein, ich bin dein.

Resumo

Um dos maiores obstáculos para o fornecimento de melhor suporte para os usuários da Web é o fato de que o significado do conteúdo da maior parte da Web não ser acessível às máquinas. Para que as máquinas consigam assimilar o conteúdo da Web, máquinas e humanos precisam compartilhar conhecimento à respeito do mundo real, ou seja, é necessário ser capaz de representar o mundo, ou partes dele, dentro dos computadores.

Ao representar o mundo, é desejável que tal representação seja o mais próxima possível da realidade para evitar que falsas suposições sejam feitas à respeito dele. Para que isso ocorra, temos que ser capazes de representar também um aspecto muito importante do mundo real: o tempo. Tempo é um aspecto muito importante da vida humana, muitos ambientes exigem uma consciência temporal.

Neste trabalho apresentamos um estudo da compatibilidade entre a linguagem temporal Temporal Web Ontology Language (tOWL) e a Web Ontology Language 2 (OWL 2), verificando quais estruturas da tOWL são compatíveis com a OWL 2 e quais estruturas requerem modificações para manter a decidibilidade da linguagem. A linguagem tOWL foi desenvolvida para um fragmento da primeira versão da OWL. Algumas estruturas não podem ser simplesmente adicionadas à OWL 2 pois isto poderia afetar a decidibilidade.

Este trabalho também apresenta os algoritmos para raciocínio automático para lidar com as modificações feitas na linguagem tOWL. Com estes algoritmos, é possível verificar consistência de base de dados, realizar consultas semânticas e obter conhecimentos implícitos, aprendendo novos fatos à respeito da base dados.

Também é apresentado um estudo de caso utilizando uma base de dados de ocorrências aéreas. Uma ontologia temporal é construída para representar ocorrências aéreas. Devido à capacidade que a linguagem tOWL possui de lidar com aspectos temporais, podemos ligar cada ocorrência ao período em que ocorreu, podendo analisar, encontrar padrões e conectar informações com outras bases de dados.

Palavras-chave: ontologia, tOWL, temporalidade, decidibilidade, raciocínio automático

Abstract

One major obstacle to provide better support for Web users is the fact that the meaning of the majority of Web content is not accessible to machines. If we want machines to understand Web content, machines and humans need to share knowledge about the real world, in other words, it is necessary to represent the world, or parts of it within the computer.

To represent the world, it is desirable that such representation is as close to reality as possible to prevent that false assumptions are made about the world. If we want this to happen, we must be able to represent a very important aspect of the real world: time. Time is a very important aspect of human life. Many environments require a temporal awareness. One example of such an environment is the air traffic control. Each aircraft must follow a strict schedule to avoid any incident. Therefore, time should also be part of the real world representations.

We present a study of the compatibility between the Temporal Web Ontology Language (towl) and the Web Ontology Language 2 (OWL 2), checking which tOWL structures are compatible with OWL 2 and which structures require modifications to maintain the decidability of the language. The tOWL language was developed for a fragment of the first version of OWL, some structures can not simply be added to OWL 2 since this could affect the decidability.

This work also presents the algorithms for reasoning to deal with the changes made in the tOWL language. With these algorithms, we can check database consistency, perform semantic queries and get implicit knowledge, learning new facts regarding the database.

We present a case study using a database of aircraft occurrences. A temporal ontology is built to represent plane accidents, due to the ability of the tOWL language to deal with temporal aspects, we can connect each occurrence to the period in which it occurred, and we may analyze events, finding patterns and connecting information with other databases.

Keywords: ontology, tOWL, temporality, decidability, reasoning

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

$\mathcal{AL}$ Attributive Language

$\mathcal{C}$ Negação/complementação

$\mathcal{D}$ Tipos de dados

$\mathcal{H}$ Hierarquia de papéis

$\mathcal{I}$ Papéis inversos

$\mathcal{N}$ Restrições numéricas

$\mathcal{S}$ Linguagem $\mathcal{ALC}$

ABox Componente de Asserções

AFND Autômato Finito Não-Determinístico

ANAC Agência Nacional de Aviação Civil

CFP *Concrete Feature Path*

HMA Hipótese de Mundo Aberto

HMF Hipótese de Mundo Fechado

LD Lógica de Descrição

OWL *Web Ontology Language*

RCQ Restrição de Cardinalidade Qualificada

RDF *Resource Description Framework*

TBox Componente de Terminologias

W3C *World Wide Web Consortium*

1

Introdução

"Somewhere, something incredible is waiting to be known."

– Carl Sagan

Um dos maiores obstáculos para o fornecimento de um melhor suporte para os usuários da Web é o fato de que o significado do conteúdo da maior parte da mesma não ser acessível às máquinas. Existem ferramentas capazes de coletar textos, separá-los em partes, verificar a ortografia, contar palavras [19]. Entretanto, em se tratando de interpretar frases e extrair informações úteis para os usuários, as atuais ferramentas ainda são limitadas [47].

Um ser humano pode ler a seguinte frase: "*O Airbus A320-200 decolou da cidade indonésia de Surabaya com destino Cingapura e cruzou uma área com muitas nuvens antes de desaparecer dos radares. A aeronave caiu logo após no Mar de Java, e não houve sobreviventes.*" e facilmente concluir que se trata de um acidente aéreo. Uma máquina não possui esta capacidade natural, sendo somente capaz de identificar palavras-chaves e a partir das delas, tentar chegar a uma conclusão, mas não sabe o que estas palavras significam. A Web Semântica possui as ferramentas necessárias para que a máquina possa assimilar o significado de expressões, de forma a otimizar a comunicação entre humano e máquina.

A Web Semântica é propagada pelo *World Wide Web Consortium*, sendo iniciada por Tim Bernes-Lee [4], que também inventou a *WWW* no final dos anos 80. Tim Bernes espera, por esta iniciativa, a realização de sua visão original da Web, uma visão onde o significado da informação teria um papel muito mais importante do que tem na Web atual. Utilizando ferramentas da Web Semântica é possível representar aspectos do mundo real dentro do computador e podemos raciocinar com esta informação [9].

A Web Semântica pode ser aplicada em diversas áreas, sendo a área de transportes uma delas, em especial, a área de aviação. De acordo com [29], esta área gera diariamente enormes quantidades de dados. De acordo com a Agência Nacional de Aviação Civil

(ANAC) no ano de 2013 mais de 947 mil voos domésticos e 140 mil voos internacionais foram realizados [41]. Cada voo gera dados sobre número de passageiros, altura, combustível, ocorrências, entre outros. É praticamente impossível um ser humano ser capaz de processar todas estas informações e tomar decisões relevantes a partir delas sem auxílio tecnológico.

Ao utilizarmos informações semânticas juntamente com ferramentas chamadas de raciocinadores automáticos, é possível fazer consultas inteligentes e descobrir conhecimentos implícitos, sendo que esta ação seria impossível para um ser humano com uma base de dados tão extensa para análise.

Ao representar o mundo, queremos que esta representação seja a mais próxima possível da realidade, para evitar que sejam feitas falsas suposições sobre o mundo. Para isto, é necessário que também sejamos capazes de representar o tempo. Tempo é um aspecto fundamental da vida humana. Diversos ambientes requerem uma consciência temporal, um exemplo claro disso é o controle de tráfego aéreo, cada aeronave precisa seguir um cronograma restrito para evitar incidentes. Portanto, o tempo também deve ser parte das nossas representações de mundo [28].

A *Web Ontology Language* (OWL) é o padrão oficial para se presentar ontologias na Web Semântica [16], esta linguagem já se encontra em sua segunda versão. Com esta linguagem, podemos representar diversos aspectos do mundo dentro dos computadores, porém não a informação complexa sobre o tempo. A semântica desta linguagem é determinada pelo fragmento da Lógica de Descrição $\mathcal{SROIQ}$ [23]. Este fragmento é decidível, de acordo com [23], isto significa que podemos raciocinar com a representação de mundo construída com a OWL, encontrando conhecimentos implícitos e verificando sua consistência.

A *Temporal Ontology Language* (tOWL) [32] é uma linguagem temporal construída como uma extensão à primeira versão da OWL. Com esta linguagem é possível representar aspectos complexos do tempo, tais como intervalos e pontos no tempo. Esta linguagem adiciona o uso de Domínios Concretos à OWL, fazendo com que seja possível utilizar predicados e tipos de dados dos Domínios Concretos, porém a linguagem limita sua expressividade para evitar que se torne não-decidível [3]. Pelo fato da linguagem tOWL ter sido desenvolvida para a primeira versão da OWL, esta poderá se tornar incompatível com a versão atual.

Neste trabalho investigamos a compatibilidade da linguagem tOWL com a segunda versão da linguagem OWL, a OWL 2, verificando quais estruturas da tOWL são compatíveis com a OWL 2 e quais necessitam ser modificadas para que a linguagem ainda se mantenha decidível.

A primeira versão da linguagem OWL obteve muito sucesso, porém apresenta limitações, alguns elementos que modeladores de ontologias consideram importantes não foram incluídos nesta primeira versão [16]. Um destes elementos é a restrição de cardinalidade qualificada, que limita não só o número de elementos que uma propriedade pode tomar, mas também o tipo de elementos que pode tomar. Por exemplo, na OWL 1, podemos representar que uma aeronave do tipo Boeing 727 possui lugar para 192 pessoas, porém não podemos representar que 3 desses assentos são do tipo "Assento para tripulação"e 189 são do tipo "Assento para passageiro". Esta estrutura foi adicionada à linguagem OWL 2, porém, como tOWL é baseada na OWL 1, não é possível utilizar esta estrutura ao utilizar a linguagem tOWL.

Sendo assim, neste trabalho apresentamos modificações à liguagem tOWL, permitindo que esta linguagem possua a mesma expressividade da OWL 2, sem comprometer a decidibilidade da linguagem. Se ambas as linguagens possuem a mesma expressividade, é possível utilizar todas as construções da linguagem OWL 2, incluindo a restrição de cardinalidade qualificada, junto com as construções da linguagem tOWL, criando assim uma extensão temporal à linguagem OWL 2.

Para ilustrar o uso das modificações feitas na linguagem tOWL, construímos uma ontologia temporal para representar ocorrências (acidentes, incidentes, etc) aéreas. Utilizamos as estruturas temporais da linguagem tOWL para representar os intervalos em que ocorrem as fases do voo e os instantes em que aconteceram as ocorrências. Este trabalho também apresenta exemplos de como um raciocinador temporal pode ser útil para extrair conhecimentos implícitos desta ontologia.

Ontologias podem ser comparadas à Banco de dados [22]. Os axiomas da ontologia são análogos aos *schemas* de um banco de dados. Os fatos da ontologia são análogos os dados de um banco de dados. Uma das principais diferenças entre banco de dados e ontologias é que em um banco de dados nós temos uma Hipótese de Mundo Fechado (HMF) e nas ontologias temos uma Hipótese de Mundo Aberto (HMA). Na HMF, qualquer informação que falte é tratada, geralmente, como falsa, já na HMA, informações que faltam são tratadas como desconhecidas. Em uma representação de ocorrências aéreas, nem sempre temos todas as informações explícitas, porém não podemos considerar que tal informação é falsa somente pelo fato de não possuímos tal informação, por isso neste trabalho optamos por esta forma de representação [33].

Apesar da área da Web Semântica já ser relativamente antiga, não foi possível encontrar na literatura científica trabalhos relevantes que apliquem ontologias e representações temporais à área de aviação. Porém, existem trabalhos que envolvem a criação de ontologias para setores específicos da aviação.

Wang *et al.* [52] propôs uma ontologia para armazenar as causas de defeitos em aeronaves, armazenando dados de diferentes fontes em uma forma uniforme e com descrições específicas, eliminando a heterogeneidade semântica. Isto evita que dados de manutenção entrem em conflito sintático e semântico entre si. Foi concluído que este método é capaz de transformar informações redundantes e incompletas em informações consistentes e completas, melhorando a acumulação e compartilhamento de informações de defeitos em aeronaves.

Gargiulo *et al.* [12] descreve uma ferramenta de busca semântica para o aeroespaço integrada com uma ontologia para auxiliar engenheiros aeroespaciais na recuperação semântica de conhecimento. Esta ferramenta é capaz de extrair e classificar conhecimentos e informações no domínio aeroespacial. Também é proposta uma nova ontologia para o domínio aeroespacial.

1.1 Objetivos

O objetivo geral deste trabalho é criar uma nova extensão temporal para a OWL 2, baseando-se na linguagem já existente, tOWL. Esta extensão será realizada de forma a manter a decidibilidade da linguagem. As propriedades computacionais desta linguagem são discutidas e possíveis aplicações são sugeridas.

Os objetivos específicos deste trabalho são:

1. Analisar quais aspectos temporais a linguagem estendida deve suportar.
2. Analisar se algum aspecto da OWL 2 já possui suporte suficiente para representar o tempo.
3. Verificar qual fragmento da OWL 2 é mais apropriado para ser estendido.
4. Verificar se a linguagem tOWL é compatível com a linguagem OWL 2 e quais estruturas da linguagem tOWL podem ser reutilizadas na OWL 2, mantendo a decidibilidade.
5. Analisar quais as propriedades computacionais desta linguagem estendida.
6. Aplicar a linguagem tOWL em um estudo de caso para exemplificar o uso da linguagem.

Este trabalho foi dividido da seguinte forma: O Capítulo 2 apresenta a Fundamentação Teórica necessária para a realização deste trabalho. O Capítulo 3 apresenta o Estado da Arte das linguagens utilizadas na extensão temporal da linguagem OWL 2. O Capítulo 4 apresenta a comparação entre a linguagem tOWL e a linguagem OWL 2 e então

propõe novas construções para criar uma extensão temporal para a linguagem OWL 2. O Capítulo 5 apresenta um estudo de caso em que a linguagem temporal criada neste trabalho foi utilizada para representar e raciocinar em cima de uma ontologia para representar ocorrências aéreas. Por último, o Capítulo 6 apresenta as considerações finais deste trabalho.

2

Fundamentação Teórica

"The mind that opens to a new idea never returns to its original size."

– Albert Einstein

Este capítulo apresenta a revisão bibliográfica necessária para o desenvolvimento deste trabalho. A Seção 2.1 discorre sobre a Representação de Conhecimento, apresentando definições e conceitos relacionados ao conhecimento e como representá-lo. A Seção 2.2 apresenta a Lógica de Descrição, um formalismo utilizado para representar bases de conhecimento e realizar inferências a partir da mesma. Neste trabalho, a Lógica de Descrição é fundamental para a criação da ontologia proposta e para a aplicação de algoritmos de raciocínio automático. A Seção 2.3 mostra como é possível utilizar a Lógica de Descrição, juntamente com a Representação de Conhecimento em raciocinadores automáticos para obter novos fatos a partir dos que já estão contidos na base de conhecimento. A Seção 2.4 trata da Web Semântica, apresentando os conceitos de ontologia, linguagem OWL e OWL DL.

2.1 Representação do conhecimento

A definição de **conhecimento** vem sendo discutida por filósofos desde a Grécia Antiga. De acordo com Platão [8], para que uma afirmação seja considerada um conhecimento, a mesma deve seguir três condições: deve ser justificável, verdadeira e credível. Esta definição foi contestada por outros filósofos, que alegam que estas três condições não são suficientes.

O problema de Gettier [14] questiona se pode-se considerar como conhecimento algo que é verdadeiro e um indivíduo crê, por razões inválidas, que aquele fato é verdadeiro. Um exemplo desta situação é o caso de um relógio parado: Alice vê um relógio que marca duas horas, ela acredita que são duas horas, e este fato é verdadeiro. Porém, Alice não

sabe que o relógio que ela esta olhando parou de funcionar doze horas atrás. O problema de Gettier questiona se tal fato é considerado conhecimento.

Dentre as diversas definições de conhecimento existente, a definição acatada por este trabalho é dada por Fishler *et al.* [11]:

> "*Conhecimento se refere à informação armazenada ou a modelos utilizados por uma pessoa ou máquina para interpretar, prever e responder apropriadamente ao mundo exterior.*"

Da mesma forma, o conceito de **representação** [46] também possui diferentes definições. De maneira informal, representação é uma relação entre dois domínios, onde o primeiro domínio deve tomar o lugar de um segundo. Geralmente, o primeiro domínio, o representante, é mais acessível ou imediato de alguma maneira, que o segundo domínio.

2.1.1 Definição de Representação do Conhecimento

A **Representação do Conhecimento** é um campo de estudo que foca no uso de símbolos formais para representar uma coleção de proposições acreditadas por um suposto agente [45]. Porém, não é necessário que estes símbolos representem todas as proposições que este agente acredita. Somente um número finito de proposições podem ser representadas, mas o número de proposições existentes pode ser infinito [6]. O campo da Representação do Conhecimento está localizado na interseção da matemática e ciências humanas. Sendo que este, fornece os modelos computacionais e matemáticos do pensamento humano, apresentando evidências de como entender esta habilidade humana [13].

Davis *et al.* [9] define a representação do conhecimento a partir de seus cinco papéis:

1. **Substituto**:

   O Raciocínio é um processo que ocorre internamente, porém a maioria dos raciocínios ocorrem sobre objetos que existem apenas externamente. A representação funciona como um substituto dentro do raciocinador, é um dublê para os objetos que existem no mundo real. Deve existir uma correspondência entre o substituto e o objeto referenciado no mundo real, esta correspondência é a semântica da representação.

   Este substituto não é uma cópia perfeita do objeto do mundo real, pois a única representação perfeita de um objeto, é o próprio objeto e quaisquer outras representações são imprecisas. O fato de termos substitutos imperfeitos gera duas consequências: a primeira é que quando descrevemos o mundo real, temos que, inevitavelmente mentir, mesmo que por omissão. A segunda consequência é que qualquer raciocínio que utiliza substitutos imperfeitos podem gerar conclusões incorretas. Todas as

representações são imperfeitas e qualquer imperfeição pode ser uma fonte de erro. Uma boa representação deve minimizar erros gerados para uma tarefa específica.

2. **Conjunto de compromissos ontológicos**:

   Ao selecionar uma representação são definidos um conjunto de decisões sobre como e o que considerar a respeito do mundo real. Ou seja, selecionar uma representação significa realizar um conjunto de compromissos ontológicos. Ao selecionar estes compromissos, é possível focar apenas em aspectos do mundo que são considerados relevantes para um determinado domínio. A complexidade do mundo real é extraordinária, portanto, é importante que uma representação foque em certos aspectos e deixe outros de lado. Um raciocinador automático precisa de auxílio para saber que detalhes do mundo devem ser atendidos e quais devem ser ignorados.

   As ontologias podem ser escritas utilizando diversas linguagens ou notações, o essencial não é a forma da linguagem, mas o seu conteúdo, ou seja, o conjunto de conceitos que fornece para raciocinar sobre o mundo. Linguagens podem fornecer uma forma de visualizar uma representação, mas não indicam como instanciá-la.

3. **Teoria fragmentária do raciocínio inteligente**:

   O terceiro papel da representação é o de uma teoria fragmentária do raciocínio inteligente. Esta teoria é fragmentária pois a representação incorpora somente partes do conhecimento e crenças que a motivou e estes conhecimentos e crenças são apenas uma parte pequena do complexo fenômeno de raciocínio inteligente.

   Esta teoria possui três componentes: o conceito de inferência inteligente para aquela representação, o conjunto de inferências que a representação aceita e o conjunto de inferências que são recomendadas. Estes componentes podem ser vistos como respostas da representação utilizada para as seguintes questões: (1) O que significa raciocinar de forma inteligente? (2) O que podemos inferir a partir do que sabemos? (3) O que devemos inferir a partir do que sabemos?

4. **Computação eficiente**:

   Ao realizar raciocínios com o auxílio de máquinas, precisamos pensar nos aspectos práticos. Para usar uma representação, precisamos ser capaz de computá-la.

5. **Meio de expressão humana**:

   As representações de conhecimento são meios pelos quais expressamos objetos do mundo real, é uma forma de conversar com a máquina sobre o mundo.

2.2 Lógica de Descrição

A **Lógica de Descrição** (LD) [3, 10, 30] é o nome dado a uma família de formalismos da representação de conhecimento que representa conceitos do domínio e usa estes conceitos para especificar propriedades de objetos e indivíduos que ocorrem neste domínio. Um sistema de representação de conhecimento baseado em Lógica Descritiva fornece meios de definir bases de conhecimento, raciocinar sobre o seu conteúdo e manipulá-lo. A Figura 2.1 apresenta a arquitetura de tal sistema.

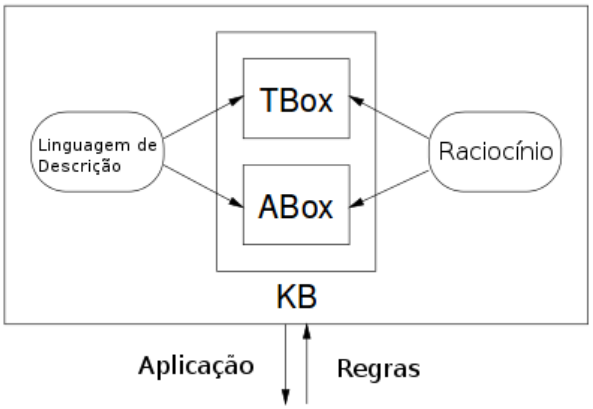

Figura 2.1: Arquitetura de um sistema de representação de conhecimento, baseada em LD [3].

De acordo com a Figura 2.1, uma base de conhecimento (KB - *Knowledge Base*) possui dois elementos, a TBox e a ABox. A TBox apresenta o vocabulário do domínio da aplicação e a ABox contém asserções sobre indivíduos nomeados de acordo com o vocabulário.

O vocabulário é formado por **conceitos** (ou classes), que denotam o conjunto de indivíduos, e os **papéis** (ou propriedades), que expressam relações binárias entre tais indivíduos. Os sistemas LD permitem que os usuários definam descrições complexas dos conceitos e dos papéis [3]. Também permitem raciocinar sobre as terminologias e as asserções.

A Lógica de descrição suporta padrões de inferências que ocorrem em diversas aplicações de sistemas inteligentes de processamento de informações [3], tais inferências também são utilizadas por humanos para estruturar e entender o mundo, classificando conceitos e indivíduos. A classificação de conceitos determina as relações de sub-conceito/super-conceito entre termos de uma dada terminologia. A classificação de indivíduos determina se um dado indivíduo é sempre instância de um certo conceito.

2.2.1 Linguagens de Descrição

Os conceitos atômicos e os papéis atômicos são descrições elementares, a partir delas podemos construir descrições mais complexas utilizando um construtor de conceitos. Nesta seção, as letras A e B são utilizadas para representar conceitos atômicos, R para papéis atômicos e as letras C e D para conceitos complexos. A Linguagem base das linguagens de descrição é a $\mathcal{AL}$ (*Attributive Language*). As definições apresentadas nesta seção se referem à linguagem $\mathcal{AL}$.

Descrição de conceitos

A descrição de conceitos na $\mathcal{AL}$ [3] é definida de acordo com a regra sintática (Equação 2.1):

$$\begin{aligned} C, D \quad \rightarrow \quad & A | (\text{conceito atômico}) \\ & \top | (\text{conceito universal}) \\ & \bot | (\text{conceito chão}) \\ & \neg A | (\text{negação atômica}) \\ & C \sqcap D | (\text{interseção}) \\ & \forall R.C | (\text{restrição de valor}) \\ & \exists R.\top (\text{quantificação existencial limitada}) \end{aligned} \tag{2.1}$$

Alguns exemplos são apresentados a seguir para ilustrar o que pode ser considerada uma expressão pela $\mathcal{AL}$. Suponha que $Pessoa$ e $Mulher$ sejam conceitos atômicos, então $Pessoa \sqcap Mulher$ e $Pessoa \sqcap \neg Mulher$ são conceitos da $\mathcal{AL}$ que descrevem pessoas que são mulheres e pessoas que não são mulheres.

Ao adicionar um papel atômico $temFilho$, pode-se formar outros conceitos, tais como, $Pessoa \sqcap \exists temFilho.\top$ e $Pessoa \sqcap \forall temFilho.Mulher$, que expressam, respectivamente, todas as pessoas que possuem um filho e todas as pessoas que possuem somente filhas. Utilizando o conceito chão, também pode-se definir as pessoas que não possuem nenhum filho: $Pessoa \sqcap \forall temFilho.\bot$.

Interpretações

Para definir uma semântica para os conceitos da $\mathcal{AL}$, são consideradas **interpretações** $\mathcal{I}$ [3], que consistem de um conjunto não vazio $\Delta^{\mathcal{I}}$, o domínio da interpretação, e uma função de interpretação, que designa cada conceito atômico A para um conjunto $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ e designa cada papel atômico R para uma relação binária $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. A função de

interpretação é estendida para a descrição de conceitos utilizando as seguintes definições indutivas (Equação 2.2):

$$\begin{aligned} \top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\ \bot^{\mathcal{I}} &= \varnothing \\ (\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} A^{\mathcal{I}} \\ (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\ (\exists R.\top)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} | \exists b.(a,b) \in R^{\mathcal{I}}\} \end{aligned} \tag{2.2}$$

Dois conceitos são considerados equivalentes, ou seja, $C \equiv D$, se $C^{\mathcal{I}} = D^{\mathcal{I}}$ para todas as interpretações $\mathcal{I}$. Por exemplo, é possível verificar que $\forall temFilho.Mulher \sqcap \forall temFilho.Estudante$ e $\forall temFilho.(Mulher \sqcap Estudante)$ são equivalentes.

Família de linguagens DL

Para obtermos uma maior expressividade da linguagem, adiciona-se novos construtores à $\mathcal{AL}$, criando extensões para a linguagem. A linguagem $\mathcal{ALC}$ [3] é obtida adicionando o operador de negação/complementação ($\neg$) à linguagem $\mathcal{AL}$.

A nomenclatura da linguagem utilizada será definida pelos elementos adicionados à linguagem, onde cada letra representa uma extensão:

- $\mathcal{S}$ para $\mathcal{ALC}$ com axiomas de transitividade;
- $\mathcal{H}$ para hierarquia de papéis (ex: $temCasa \sqsubseteq temImovel$);
- $\mathcal{R}$ para box de papéis (ex: $temPai \circ temIrmao \sqsubseteq temTio$);
- $\mathcal{O}$ para nominais (ex: {Brasil});
- $\mathcal{I}$ para papéis inversos (ex:$filhoDe \equiv temFilho^{-}$);
- $\mathcal{N}$ para restrições numéricas(ex: $\geqslant 2temFilho, \geqslant 3temFilho$);
- $\mathcal{Q}$ para restrições numéricas qualificadas (ex: $\geqslant 2temFilho.Menina$);
- $\mathcal{F}$ para restrições numéricas funcionais (ex: $\leqslant 1temMae$);

Ex: $\mathcal{SHIQ}$ = $\mathcal{S}$ + hierarquia de papéis + papéis inversos + restrições numéricas qualificadas.

2.2.2 Terminologias - TBox

Geralmente, os axiomas terminológicos possuem a seguinte forma [3]:

$$C \sqsubseteq D \, (R \sqsubseteq S) \text{ ou } C \equiv D \, (R \equiv S) \tag{2.3}$$

Onde C e D são conceitos e R e S são papéis. Os axiomas do primeiro tipo são chamados de inclusões e os do segundo tipo são chamados de igualdades.

Definições

Uma igualdade [3], onde o lado esquerdo é um conceito atômico, é uma definição. As definições são utilizadas para apresentar nomes simbólicos para descrições complexas, como, por exemplo, o seguinte axioma:

$$Mae = Mulher \sqcap \exists temFilho.Pessoa \tag{2.4}$$

Chamamos um conjunto de definições $\mathcal{T}$ de terminologia ou TBox se nenhum nome simbólico é definido mais de uma vez. Os conceitos que ocorrem em $\mathcal{T}$ são divididos em símbolos de nome $\mathcal{N}_\mathcal{T}$, que ocorrem no lado esquerdo do axioma, e em símbolos base $\mathcal{B}_\mathcal{T}$ que ocorrem apenas no lado direito dos axiomas. Os símbolos de nome são geralmente chamados de conceitos definidos e os símbolos base são chamados de conceitos primitivos.

Interpretações

Uma interpretação base para $\mathcal{T}$ é uma interpretação que interpreta apenas símbolos base [3]. Seja $\mathcal{J}$ a interpretação base, uma interpretação $\mathcal{I}$, que interpreta também símbolos de nome, é uma extensão de $\mathcal{J}$ e possui o mesmo domínio de $\mathcal{J}$. Dizemos que $\mathcal{T}$ é definitória se cada interpretação base possui exatamente uma extensão que é um modelo de $\mathcal{T}$.

A questão de se uma terminologia é definitória ou não, é relacionada com a questão de definições serem cíclicas ou não. Uma terminologia que possui o seguinte axioma possui um ciclo:

$$Humano' = Animal \sqcap \forall temProgenitor.Humano' \tag{2.5}$$

Sejam A, B conceitos atômicos que ocorrem em $\mathcal{T}$, dizemos que A utiliza B diretamente em $\mathcal{T}$ se B aparece no lado direito da definição de A. Então $\mathcal{T}$ possui um ciclo, sse, existe um conceito atômico em $\mathcal{T}$ que utiliza a ele mesmo. Caso contrário, $\mathcal{T}$ é acíclico.

Se uma terminologia $\mathcal{T}$ é acíclica, então ela é definitória. Isto se deve ao fato que podemos expandir iterativamente as definições em $\mathcal{T}$, substituindo-as pelos conceitos que

são representados. Já que não existem ciclos no conjunto de definições, o processo eventualmente para com uma terminologia $\mathcal{T}'$ consistindo somente por definições da forma $A \equiv C'$, onde C' contém apenas símbolos base. Chamamos $\mathcal{T}'$ de expansão de $\mathcal{T}$.

2.2.3 Descrição do Mundo - ABox

O segundo componente de uma base de conhecimento, além do TBox, é a descrição do mundo ou ABox [3]. Na ABox descrevemos detalhes do domínio de aplicação em termos de conceitos e papéis. Alguns dos conceitos e dos papéis na ABox podem ser nomes definidos na TBox. Utilizando a ABox, podemos dar nomes e propriedades para os indivíduos. Os nomes dos indivíduos serão dados aqui por a, b e c. Utilizando conceitos C e papéis R, podemos fazer asserções de duas maneiras:

$$C(a),\ R(b,c) \tag{2.6}$$

O primeiro tipo de asserção é chamado de asserção de conceito, que afirma que a pertence à interpretação de C. O segundo tipo é chamado asserção de papel, que afirma que c preenche o papel R em relação à b. Por exemplo, se PEDRO, PAULO e MARIA são indivíduos, então Pai(PEDRO) significa que Pedro é um pai e temFilho(MARIA, PAULO) significa que Paulo é filho de Maria. Uma ABox, denotada por $\mathcal{A}$, é um conjunto finito de tais asserções.

De uma forma simplificada, podemos ver uma ABox como uma instância de um banco de dados relacional com relações unárias e binárias. Mas ao contrário da semântica de mundo fechado utilizada pelos banco de dados clássicos, a semântica da ABox é uma semântica de mundo aberto, já que, normalmente os sistemas de representação de conhecimento são aplicados em situações onde não podemos assumir que o conhecimento da base de conhecimento é completo.

A semântica das ABoxes é dada estendendo-se interpretações para nomes de indivíduos. A interpretação $\mathcal{I}$ satisfaz a asserção de conceito $C(a)$ se $a^{\mathcal{I}} \in C^{\mathcal{I}}$, e satisfaz a asserção de papel $R(a,b)$ se $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. Uma interpretação satisfaz a ABox $\mathcal{A}$ se ela satisfaz cada asserção de $\mathcal{A}$. Neste caso, podemos dizer que $\mathcal{I}$ é um modelo da ABox. Finalmente, $\mathcal{I}$ satisfaz um ABox $\mathcal{A}$ com respeito à uma TBox $\mathcal{T}$, se, além de ser um modelo de $\mathcal{A}$, também é um modelo de $\mathcal{T}$.

2.2.4 Inferências

Um sistema de representação de conhecimento baseado em LDs consegue realizar tipos específicos de raciocínio. Uma base de conhecimento, que consiste de uma TBox e uma ABox, possui uma semântica que a torna equivalente a um conjunto de axiomas da lógica

de primeira ordem. Portanto, assim como qualquer conjunto de axiomas, contém conhecimentos implícitos que podem ser feitos explícitos através de inferências. Os diferentes tipos de raciocínio realizados por um sistema LD são definidos como inferências lógicas.

Checar a satisfabilidade de conceitos é uma inferência chave [3]. Para checarmos se um modelo de um domínio está correto, ou para otimizar consultas que são formuladas como conceitos, é necessário saber se um conceito é mais geral que o outro, este é o problema da subsunção [31]. Um conceito C é subsumido por um conceito D se em cada modelo de $\mathcal{T}$ o conjunto formado por C é um subconjunto do conjunto formado por D. Estas propriedades são definidas formalmente a seguir:

Seja $\mathcal{T}$ uma TBox [30]:

1. **Satisfabilidade:** Um conceito C é satisfatível com respeito a $\mathcal{T}$ se existe um modelo $\mathcal{I}$ de $\mathcal{T}$ tal que $C^{\mathcal{I}}$ não é vazio. Neste caso, dizemos que $\mathcal{I}$ é modelo de C.

2. **Subsunção:** Um conceito C é subsumido por um conceito D com respeito a $\mathcal{T}$ se $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ para todo modelo $\mathcal{I}$ de $\mathcal{T}$. Neste caso, escrevemos $C \sqsubseteq_{\mathcal{T}} D$ ou $\mathcal{T} \models C \sqsubseteq D$.

3. **Equivalência:** Dois conceitos C e D são equivalentes com respeito a $\mathcal{T}$ se $C^{\mathcal{I}} = D^{\mathcal{I}}$ para cada modelo $\mathcal{I}$ de $\mathcal{T}$. Neste caso, escrevemos $C \equiv_{\mathcal{T}} D$ ou $\mathcal{T} \models C \equiv D$

4. **Disjunção:** Dois conceitos C e D são disjuntos com respeito a $\mathcal{T}$ se $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \varnothing$ para cada modelo $\mathcal{I}$ de $\mathcal{T}$.

Considerando a seguinte TBox 2.7:

$$\begin{aligned}
Mulher &\equiv Pessoa \sqcap Fêmea \\
Homem &\equiv Pessoa \sqcap \neg Mulher \\
Mãe &\equiv Mulher \sqcap \exists temFilho.Pessoa \\
Pai &\equiv Homem \sqcap \exists temFilho.Pessoa \\
Progenitor &\equiv Pai \sqcup Mãe \\
Avó &\equiv Mãe \sqcap \exists temFilho.Progenitor \\
MãeComVriosFilhos &\equiv Mãe \sqcap \geqslant 3\, temFilho \\
MãeSemFilha &\equiv Mãe \sqcap \forall temFilho.\neg Mulher
\end{aligned} \tag{2.7}$$

Temos que Pessoa subsume Mulher, ambos Mulher e Progenitor subsume Mãe, e Mãe subsume Avó. Além disso, Mulher e Homem, Pai e Mãe são disjuntos. O fato de Homem ser disjunto de Mulher é devido ao fato de que Homem é subsumido pela negação de Mulher.

O mecanismo básico de raciocínio fornecido pelos sistemas LD checa a subsunção de conceitos. Isto é suficiente para implementar outras inferências, através das reduções a seguir.

Redução por Subsunção

Para conceitos C e D temos:

(i) C é insatisfatível $\Leftrightarrow$ C é subsumido por $\bot$;

(ii) C e D são equivalentes $\Leftrightarrow$ C é subsumido por D e D é subsumido por C;

(iii) C e D são disjuntos $\Leftrightarrow$ $C \sqcap D$ é subsumido por $\bot$

Se o sistema permitir o uso da negação de uma descrição, podemos também reduzir subsunção, equivalência e disjunção de conceitos ao problema de satisfabilidade:

Redução por Insatisfabilidade

(i) C é subsumido por $D \Leftrightarrow C \sqcap \neg D$ é insatisfatível;

(ii) C e D são equivalentes $\Leftrightarrow$ ambos $(C \sqcap \neg D)$ e $(\neg C \sqcap D)$ são insatisfatíveis;

(iii) C e D são disjuntos $\Leftrightarrow$ $C \sqcap D$ é insatisfatível.

2.2.5 Algoritmo *Tableau* para $\mathcal{ALC}$

Dada uma base de conhecimento $(\mathcal{T}, \mathcal{A})$, podemos assumir, sem perda de generalidade, que todos os conceitos em $\mathcal{T}$ e $\mathcal{A}$ estão na forma negativa normal (NNF - *negative normal form*), i.e., a negação é aplicada somente à nomes de conceitos. Um conceito arbitrário em $\mathcal{ALC}$ pode ser transformado a um conceito equivalente na forma NNF empurrando as negações para dentro da expressão utilizando uma combinação das leis de Morgan e a dualidade existente entre restrições existenciais e universais ($\neg\exists r.C \equiv \forall r.\neg C$ e $\neg\forall.C \equiv \exists r.\neg C$). Por exemplo, o conceito $\neg(\exists r.A \sqcap \forall s.B)$, onde A e B são nomes de conceitos, pode ser transformado para o seguinte conceito equivalente na forma NNF: $(\forall r.\neg A) \sqcup (\exists s.\neg B)$ [51].

A ideia deste algoritmo é provar a consistência de uma base de conhecimento $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ construindo um modelo para $\mathcal{K}$. Ele realiza esta tarefa iniciando-se a partir da situação concreta descrita em $\mathcal{A}$ e desenvolvendo restrições adicionais no modelo, que estão implícitas nos conceitos $\mathcal{A}$ e nos axiomas em $\mathcal{T}$. A linguagem $\mathcal{ALC}$ possui uma propriedade chamada de "modelo de floresta", isto significa que podemos assumir que este modelo possui a forma de um conjunto de árvores potencialmente infinitas em que os nós raiz podem

ser arbitrariamente interconectados. Se quisermos obter um procedimento de decisão, podemos construir apenas árvores finitas representando as árvores potencialmente infinitas, isto pode ser feito de tal forma que a representação finita pode ser resolvida como um modelo de floresta infinita $\mathcal{I}$ de $(\mathcal{T}, \mathcal{A})$.

Para construímos tal representação finita, o algoritmo utiliza uma estrutura de dados chamada *completion forest*. Ela consiste em um grafo direcionado com *labels*, em que cada nó é a raiz de uma *completion tree*. Cada nó x em uma *completion forest*, que pode ser um nó raiz ou um nó dentro de uma *completion tree*, é rotulado com um conjunto de conceitos $\mathcal{L}(x)$ e cada aresta $\langle x, y\rangle$, que pode estar entre nós raízes ou nós dentro de uma *completion tree*, é rotulada por um conjunto de nomes de papéis $\mathcal{L}(\langle x, y\rangle)$. Se $\langle x, y\rangle$ é uma aresta em uma *completion tree*, dizemos que x é um antecessor de y. No caso em que $\langle x, y\rangle$ é rotulado com um conjunto contendo o papel de nome r, então dizemos que y é um r-sucessor de x.

Ao iniciar o algoritmo com uma base de conhecimento $(\mathcal{T}, \mathcal{A})$, a *completion forest* $\mathcal{F}_{\mathcal{A}}$ é inicializada de tal forma que contenha um nó raiz x_a, com $\mathcal{L}(x_a) = \{C | a : C \in \mathcal{A}\}$, para cada nome individual a ocorrendo em $\mathcal{A}$, e uma aresta $\langle x_a, X_b\rangle$, com $\mathcal{L}(\langle x_a, X_b\rangle) = \{r|(a, b) : r \in \mathcal{A}\}$, para cada par (a, b) de nomes individuais para quais o conjunto $\{r|(a, b) : r \in \mathcal{A}\}$ é não vazio.

Após a inicialização, o algoritmo aplica as regras de expansão (Tabela 2.1), que sintaticamente decompõe os conceitos em etiquetas para os nós, inferindo novas restrições para um dado nó ou estendendo a árvore de acordo com estas restrições. O bloqueio de nós é utilizado para evitar que regras de aplicação sejam aplicadas quando se tornam repetitivas, ou seja, quando uma sub-árvore com raiz em um nó x será similar a uma sub-árvore que possui raiz em um antecessor de x. Mais precisamente, um nó y é antepassado de x se eles pertencem à mesma *completion tree* e se y é antecessor de x, ou existe um antecessor z de x, tal que y é antepassado de z. Um nó x é bloqueado se existe um antepassado y de x, tal que, $\mathcal{L}(x) \subseteq \mathcal{L}(y)$ (neste caso dizemos que y bloqueia x), ou se existe um antepassado z de x, tal que, z foi bloqueado. Se um nó x é bloqueado e nenhum dos seus antepassados foram bloqueados, dizemos que x foi bloqueado diretamente.

O algoritmo para sua execução ao encontrar um *clash*, ele ocorre em uma *completion forest* em que $\{A, \neg A\} \subseteq \mathcal{L}(x)$ para um nó x e um nome de conceito A. Neste caso a floresta contém uma inconsistência, e então, não representa um modelo. Se o algoritmo parar sem encontrar um *clash*, então a floresta produz uma representação finita de um modelo de floresta e o algoritmo pode responder "$(\mathcal{T}, \mathcal{A})$ é consistente". Se nenhuma das possíveis escolhas não determinísticas da regra-$\sqcup$ leva a tal representação de um modelo de floresta, ou seja, todas elas levam a um *clash*, então o algoritmo responde "$\mathcal{T}, \mathcal{A}$ é inconsistente". Quando um algoritmo para com uma floresta sem *clash*, um galho que

Tabela 2.1: Regras de expansão do tableau para $\mathcal{ALC}$ [51].

$\sqcap$-rule:	se 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x não está bloqueado, e
	2. $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$
	então defina $\mathcal{L}(x) = \mathcal{L}(x) \cup C_1, C_2$
$\sqcup$-rule:	se 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x não está bloqueado, e
	2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \varnothing$
	então defina $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ para algum $C \in \{C_1, C_2\}$
$\exists$-rule:	se 1. $\exists r.C \in \mathcal{L}(x)$, x não está bloqueado, e
	2. x não possui r-sucessor y com $C \in \mathcal{L}(x)$
	então crie um novo nó y com $\mathcal{L}(\langle x_a, X_b \rangle) = \{r\}$ e $\mathcal{L}(y) = \{C\}$
$\forall$-rule:	se 1. $\forall r.C \in \mathcal{L}(x)$, x não está bloqueado, e
	2. existe um r-sucessor y de x com $C \notin \mathcal{L}(y)$
	então defina $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\sqsubseteq$-rule:	se 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x não está bloqueado, e
	2. $C_2 \sqcup \neg \notin \mathcal{L}(x)$
	então defina $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \neg C_1\}$

contém um nó x bloqueado diretamente representa um galho infinito no modelo e que possui uma estrutura regular que corresponde à uma repetição infinita entre a seção do grafo entre x e o nó que o bloqueia.

2.3 Raciocinador Automático (*Reasoner*)

Um dos principais usos da representação de conhecimento é combiná-la com um Raciocinador Automático [7], ou *Reasoner* do inglês, para realizar inferências e obter novos fatos. Um **Raciocinador Automático** é um software capaz de inferir consequências lógicas a partir de um conjunto de fatos declarados ou axiomas. A noção de raciocinador automático generaliza a noção de motor de inferências, fornecendo um conjunto mais rico de mecanismos a serem trabalhados. As regras de inferência são geralmente definidas através de uma linguagem de ontologia [3].

Os principais usos de um Raciocinador Automático é para classificação e dar respostas à questões. Dado que foi selecionada uma ontologia O para ser raciocinada, a partir da mesma, o raciocinador funciona com as seguintes etapas, de maneira simplificada [42]:

1. O racionador primeiro verifica se existe um modelo para O, ou seja, se existe uma estrutura que satisfaz todos os axiomas em O. Dada a seguinte estrutura:

   Class: Humano

```
Class: Cachorro
Individual: Kyle types: Humano
Individual: Kyle types: Cachorro
DisjointClasses: Humano, Cachorro
```

esta apresentaria um erro de inconsistência, pois Humano e Cachorro são classes disjuntas, portanto, o indivíduo Kyle não pode pertencer à classe Cachorro e Humano simultaneamente.

2. Em seguida, para cada classe A que pertence à O, o raciocinador verifica se existe um modelo para O no qual é possível encontrar uma instância x de A, ou seja, existe uma estrutura que satisfaz todos os axiomas em O e, nessa estrutura, A possui uma instância x. A seguinte ontologia seria considerada consistente, mas não satisfatível para A:

   ```
   Class: Humano
   Class: Cachorro
   DisjointClasses: Humano, Cachorro
   Class: A SubClassOf: (gosta de (Humano and Cachorro))
   ```

   Não podemos ter instâncias de A para nenhum modelo desta ontologia, pois a interseção de Humano e Cachorro é o conjunto vazio. Este exemplo mostra que existem ontologias que são consistentes, mas possuem classes insatisfatíveis ao mesmo tempo.

3. Para quaisquer duas classes, denotadas por A e B, que ocorrem em O, o racionador testa se A é subsumida por B, isto é, se em cada modelo de O, cada instância de A também é uma instância de B. No exemplo a seguir, é verificado se em cada estrutura que satisfaz todos os axiomas em O, cada instância de SBL é também uma instância de SL:

   ```
   Class: Animal
   Class: Humano SubClassOf: Animal
   Class: Cachorro SubClassOf: Animal
   Class: SBL EquivalentTo: (gosta de (Humano and Cachorro))
   Class: AL EquivalentTo: (gosta de Animal)
   Class: DL SubClassOf: (gosta de Humano) and (gosta somente de Cachorro)
   ```

   Neste exemplo, DL é subsumido por SBL, que é subsumido por AL.

É fundamental que um raciocinador seja correto, completo e terminante. Existem duas abordagem principais para a construção de um racionador: *consequence-driven* e *tableau-based*.

Consequence-Driven

A abordagem *consequence-driven* [3] é bastante eficiente para os fragmentos Horn. Um fragmento Horn é aquele que exclui qualquer forma de disjunção, ou seja, não possui "ou" em expressões de classe ou (*not* (*A and B*)) ou outras construções que exigem alguma forma de raciocínio por caso. Dada uma ontologia O, o raciocinador aplica regras de dedução para inferir consequências lógicas de O. Por exemplo, uma destas regras de dedução pode inferir:

```
M SubClassOf ( p algum B)
```

a partir de:

```
 A SubClassOf: B
M: SubClassOf ( p algum A)
```

Tableau-Based

Como foi apresentado anteriormente, as duas primeiras etapas do raciocinador, consistência e satisfabilidade, exigem um modelo para a ontologia. O raciocinador *Tableau-Based* [3] tenta construir tal modelo utilizando regras de completude que funcionam neste modelo, entendendo-as de forma a satisfazer todos os axiomas. Em uma ontologia inconsistente, o algoritmo terminaria com um *clash*. Na Seção 2.2.5 é descrito um algoritmo de *Tableau* com mais detalhes.

2.4 Web Semântica

A Web Semântica é uma Web de dados [4]. Ela fornece um *framework* comum, permitindo que dados sejam compartilhados e reutilizados através de aplicações, empresas e comunidades. É um esforço colaborativo conduzido pela W3C com participação de diversos pesquisadores e parceiros industriais [4]. Os dados contidos na Web Semântica devem ser disponibilizados de forma que possam ser processados automaticamente por ferramentas e nos permitam encontrar novas relações entre diferentes dados.

As tecnologias da Web Semântica [26] podem ser utilizadas em diversas áreas, por exemplo, na integração de dados, descoberta de recursos e classificação, ferramentas de busca para domínios específicos, catalogação para descrever conteúdos e relação entre conteúdos de um Web site, facilitação de troca e compartilhamento de conhecimento por softwares de agentes inteligentes, classificação de conteúdo, descrição de uma coleção de páginas, descrição de direitos de propriedade intelectual de Web sites, entre outros.

O maior desafio para se implantar a Web Semântica não é científico, e sim, uma questão de adoção de tecnologia.

A Web Semântica é implementada numa abordagem de camadas, conforme é apresentado na Figura 2.2. As camadas são [2]:

1. XML: linguagem que permite escrever documentos estruturados para a WWW com termos definidos pelo usuário.

2. RDF: Modelo básico de dados para escrever declarações simples sobre objetos da Web.

3. RDF *Schema*: Fornece primitivas para organizar objetos da Web em hierarquias. Pode ser visto como uma linguagem primitiva para se escrever ontologias.

4. *Ontology vocabulary*: Formada pelas linguagens que permitem escrever ontologias, por exemplo, a OWL (*Web Ontology Language*).

5. *Logic*: Camada que melhora a linguagem de ontologia e permite escrever conhecimentos para aplicações específicas.

6. *Proof*: A camada de prova envolve o processo dedutivo, prova das representações da Web e validação de provas.

7. *Trust*: A camada de confiança surge através do uso de assinaturas digitais e outros tipos de conhecimentos, baseados em recomendação por agentes confiavéis.

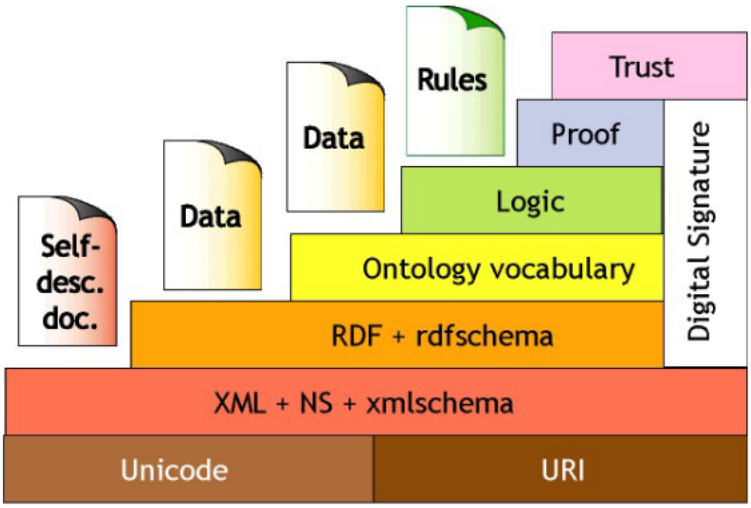

Figura 2.2: As camadas da Web Semântica, também chamadas de *Layer Cake* [2].

2.4.1 Ontologias

Uma ontologia é uma especificação formal e explícita de uma conceitualização, esta descreve formalmente o domínio do discurso [2]. A ontologia deve ser desenvolvida como um conjunto de declarações precisas sobre uma parte do mundo, o domínio do discurso. De acordo com [2], o uso de declarações precisas previne os desentendimentos que existem na comunicação humana e certifica que um programa que utiliza a ontologia se comporte de maneira uniforme e previsível e que também funcione bem com outros softwares.

Tipicamente, uma ontologia consiste de uma lista finita de termos e de relações entre estes termos. Os termos denotam conceitos importantes do domínio, por exemplo, ao imaginar o domínio de um restaurante, temos diferentes conceitos: funcionários, garçons, cozinheiros, clientes, mesas, comidas, etc. As relações são representadas por hierarquias de classes. Uma hierarquia especifica uma classe C como uma subclasse de cada classe C' se cada objeto em C também está em C'. Considerando o exemplo anterior, os garçons e cozinheiros são todos funcionários, ou seja, ambos são subclasses da classe funcionários. Além das relações de subclasse, as ontologias podem incluir informações como:

- Propriedades (Ex: X cozinha Y);
- Restrições de valores (Ex: Um prato Y só é feito por um cozinheiro X);
- Afirmações de disjunção (Ex: Os clientes e os funcionários são grupos disjuntos);
- Especificação de relações lógicas entre objetos (Ex: Cada garçom deve ser responsável por um número n de mesas).

As ontologias são úteis para a organização e navegação de Web sites. Muitos Web sites já utilizam as ontologias para criar menus, onde as páginas são organizadas por uma hierarquia de termos [4]. Além disso, as ontologias são úteis para melhorar a precisão de buscas na Web. As ferramentas de buscam podem pesquisar nas páginas por conceitos precisos de uma ontologia, ao invés de coletar todas as páginas que contém certas palavras-chave. As ferramentas de busca também podem fazer a generalização dos conceitos, caso alguma busca não retorne nenhum resultado, a busca pode ser modificada para retornar resultados mais gerais, baseando-se na hierarquia da ontologia.

De acordo com [34], as linguagens mais utilizadas para a Web Semântica são as seguintes:

- XML fornece uma sintaxe para documentos estruturados, mas não impõe restrições semânticas no significado destes documentos.
- XML Schema é uma linguagem para restringir a estrutura dos documentos XML.

- RDF é um modelo de dados para objetos (recursos) e relações entre eles. Ele fornece uma semântica simples para e pode ser representado com a sintaxe XML.
- RDF Schema é o vocabulário da linguagem de descrição que descreve propriedades e classes dos recursos RDF, com semântica para generalização de hierarquias e tais propriedades e classes.
- OWL é uma linguagem de descrição com um vocabulário mais rico para descrever propriedades e classes, pode descrever relações entre classes, cardinalidade, igualdade, mais opções de tipo para as classes, classes enumeradas, características de propriedades. Esta linguagem será descrita com mais detalhes no próximo capítulo.

3

Estado da Arte

"Sem a alquimia do conhecimento o tempo passa a esmo."

– Elizabeth Mendes

Este capítulo apresenta o estado da arte atual das linguagens de representação de conhecimento para a Web Semântica. A Seção 3.1 apresenta a *Web Ontology Language*, OWL, utilizada para definir e instanciar ontologias na Web. A Seção 3.2 descreve detalhes fundamentais sobre a *Temporal Web Ontology Language*, tOWL, linguagem que adiciona a capacidade de representar intervalos e pontos no tempo à linguagem OWL.

3.1 OWL 2 - *Web Ontology Language*

A *Web Ontology Language*, OWL [34], é uma linguagem da Web Semântica desenvolvida para representar conhecimento a respeito de coisas, grupos de coisas e relações entre coisas, é utilizada para representar ontologias, a linguagem está atualmente em sua segunda versão, OWL 2.0 [16]. OWL é uma linguagem baseada em lógica, isto implica que o conhecimento expressado na OWL pode ser raciocinado automaticamente por programas para verificar a consistência da base de conhecimento ou para encontrar novos conhecimentos que estão implícitos [20].

A OWL 2 não é uma linguagem de programação, e sim declarativa, ou seja, descreve um conjunto de asserções de forma lógica [2]. Ferramentas apropriadas, como os raciocinadores automáticos, podem ser utilizados para inferir mais informações a respeito das asserções feitas.

De acordo com [20], os conceitos básicos da OWL são:

- **Axiomas:** As declarações básicas que uma ontologia OWL expressa, podem ser verdadeiras ou falsas. Podemos ter declarações que dependem de outras declarações para serem verdadeiras ou falsas, ou seja, um conjunto de declarações A acarreta

uma declaração a se quando todos as declarações em A forem verdadeiras, a também é verdadeira. Um conjunto de declarações pode ser consistente, quando todas as declarações podem ser consideradas verdadeiras em uma mesma situação, ou inconsistente, quando não é possível encontrar tal situação. Existem ferramentas da OWL, os raciocinadores automáticos, que podem automaticamente computar se uma declaração é uma consequência de outras declarações.

- **Entidades:** elementos utilizados para se referir a objetos do mundo real. São os componentes atômicos das declarações, por exemplo, objetos, categorias, relações, etc. Na OWL, os objetos são considerados indivíduos, as categorias são consideradas classes e as relações são consideradas propriedades. As propriedades podem ser divididas em propriedades de objetos, propriedades de tipo de dados e propriedades de anotação.

- **Expressões:** combinações de entidades para formar descrições complexas a partir de elementos mais básicos. Podemos combinar nomes de entidades em expressões utilizando construtores. Por exemplo, as classes atômicas "animal" e "mamífero" podem ser combinadas para descrever a classe de animais que são mamíferos. Esta nova classe é representada pela OWL por uma expressão de classe, que pode ser utilizada em declarações ou em outras expressões.

3.1.1 Modelagem de Classes, Propriedades e Indivíduos

Seguindo descrito em [20], a linguagem OWL possui as seguintes construções básicas, utilizando a notação funcional:

1. **Classes e instâncias**

   Podemos instanciar um indivíduo da seguinte forma:

   ```
   ClassAssertion( :Cachorro :Rex )
   ```

   Temos um indivíduo Rex que pertence à classe Cachorro. Um indivíduo pode pertencer à diferentes classes ao mesmo tempo.

2. **Hierarquia de Classes**

   As subclasses são definidas na forma a seguir:

   ```
   SubClassOf( :Cachorro :Mamifero )
   SubClassOf( :Mamifero :Animal )
   ```

   A classe Cachorro é uma subclasse de Mamífero e a classe Mamífero é subclasse de Animal. A relação de subclasse é transitiva, portanto, podemos inferir que Cachorro

é uma subclasse de Animal. Além disso, a relação também é reflexiva, ou seja, toda classe é sua própria subclasse, por exemplo, todo Animal é um Animal.

Para representar classes que possuem exatamente os mesmos elementos, ou seja, são semanticamente equivalentes, utilizamos a seguinte estrutura:

```
EquivalentClasses( :Animal :Bicho )
```

Onde, a classe Animal e Bicho possuem os mesmos indivíduos.

3. **Disjunção de Classes**

   Em alguns casos, pertencer à uma classe, exclui a possibilidade de pertencer a outra classe, por exemplo, um animal não pode ser gato e cachorro ao mesmo tempo. Isto é representado da seguinte maneira:

   ```
   DisjointClasses( :Cachorro :Gato )
   ```

   Adicionando disjunções podemos chegar a diversas consequências que podem nos trazer novas informações à respeito de uma base de dados.

4. **Propriedades de Objetos**

   Podemos relacionar objetos através de propriedades:

   ```
   ObjectPropertyAssertion( :temDono :Rex :Lucas )
   ```

   A ordem dos indivíduos é importante, nesta construção indicamos que o Rex tem como dono Lucas. Também podemos demonstrar a negação de propriedades, na seguinte construção representamos que Maria não é dona de Rex.

   ```
   NegativeObjectPropertyAssertion( :temDono :Rex :Maria )
   ```

5. **Hierarquias de Propriedades**

   Podemos representar também hierarquias de propriedades:

   ```
   SubObjectPropertyOf( :temDono :cuidadoPor )
   ```

   A propriedade de ter um dono é sub-propriedade da propriedade de ser cuidado por um objeto. Notemos que o contrário não é necessariamente verdade.

6. **Restrições de Domínio e Alcance**

   Quando dos indivíduos estão conectivos por uma certa propriedades, podemos fazer algumas inferências a mais sobre estes indivíduos:

   ```
   ObjectPropertyDomain( :temDono :Humano )
   ObjectPropertyRange( :temDono :AnimalDomestico )
   ```

Definimos que o objeto da propriedade deve ser um humano e o sujeito da propriedade deve ser um animal doméstico. Ao encontrar uma construção que diz "Tutu tem dono João", podemos imediatamente inferir que Tutu é um animal doméstico e que João é um Humano.

7. **Igualdade e Não Igualdade de Indivíduos**

   Podemos representar indivíduos diferentes:

   ```
   DifferentIndividuals( :Rex :Tutu )
   ```

   E os mesmos indivíduos:

   ```
   SameIndividual( :Bela :Belinha )
   ```

8. **Tipos de Dados**

   Em diversos casos queremos representar indivíduos também utilizando tipos de dados. Podemos utilizar as propriedades de tipos de dados para isto:

   ```
   DataPropertyAssertion( :temIdade :Rex "4"^^xsd:integer )
   ```

   Neste exemplo, representamos que Rex possui 4 anos, e 4 é do tipo inteiro, representado pelo tipo de dado `xsd:integer`. Também podemos representar o contrário, ou seja, Rex não tem 4 anos:

   ```
   NegativeDataPropertyAssertion( :temIdade :Rex "4"^^xsd:integer
   )
   ```

3.1.2 Semântica da OWL 2

Como apresentado em [2], existem duas formas para designar significado para ontologias na OWL 2: a Semântica Direta (*Direct Semantics*) e a Semântica Baseada em um *Resource Description Framework* (RDF) (*RDF-Based Semantics*). A Semântica Direta pode ser aplicada a ontologias que estão no subconjunto OWL 2 DL (ou, somente OWL DL) da OWL 2. As ontologias que não pertencem à OWL 2 DL, pertencem à OWL 2 *Full* e só podem ser interpretadas utilizando a Semântica Baseada em RDF.

A Semântica Direta fornece significado para a OWL 2 em um estilo da Lógica de Descrição. Já a Semântica Baseada em RDF é uma extensão da semântica para RDFs. A OWL 2 DL pode ser vista como uma versão sintaticamente restrita da OWL 2 *Full*, essas restrições foram feitas para facilitar a implementação de ferramentas para a linguagem. A OWL 2 *Full* não é decidível, portanto, para construirmos um raciocinador automático

que seja capaz de responder "sim" ou "não" para nossas perguntas, é necessário utilizar a OWL 2 DL.

A OWL DL inclui todas as construções da linguagem OWL com restrições, tal como separação de tipos, por exemplo, uma classe não pode ser ao mesmo tempo um indivíduo ou propriedade, uma propriedade não pode ser um indivíduo ou classe.

O presente trabalho irá focar na Semântica Direta, que já foi provada ser decidível e já possui diversos racionadores automáticos implementados [43, 44]. Esta semântica é relacionada à semântica da Lógica de Descrição e extende a semântica da LD $\mathcal{SROIQ}(D)$.

Lógica de Descrição $\mathcal{SROIQ}(D)$

Formalmente, cada ontologia DL é baseada em três conjuntos finitos de símbolos: um conjunto N_I de nomes de indivíduos, um conjunto N_C de nomes de conceitos e um conjunto N_R um conjunto de nomes de papéis, sendo N_R^- seu inverso. O conjunto de expressões de papéis R é definido pela seguinte gramática [27] (Equação 3.1):

$$R ::= U \mid N_R \mid N_R^- \tag{3.1}$$

onde U é o papel universal.

O conjunto de expressões de conceito C é definido como (Equação 3.2):

$$C ::= N_C \mid (C \sqcap C) \mid (C \sqcup C) \mid \neg C \mid \top \mid \bot \mid \exists R.C \mid \forall R.C \mid \geqslant nR.C \mid \leqslant nR.C \mid \exists R.Self \mid \{N_I\} \tag{3.2}$$

Os axiomas podem ser definidos da seguinte forma (Equações 3.3, 3.4 e 3.5):

$$ABox: \quad C(N_I) \quad R(N_I, N_I) \quad N_I \approx N_I \quad N_I \not\approx N_I \tag{3.3}$$

$$TBox: \quad C \sqsubseteq C \quad C \equiv C \tag{3.4}$$

$$RBox: \quad R \sqsubseteq R \quad R \equiv R \quad R \circ R \sqsubseteq R \quad Disjoint(R, R) \tag{3.5}$$

Uma ontologia $\mathcal{SROIQ}$ requer que os seguintes axiomas e conceitos contenham somente papéis simples:

- Axiomas restritos: $Disjoint(R, R)$
- Expressões de conceitos restritas: $\exists R.Self \quad \geqslant nR.C \quad \leqslant nR.C$

Os papéis não-simples são definidos como:

- se O contém um axioma $S \circ T \sqsubseteq R$, então R é não-simples,

- se R é não-simples, então seu inverso, R^- também é não-simples,
- se R é não-simples e O contém algum dos axiomas $R \sqsubseteq S$, $S \equiv R$ ou $R \equiv S$, então S também é não-simples.

Tabela 3.1: Tradução construções OWL para lógica de descrição [26].

	Sintaxe em estilo de funções OWL	Sintaxe LD
Axiomas	SubClassOf(C D)	$C \sqsubseteq D$
	ClassAssertion(C a)	$C(a)$
	ObjectPropertyAssertion(P a b)	$P(a, b)$
Expressões de Classe	ObjectIntersectionOf(C D)	$C \sqcap D$
	ObjectUnionOf(C D)	$C \sqcup D$
	ObjectComplementOf(C D)	$\neg C$
	owl:Thing	$\top$
	owl:Nothing	$\bot$
	ObjectSomeValuesFrom(P C)	$\exists P.C$
	ObjectAllValuesFrom(P C)	$\forall P.C$
Expressões de Propriedades	ObjectInverseOf(P)	P^-

A Tabela 3.1 apresenta como é possível traduzir algumas construções de linguagem OWL para uma linguagem de descrição. Os blocos de construção da DL são similares aos da OWL 2. Os axiomas DL são construídos a partir de elementos do vocabulários e construtores, ou seja, operadores lógicos. As ontologias na LD são geralmente chamadas de bases de conhecimentos e as classes e propriedades da LD são chamadas de conceitos e papéis [26].

Elementos de ontologias LD são definidos através de uma semântica de um modelo teórico, interpretando nomes de indivíduos como objetos, classes como conjuntos e propriedades como relações. Traduzindo OWL para LD, obtemos a Semântica Direta para OWL.

3.1.3 Perfis da OWL 2

Um perfil da OWL 2 é uma versão limitada da OWL 2 que troca poder de expressividade por eficiência da raciocínio automático. Além dos perfis discutidos anteriormente, OWL 2 DL e OWL 2 Full, a OWL 2 possui mais três outros perfis [26]:

OWL 2 EL

Este perfil possui o poder expressivo utilizado por muitas ontologias grandes e é um subconjunto da OWL 2 cujos problemas de raciocínio podem ser resolvidos em tempo polinomial em relação ao tamanho da ontologia.

É apropriada para aplicações que utilizam ontologias que definem um grande número de classes e/ou propriedades. O acrônimo EL se refere ao perfil base da família de lógica de descrição EL, esta lógica fornece apenas quantificação existencial.

OWL 2 QL

A OWL 2 QL é voltada para aplicações que utilizam um grande volume de dados instanciados, onde obter respostas às consultas é a tarefa mais importante de raciocínio automático. Neste perfil, as respostas à consultas podem ser implementadas utilizando sistemas de banco de dados convencionais. Utilizando uma técnica de raciocínio automático apropriada, as respostas à consultas podem ser feitas de forma correta e completa em LOGSPACE em relação ao tamanho dos dados (asserções).

O poder expressivo deste perfil é bastante limitado, porém inclui a maioria dos aspectos dos modelos conceituais, tais como diagrama de classes UML e diagramas ER. O acrônimo QL se refere ao fato de que as respostas às consultas neste perfil podem ser implementadas reescrevendo as consultas em uma *Query Language* padrão relacional.

OWL 2 RL

A OWL 2 RL é o foco de aplicações que necessitam de raciocínio automático escalável sem sacrificar demais do poder expressivo. Foi desenvolvido para acomodar aplicações da OWL 2 que podem trocar completa expressividade da linguagem por eficiência, assim como aplicações RDF(S) que precisam de uma expressividade adicional.

Os sistemas de raciocínio automático OWL 2 RL podem ser implementados utilizando máquinas de raciocínio baseadas em regras. A consistência das ontologias, satisfabilidade de expressões de classe, subsunção de expressões de classe e respostas à consultas podem ser resolvidas em tempo polinomial em relação ao tamanho da ontologia. O acrônimo RL se refere ao fato de que o raciocínio automático pode ser implementado utilizando Linguagem de Regras.

3.1.4 Tipos de dados

Nesta Seção apresentamos as definições formais fundamentais para o sistema de tipos de dados da OWL 2. A noção central neste sistema é o *datatype map*. Na OWL 2, *datatype*

maps suportam *facets*, essas são expressões que podem ser aplicadas à um tipo de dados para restringir sua interpretação.

Um *datatype map* é uma 4-tupla $\mathcal{D} = (N_D, N_C, N_F, \cdot^{\mathcal{D}})$, onde:

- N_D é um conjunto de tipos de dados,
- N_C é uma função que mapeia um conjunto de constantes $N_C(d)$ para cada $d \in N_D$,
- N_F é uma função que mapeia um conjunto de *facets* $N_F(d)$ para cada $d \in N_D$,
- $\cdot^{\mathcal{D}}$ é a função que mapeia uma interpretação de tipo de dado $d^{\mathcal{D}}$ para cada tipo de dado $d \in N_D$, uma interpretação *facet* $f^{\mathcal{D}} \subseteq d^{\mathcal{D}}$ para cada *facet* $f \in N_F(d)$ e um valor de dados $v^{\mathcal{D}} \in d^{\mathcal{D}}$ para cada constante $v \in N_C(d)$.

Uma expressão *facet* para um tipo de dados $d \in N_D$ é uma fórmula φ construída utilizando conectivos proposicionais sobre os elementos de $N_F(d) \cup \{\top_d, \bot_d\}$. A função $\cdot^{\mathcal{D}}$ é estendida para expressões *facets* para d por definição, para $f_{(i)} \in N_F(d)$, $\top_d^{\mathcal{D}} = d^{\mathcal{D}}$, $\bot_d^{\mathcal{D}} = \varnothing$, $(\neg f)^{\mathcal{D}} = d^{\mathcal{D}} \backslash f^{\mathcal{D}}$, $(f_1 \wedge f_2)^{\mathcal{D}} = f_1^{\mathcal{D}} \cap f_2^{\mathcal{D}}$ e $(f_1 \vee f_2)^{\mathcal{D}} = f_1^{\mathcal{D}} \cup f_2^{\mathcal{D}}$.

Por exemplo, $\mathcal{D}$ pode ser um *datatype map* com $N_D = str, real$, onde $str^{\mathcal{D}}$ e $real^{\mathcal{D}}$ são os conjuntos de todas strings e números reais, respectivamente. Os conjuntos $N_C(str)$ e $N_C(real)$ contém todas as strings constantes e todas as representações decimais dos números reais. Finalmente, o conjunto $N_F(real)$ pode conter a *facet int*, interpretada como o junto de todos os inteiros e *facets* da forma $<_w$, $>_w$, $\leqslant_w$ e $\geqslant_W$ para cada número decimal w. Com isso, a expressão *facet* $int \wedge >_{12} \wedge <_{15}$ representa os inteiros 13 e 14.

Seja $\mathcal{D} = (N_D, N_C, N_F, \cdot^{\mathcal{D}})$ um *datatype map*. O conjunto de intervalo de dados para $\mathcal{D}$ é o menor conjunto que contém $\top_{\mathcal{D}}$, d, $d[\varphi]$, $\{v_1, ..., v_n\}$, $\overline{dr}$, para $d \in N_D$, sendo φ uma expressão *facet* para d, $v_i \in N_C$ e dr um intervalo de dados.

Seja $\mathcal{DL}$ uma lógica de descrição, definida sobre um conjunto de indivíduos N_I e seja N_{DP} um conjunto de propriedades de dados disjunto de cada um dos conjuntos de símbolos usados em $\mathcal{DL}$. A lógica $\mathcal{DL} + \mathcal{D}$, obtida ao se estender $\mathcal{DL}$ com $\mathcal{D}$, é definida da seguinte forma. O conjunto de conceitos de $\mathcal{DL} + \mathcal{D}$ estende o conjunto de conceitos da $\mathcal{DL}$ com conceitos de tipos de dados da forma $\exists U.dr$, $\forall U.dr$,$\geqslant nU.dr$ e $\leqslant nU.dr$, para $u \in N_{DP}$, um inteiro não negativo n, e um intervalo de dados dr para $\mathcal{D}$.

O conjunto de axiomas $\mathcal{DL} + \mathcal{D}$ estende o conjunto de axiomas $\mathcal{DL}$ com os axiomas de disjunção de propriedades de dados Dis(U_1, U_2), axiomas de inclusão de propriedade de dados $U_1 \sqsubseteq U_2$ e asserções de propriedade de dados $U(a, v)$, onde $U_{(i)} \in N_{DP}$, $a \in N_{DP}$ e $v \in N_C$.

3.1.5 Limitações da OWL 1

Ao desenvolver a segunda versão da OWL, muitas limitações e problemas encontrados na OWL 1 foram tratados [16]. Nesta Seção apresentamos algumas das limitações da OWL 1, que motivaram a criação de uma nova versão da linguagem.

Limitações de Expressividade

A experiência de desenvolvedores com a OWL 1 demonstrou que a OWL 1 DL, a linguagem mais expressiva e decidível da família de linguagens OWL, não possui diversos construtores fundamentais para a modelagem de domínios mais complexos [16].

Restrição de cardinalidade qualificada: Um desses construtores é a restrição de cardinalidade qualificada(RCQ) , a OWL 1 DL permite que restrições existenciais sejam qualificadas com uma classe, mas não permite o mesmo para restrições de cardinalidade. Por exemplo, podemos representar a classe com as seguintes restrições em OWL 1: "Pessoas que possuem pelo menos um filho do sexo masculino"(restrição existencial) ou "pessoas que possuem pelo menos três filhos"(restrição de cardinalidade). Porém, não podemos representar a seguinte restrição de cardinalidade qualificada: "pessoas que possuem pelo menos três filhos do sexo masculino"sem utilizar recursos que tornem a linguagem incompleta ou incorreta.

Propriedades: A OWL 1 possui também diversas limitações relacionadas à criação de propriedades. Algumas aplicações precisam de propriedades que se propagam entre as classes e subclasses, ou seja, de uma propagação transitiva de papéis. Esta característica é especialmente importante para os domínio das ciências biológicas, onde é necessário descrever interações entre propriedades locais e propriedades do todo. Em OWL 1 não podemos representar de maneira direta que uma anormalidade em uma parte de uma estrutura anatômica constitui uma anormalidade na estrutura completa.

Um outro exemplo de limitação é a propriedade "parteDe", que é frequentemente especificada como transitiva (se x é parte de y e y é parte de z, então x é parte de z), reflexiva (cada objeto é parte dele mesmo) e assimétrica (nenhuma parte é parte de uma de suas partes). Em OWL 1 não podemos utilizar esse tipo de "propriedades de propriedade"sem termos problemas na prática.

Tipos de Dados: a OWL 1 fornece um poder de expressividade bastante limitado para descrever classes com instâncias relacionadas à valores concretos, por exemplo, inteiros e strings. Não é possível representar construções como:

- restrições à um subconjunto de valores de um tipo de dados (ex: valores saudável de glicose no sangue estão entre 60 e 110 mg/dL).
- relações entre valores de propriedades de dados em um objeto (ex: uma mesa quadrada possui a largura igual sua profundidade).
- relações entre valores de propriedades de dados em objetos diferentes (ex: pessoas que são mais velhas que seus chefes).
- funções de agregação (ex: a duração de um processo é a soma da duração de seus subprocessos).

Chaves: a OWL 1 DL não fornece meios para expressar chaves de valor restrito em propriedades de dados, um aspecto fundamental para as tecnologias atuais de banco de dados. Por exemplo, em OWL 1 não é possível definir que "um cidadão brasileiro é identificado unicamente pelo seu valor de CPF".

Semântica da OWL 1 DL

A OWL 1 DL foi criada como uma variação notacional da lógica de descrição $\mathcal{SHOIN}(D)$. A lógica de descrição $\mathcal{SHOIN}(D)$ tem sido extensivamente investigada na literatura e suas propriedades computacionais e de expressividade são bem conhecidas. A especificação da OWL 1 DL fornece uma definição explícita que não corresponde exatamente à lógica $\mathcal{SHOIN}(D)$. Isto causou inúmeros problemas relativos a suas habilidades de apresentação e entendimento.

Os desenvolvedores de ferramentas, como raciocinadores automáticos, que utilizam a OWL 1 DL precisaram fazer uma correspondência entre a OWL 1 DL e a $\mathcal{SHOIN}(D)$ [16]. Um problema é a habilidade de utilizar indivíduos sem nome nos fatos da OWL 1. Por exemplo, uma pessoa pode afirmar que um indivíduo João é amigo de um amigo de um indivíduo chamado Paulo, nem nomear tal indivíduo. Isto não está disponível diretamente em $\mathcal{SHOIN}(D)$, portanto, a maneira de simular esta habilidade não é óbvia.

3.2 tOWL - *Temporal Web Ontology Language*

A *Temporal Web Ontology Language*, tOWL [32], é uma extensão da OWL que permite a comunicação entre máquinas em contextos que exigem temporalidade. A linguagem tOWL permite que sejam feitas inferências de conhecimentos implícitos na existência de temporalidade e também permite que esse conhecimento seja representado quando existe uma dimensão temporal envolvida.

A tOWL foi desenvolvida como uma extensão da OWL DL, perfil da primeira versão da OWL, com a adição da unidade de tempo. O fragmento da OWL DL considerado

foi o $\mathcal{SHIN}(\mathcal{D})$, que representa a OWL DL sem o uso de nominais, as letras desta nomenclatura possuem o seguinte significado, a saber: $\mathcal{S}$ representa a linguagem $\mathcal{ALC}$, $\mathcal{H}$ representa a hierarquia de papéis, $\mathcal{I}$ para papéis inversos, $\mathcal{N}$ para restrições numéricas, $\mathcal{D}$ representa suporte para tipos de dados. Este fragmento da OWL DL baseado na $\mathcal{SHIN}(\mathcal{D})$ é denotado por OWL DL^{-}. Este fragmento possui um algoritmo terminante, correto e completo para se realizar o raciocínio automático.

O conceito de tempo é bastante discutido e possui diferentes definições. A tOWL implementa dois aspectos do tempo: infraestrutura temporal e mudança [32]. A infraestrutura temporal se refere à representação do tempo em forma de intervalos ou instantes. A linguagem permite o tempo em forma de pontos e em forma de intervalos.

O aspecto relacionado à mudança é adicionado em modificações na ABox, ou seja, a nível de indivíduo. São permitidas três tipos de mudanças em relação ao tempo:

1. Mudanças em valores de atributos concretos de um indivíduo, tal como, cor de cabelo.

2. Mudanças em relações entre entidades, por exemplo, quais professores lecionam uma disciplina.

3. Transições de estados, tal como, uma pessoa passando de um estado "solteira" para um estado "casada"

A linguagem foi desenvolvida em três camadas: (1) Camada de Domínios Concretos, (2) Camada de Referência Temporal e (3) Camada de 4D *Fluents*.

Camada de Domínios Concretos

Esta camada permite a representação de restrições utilizando predicados binários de domínios concretos. Em tOWL é possível representar *feature chains* composta com um *concrete feature* g, formando um *concrete feature path* (CFP), que é equivalente à seguinte composição:

$$f_1 \circ f_2 \circ ...f_n \circ g, \tag{3.6}$$

onde $n \in N$. Um exemplo de uso do CFP, seria a composição do *abstract feature* **time** com o *concrete feature* **start**, da seguinte forma:

$$time \circ start. \tag{3.7}$$

Esta construção denota o início de um ponto em um intervalo, aplicando primeiramente o *abstract feature time* para obter o intervalo associado com um indivíduo e então aplicar o *concrete feature start* para obter o início daquele intervalo.

A Tabela 3.2 resume a semântica introduzida por esta camada, com a sintaxe abstrata proposta para os construtores da linguagem tOWL. As definições apresentadas são:

- `ConcreteFeatureChain`: define que a interpretação de tal conceito é formada por todos os pares de indivíduos do domínio abstrato e domínio concreto, respectivamente, tal que cada um dos indivíduos abstratos está na interpretação do *abstract feature* f_1 junto com exatamente um outro indivíduo abstrato, a_2, que, por sua vez, está na interpretação de f_2, junto de exatamente um indivíduo , a_3, e assim em diante, até a_{n+1}. Por fim, a interpretação do *concrete feature* g no indivíduo a_{n+1} deverá ser definida e tomar somente um valor concreto, neste caso, nomeado b.
- `dataSomeValuesFrom`: define que a interpretação de tal conceito consiste de todos os indivíduos do domínio abstrato que, quando duas *concrete feature chain* u_1 e u_2 são interpretadas sobre estes indivíduos, o resultado consiste dos valores concretos q_1 e q_2, que estão na interpretação do predicado de domínio concreto p_d.
- `dataAllValuesFrom`: similar à construção anterior, porém, neste caso a relação p_d deve ser verdadeira para todos os valores de q_1 e q_2.

Tabela 3.2: Semântica para a camada de domínios concretos [32].

Sintaxe abstrata tOWL	**Semântica do modelo teórico**
`ConcreteFeatureChain`$(f_1 \ \ f_2 \ \ ... \ \ f_n \ \ g)$	$\{(a_1, b) \in \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}} \mid \exists! a_2 \in \Delta^{\mathcal{I}}, ..., \exists! a_{n+1} \in \Delta^{\mathcal{I}} \wedge$ $\wedge \ \exists! b \in \Delta_{\mathcal{D}} : (a_1, a_2) \in f_1^{\mathcal{I}}, ...$ $(a_n, a_{n+1}) \in f_n^{\mathcal{I}} \wedge g^{\mathcal{I}}(a_{n+1}) = b\}$.
`dataSomeValuesFrom`$(u_1 \ \ u_2 \ \ p_d)$	$\{x \in \Delta^{\mathcal{I}} \mid \exists! q_1 \in \Delta_{\mathcal{D}}, \exists! q_2 \in \Delta_{\mathcal{D}} :$ $u_1^{\mathcal{I}}(x) = q_1 \wedge u_2^{\mathcal{I}}(x) = q_2 \wedge \ (q_1, q_2) \in p_d^{\mathcal{I}}\}$.
`dataAllValuesFrom`$(u_1 \ \ u_2 \ \ p_d)$	$\{x \in \Delta^{\mathcal{I}} \mid \forall q_1 \in \Delta_{\mathcal{D}}, \forall q_2 \in \Delta_{\mathcal{D}} :$ $u_1^{\mathcal{I}}(x) = q_1 \wedge u_2^{\mathcal{I}}(x) = q_2 \wedge \ (q_1, q_2) \in p_d^{\mathcal{I}})\}$.

Camada de Referência Temporal

A Camada de Referência Temporal apresenta *timepoints*, relações entre *timepoints* e intervalos. Os intervalos são definidos utilizando o predicado de domínio concreto

$<$ e os dois *concrete features start* e *end* para definir que o início de um intervalo deve ser sempre estritamente menor que o final deste intervalo:

$$ProperInterval \equiv \exists(begin, end).\ <\tag{3.8}$$

Camada de 4D *Fluents / TimeSlices*

Esta camada apresenta uma visão perdurantista de indivíduos, permitindo representar aspectos temporais complexos, como transição de estados em processos. A Tabela 3.3 apresenta os axiomas da TBox correspondentes a *timeslaces/fluents layer*.

O conceito de `TimeSlice` é definido como todos os indivíduos para quais a propriedade `time` é definida e toma um valor do tipo `Interval`, e para quais a propriedade `timeSliceOf` é definida e toma um valor que não é um `Interval`, um `TimeSlice` ou um `Literal`. O conceito de `Interval` é definido como todos os indivíduos cujas propriedades `start` e `end` são definidas e tomam um valor do XML Schema `dateTime`, tal que o valor associado com o ponto inicial deve ser menor que o valor associado com o ponto final.

O conceito `FluentProperty` é definido como uma subclasse da classe RDF `property`, e é uma superclasse dos construtores `FluentObjectProperty` e `FluentDataProperty`. A propriedade `timeSliceOf` é definida como a propriedade que pode ser aplicada à *timeslices*, *intervas* ou *literals*. A propriedade `time` é definida como uma propriedade que somente toma valores do tipo `Interval` e pode ser aplicada à indivíduos do tipo `TimeSlice`. As propriedades `start` e `end` são definidas como as propriedades que são definidas para os intervalos e tomam valores do *XML Schema* `dateTime`.

A Tabela 3.4 fornece uma visão geral das descrições tOWL permitidas pela linguagem e a Tabela 3.5 apresenta alguns dos axiomas e fatos que são aceitos pela tOWL.

Tabela 3.3: Axiomas tOwl para a camada *4DFluents* [32].

Construtores tOWL 4dFluents	**Axiomas tOWL em OWL-DL**
Class(TimeSlice)	$\exists$`time.Interval` $\sqcap$ (= 1 `time`) $\sqcap$ $\exists$`timeSliceOf`.$\neg$(`TimeSlice` $\sqcup$ `Interval`$\sqcup$ $\sqcup$ `rdfs:Literal`) $\sqcap$ (= 1 `timeSliceOf`)
Class(Interval)	$\exists$(`start`, `end`). $\leqslant$ $\sqcap$$\exists$`start.dateTime` $\sqcap$ $\exists$`end.dateTime`$\sqcap$ $\sqcap$(= 1 `start`) $\sqcap$ (= 1 `end`)
Class(FluentProperty)	`FluentProperty` $\sqsubset$ `rdf:Property`
Class(FluentObjectProperty)	`FluentObjectProperty` $\sqsubset$ `FluentProperty`
Class(FluentDatatypeProperty)	`FluentDatatypeProperty` $\sqsubset$ `FluentProperty`
Property(timeSliceOf)	$\geqslant$ 1 `timeSliceOf` $\sqsubseteq$ `TimeSlice` $\top$ $\sqsubseteq$ $\forall$`timeSliceOf`.$\neg$(`TimeSlice` $\sqcup$ `Interval` $\sqcup$ `rdfs:Literal`)
Property(time)	$\geqslant$ 1 `time` $\sqsubseteq$ `TimeSlice` $\top$ $\sqsubseteq$ $\forall$`time.Interval`
Property(start)	$\geqslant$ 1 `start` $\sqsubseteq$ `Interval` $\top$ $\sqsubseteq$ $\forall$`start.dateTime`
Property(end)	$\geqslant$ 1 `end` $\sqsubseteq$ `Interval` $\top$ $\sqsubseteq$ $\forall$`end.dateTime`

Tabela 3.4: Construtores tOWL [32].

Sintaxe Abstrata tOWL	Sintaxe DL	Semântica
A (URI Reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`towl:Thing`	$\top$	$\Delta^{\mathcal{I}}$
`towl:Nothing`	$\bot$	$\{\}$
`intersectionOf`(C_1 C_2...)	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
`unionOf`(C_1 C_2...)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
`complementOf`(C)	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
`restriction`(R `someValuesFrom`(C))	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$
`restriction`(R `allValuesFrom`(C))	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
`restriction`(R `minCardinality`(n))	$\geqslant n\ R$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp(\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}) \geqslant n\}$
`restriction`(R `maxCardinality`(n))	$\leqslant n\ R$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp(\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}) \leqslant n\}$
`restriction`(U `someValuesFrom`(D))	$\exists U.D$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x,y\rangle \in U^{\mathcal{I}}$ and $y \in D^{\mathcal{D}}\}$
`restriction`(U `allValuesFrom`(D))	$\forall U.D$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x,y\rangle \in U^{\mathcal{I}}$ and $y \in D^{\mathcal{D}}\}$
`restriction`(U `minCardinality`(n))	$\geqslant n\ U$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp(\{y.\langle x,y\rangle \in U^{\mathcal{I}}\}) \geqslant n\}$
`restriction`(U `maxCardinality`(n))	$\leqslant n\ U$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp(\{y.\langle x,y\rangle \in U^{\mathcal{I}}\}) \leqslant n\}$
`ConcreteFeatureChain`(f_1 f_2 ... f_n g)	$f_1 \circ ... \circ f_n \circ g$	$\{(a_1,b) \in \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}} \mid \exists! a_2 \in \Delta^{\mathcal{I}}, ..., \exists! a_{n+1} \in \Delta^{\mathcal{I}} \wedge$ $\wedge\ \exists! b \in \Delta_{\mathcal{D}} : (a_1,a_2) \in f_1^{\mathcal{I}}, ...$ $(a_n,a_{n+1}) \in f_n^{\mathcal{I}} \wedge g^{\mathcal{I}}(a_{n+1}) = b\}$.
`restriction`((u_1,u_2) `someValuesFrom`(p_d))	$\exists(u_1,u_2).p_d$	$\{x \in \Delta^{\mathcal{I}} \mid \exists! q_1 \in \Delta_{\mathcal{D}}, \exists! q_2 \in \Delta_{\mathcal{D}} : u_1^{\mathcal{I}}(x) = q_1 \wedge$ $\wedge\ u_2^{\mathcal{I}}(x) = q_2 \wedge\ (q_1,q_2) \in p_d^{\mathcal{I}}\}$
`restriction`((u_1,u_2) `allValuesFrom`(p_d))	$\forall(u_1,u_2).p_d$	$\{x \in \Delta^{\mathcal{I}} \mid \forall q_1 \in \Delta_{\mathcal{D}}, \forall q_2 \in \Delta_{\mathcal{D}} : u_1^{\mathcal{I}}(x) = q_1 \wedge$ $\wedge\ u_2^{\mathcal{I}}(x) = q_2 \wedge\ (q_1,q_2) \in p_d^{\mathcal{I}})\}$

Tabela 3.5: Axiomas e Fatos tOWL [32].

Sintaxe Abstrata tOWL	Sintaxe DL	Semântica
`Class`(A `partial` $C_1...C_n$)	$A \sqsubseteq C_1 \sqcap ... \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$
`Class`(A `complete` $C_1...C_n$)	$A = C_1 \sqcap ... \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$
`SubClassOf` (C_1 C_2)	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
`EquivalentClasses` ($C_1...C_n$)	$C_1 = ... = C_n$	$C_1^{\mathcal{I}} = ... = C_n^{\mathcal{I}}$
`DisjointClasses` ($C_1...C_n$)	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \{\}, i \neq j$
`Datatype`(D)		$D^{\mathcal{I}} \subseteq \Delta_D^{\mathcal{I}}$
`DatatypeProperty`(U `super`(U_1)...`super`(U_n)	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$
`domain`(C_1)...`domain`(C_m)	$\geqslant 1U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$
`range`(D_1)...`range`(D_l)	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
`[Functional])`	$\top \sqsubseteq \leqslant 1U$	$U^{\mathcal{I}}$ is functional
`SubPropertyOf`(U_1 U_2)	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
`EquivalentProperties`($U_1...U_n$)	$U_1 = ... = U_n$	$U_1^{\mathcal{I}} = ... = U_n\mathcal{I}$
`ObjectProperty`(R `super`(R_1)...`super`(R_n)	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
`domain`(C_1)...`domain`(C_m)	$\geqslant 1R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
`range`(C_1)...`range`(C_l)	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
`[inverseOf`(R_0)`]`	$R = ({}^{-}R_0)$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^{-}$
`[Symmetric]`	$R = ({}^{-}R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{-}$
`[Functional]`	$\top \sqsubseteq \leqslant 1R$	$R^{\mathcal{I}}$ is functional
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1R^{-}$	$(R^{\mathcal{I}})^{-}$ is functional
`[Transitive])`	$R^{+} \sqsubseteq R$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
`SubPropertyOf`(R_1 R_2)	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
`EquivalentProperties`($R_1...R_n$)	$R_1 = ... = R_n$	$R_1^{\mathcal{I}} = ... = R_n^{\mathcal{I}}$
`AnnotationProperty`(S)		
`FluentDatatypeProperty`(U^{FD} `super`(U_1^{FD})...`super`(U_n^{FD})	$U^{FD} \sqsubseteq U_i^{FD}$	$(U^{FD})^{\mathcal{I}} \subseteq (U_i^{FD})^{\mathcal{I}}$
`domain`(C_1^{TS})...`domain`(C_m^{TS})	$\geqslant 1U^{FD} \sqsubseteq C_i^{TS}$	$(U^{FD})^{\mathcal{I}} \subseteq (C_i^{TS})^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$
`range`(D_1)... `range` (D_l))	$\top \sqsubseteq \forall U^{FD}.D_i$	$(U^{FD})^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
`FluentObjectProperty`(R^{FO} `super`(R_1^{FO})...`super`(R_n^{FO})	$R^{FO} \sqsubseteq R_i^{FO}$	$(R^{FO})^{\mathcal{I}} \subseteq (R_i^{FO})^{\mathcal{I}}$
`domain`(C_1^{TS})...`domain`(C_m^{TS})	$\geqslant 1R^{FO} \sqsubseteq C_i^{TS}$	$(R^{FO})^{\mathcal{I}} \subseteq (C_i^{TS})^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$
`range`(C_1^{TS})...`range`(C_l^{TS}))	$\top \sqsubseteq \forall R^{FO}.C_i^{TS}$	$(R^{FO})^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times (C_i^{TS})^{\mathcal{I}}$
`Individual`(o `type` (C_1)... `type` (C_n)	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$
`value`(R_1 o_1)... `value` (R_n o_n)	$\langle o, o_i \rangle \in R_i$	$\langle o^{\mathcal{I}}, o_i^{\mathcal{I}} \rangle \in R_i^{\mathcal{I}}$
`value`(U_1 v_1)... `value` (U_n v_n))	$\langle o, v_i \rangle \in U_i$	$\langle o^{\mathcal{I}}, v_i^{\mathcal{I}} \rangle \in U_i^{\mathcal{I}}$
`SameIndividual`($o_1...o_n$)	$o_1 = ... = o_n$	$o_1^{\mathcal{I}} = ... = o_n^{\mathcal{I}}$
`DifferentIndividuals`($o_1...o_n$)	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$
`TimeSlice`(o^{TS} `type` (C_1^{TS})... `type` (C_n^{TS})	$o^{TS} \in C_i^{TS}$	$(o^{TS})^{\mathcal{I}} \in (C^{TS})^{\mathcal{I}}$
`value`(`timeSliceOf` o)	$\langle o^{TS}, o \rangle \in$ `timeSliceOf`	$\langle (o^{TS})^{\mathcal{I}}, o^{\mathcal{I}} \rangle \in$ `timeSliceOf`$^{\mathcal{I}}$
`value`(R_1^{FO} o_1^{TS})... `value` (R_n^{FO} o_n^{TS})	$\langle o^{TS}, o_i^{TS} \rangle \in R_i^{FO}$	$\langle (o^{TS})^{\mathcal{I}}, (o_i^{TS})^{\mathcal{I}} \rangle \in (R_i^{FO})^{\mathcal{I}}$
`value`(U_1^{FD} v_1)... `value` (U_n^{FD} v_n))	$\langle o^{TS}, v_i \rangle \in U_i^{FD}$	$\langle (o^{TS})^{\mathcal{I}}, v_i^{\mathcal{I}} \rangle \in (U_i^{FD})^{\mathcal{I}}$

4

Estendendo a linguagem tOWL

"Rien n'est plus puissant qu'une idée dont l'heure est venue."

– Victor Hugo

Neste capítulo é apresentado um estudo sobre a compatibilidade de extensão da linguagem OWL e a linguagem tOWL. Será apresentada uma extensão à tOWL com novidades adicionadas na expressividade de linguagem, enquanto a primeira versão é baseada no fragmento $\mathcal{SHIN}(D)$ (OWL-DL$^-$) da lógica de descrição, esta extensão se baseia no fragmento $\mathcal{SROIQ}(D)$ (OWL 2 DL). O fragmento $\mathcal{SROIQ}(D)$ foi provado ser decidível [23, 34] e já existem *reasoners* implementados para o mesmo [16], portanto, podemos raciocinar utilizando este fragmento. A Seção 4.1 apresenta os conceitos de tempo representados na linguagem e a Seção 4.2 apresenta como as camadas foram modificadas para se adequar ao novo padrão da OWL. A Seção 4.3 apresenta como podemos raciocinar com as estruturas existentes na OWL 2 e com as estruturas adicionadas à tOWL.

4.1 Aspectos temporais

Um dos aspectos mais importantes ao se definir uma linguagem para representar o tempo é a definição dos aspectos do tempo a linguagem será capaz de representar.

Ao se representar o tempo através de uma representação abstrata, é necessário considerar dois aspectos do tempo: o tempo pode ser representado como um sistema de referência, ordenando eventos. Neste caso, o tempo pode ser representado por instantes ou por intervalos, sendo os instantes pontos básicos no tempo, sem duração, e intervalos sendo pares de instantes distintos, representando um período de tempo.

Outro aspecto do tempo a ser considerado ao representar o tempo, são as relações efêmeras, ou seja de curta duração, entre indivíduos. A linguagem deve ser capaz de

representar mudanças. Essas mudanças são, por exemplo em [32], as descrições de indivíduos que possuem valores diferentes para uma propriedade em diferentes momentos.

A forma como um indivíduo persiste no tempo também é importante para a definição da linguagem. As duas teorias mais populares à respeito da persistência de indivíduos no tempo são o endurantismo (três dimensões) e o perdurantismo (quatro dimensões), descritas em [18].

O endurantismo trata os objetos como elementos que persistem em três dimensões e se movem através do tempo. Persistir em três dimensões significa que um objeto está presente em um tempo, e então está em um próximo tempo e depois em um próximo tempo de novo, e assim em diante. As partes do objeto estão completamente presentes em cada tempo em que estes existem. Um objeto que está presente aqui neste momento está inteiramente aqui neste momento e somente aqui neste momento.

O perdurantismo é a negação do endurantismo: os objetos não estão completamente presentes em cada momento no tempo em que eles existem. Os objetos são vistos como partes temporais. Ao se observar um objeto aqui e agora, estamos vendo partes do objeto que estão aqui agora, mas também existem partes do objeto que existem em outros momentos que podemos ainda encontrar ou que já encontramos. A famosa citação de Heráclito se relaciona ao perdurantismo: "Ninguém se banha duas vezes na àgua do mesmo rio" [48].

Assim como a primeira versão da linguagem, a extensão proposta utiliza uma abordagem perdurantista, ou 4D, para representar a persistência de objetos através do tempo. A abordagem utilizada na tOWL é a de ontologia de *4D fluents* [53]. Um *fluent* é uma função que toma objetos e situações e os mapeia para um valor de verdade, são relações cujos valores variam de um intervalo para o outro.

A sintaxe abstrata da OWL para *4D fluents* é a seguinte:

```
Ontology(4dfluents
  Class(Timeinterval partial)
  Class(TemporalPart partial)
  DisjointClasses(TimeInterval TemporalPart)
  ObjectProperty(fluentProperty
domain(TemporalPart)
    range(TemporalPart))
  ObjectProperty(temporalExtent Functional
   domain(TemporalPart)
    range(TimeInterval))
  ObjectProperty(temporalPartOf Functional
   inverseOf(hasTemporalPart)
```

```
    domain(TemporalPart)
    range(complementOf(TimeInterval)))
)
```

A classe *4dFluents:TimeInterval* deve ser substituída pela classe de tempo associada com a ontologia construída. Essa construção apresenta algumas restrições: (i) as partes temporais devem ser máximas com respeito aos intervalos de todos os *fluents* dos quais participam. Ou seja, o intervalo de tempo de uma parte temporal deve ser definido com a mesma duração de um *fluent* com o qual se relaciona; (ii) duas partes temporais da mesma entidade em um mesmo intervalo devem ser iguais; (iii) as partes temporais que participam de um mesmo *fluent* devem ter a mesma duração.

A Figura 4.1 apresenta um exemplo de uso de *fluents*, em que esta ontologia mostra como podemos representar partes temporais de objetos. Neste exemplo tem-se os objetos IBM e SAM, que são instâncias de *Company* e *Person*, respectivamente. Os objetos *IBM@t1* e *Sam@t1* são partes temporais dos objetos já mencionados e se conectam entre si pelo relacionamento *ceoOf*. Estas partes temporais existem em um intervalo t_1, este intervalo é uma instância da classe *Interval*, que é definida como o intervalo de tempo que ocorre entre dois pontos no tempo, onde um ponto deve ser estritamente menor que o outro.

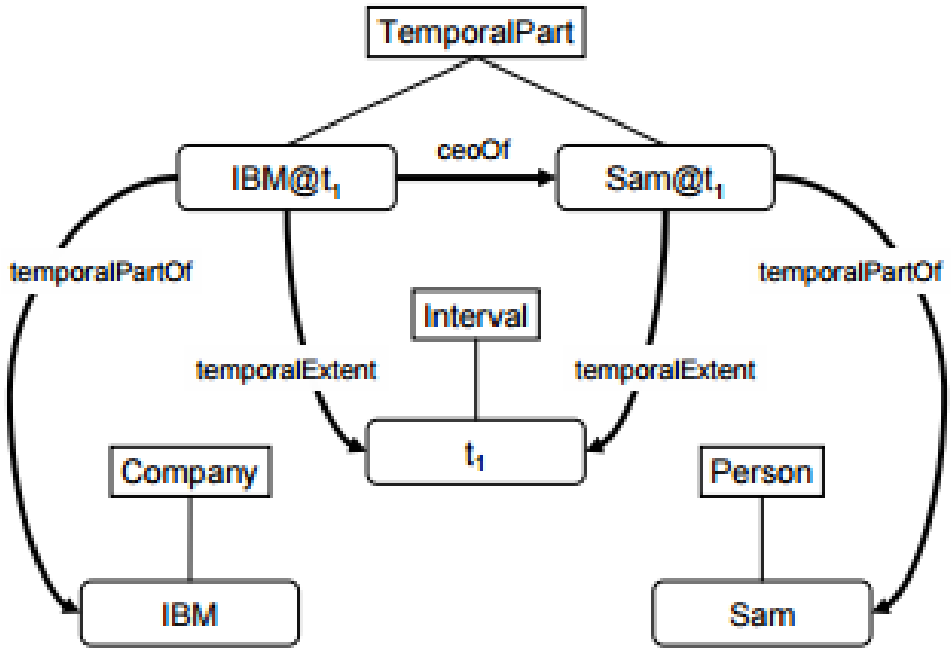

Figura 4.1: Exemplo de uso de *fluents* [53].

Utilizando a ontologia definida anteriormente para *4D fluents*, a ontologia da Figura 4.1 seria representada da seguinte forma:

```
Individual(IBM type(Company))
Individual(SamPalmisano type(Person))
Individual(t1 type(time:Interval))
Individual(SamPalmisano@t1 type(4dFluents:TemporalPart)
 value(4dFluents:temporalPartOf SamPalmisano)
 value(4dFluents:temporalExtent t1)
 value(ceoOf IBM@t1))
Individual(IBM@t1 type(4dFluents:TemporalPart)
 value(4dFluents:temporalPartOf IBM)
 value(4dFluents:temporalExtent t1))
```

Uma questão que surge ao se utilizar objetos que persistem no tempo de forma perdurante é o problema de identidade [37]. A questão: as partes temporais de um mesmo objeto podem ser consideradas equivalentes? Por exemplo, um homem com 50 anos de idade e com cabelos grisalhos deve ser considerado equivalente ao homem de 30 anos atrás, com 20 anos de idade e cabelos ainda com cor? Esta é a questão de como a identidade varia com o tempo.

Existem dois tipos de identidade, segundo [37], a saber: sincrônica ou diacrônica. A identidade sincrônica é aquela identidade que é verdadeira em um momento no tempo, ou seja, o que torna x realmente x em um tempo t. Um exemplo é o fato das partes do corpo de um indivíduo serem o indivíduo em si.

A identidade diacrônica se refere à como a identidade se mantém ao longo do tempo, ou seja, o que torna um objeto x em um momento t_1 igual a esse x em um momento t_2. Por exemplo, que fatores tornam um navio de ontem o mesmo navio de hoje?

Esta nova extensão da linguagem tOWL precisa ser capaz de lidar com o problema da identidade diacrônica. Esta questão é tratada pelo Princípio da Identidade dos Indiscerníveis e o Princípio da Indiscernibilidade de Idênticos propostos por Leibniz [5]. O Princípio da Identidade dos Indiscerníveis é formulado da seguinte forma: se, para cada propriedade F, um objeto x possui F se e somente se um objeto y possui F, então x é idêntico à y. Em notação de lógica simbólica:

$$\forall F(Fx \leftrightarrow Fy) \rightarrow x = y \tag{4.1}$$

O Princípio da Indiscernibilidade de Idênticos é formulado de forma inversa ao princípio anterior:

$$x = y \rightarrow \forall F(Fx \leftrightarrow Fy) \tag{4.2}$$

A Lei de Leibniz é formada pela junção destes dois princípios, esta lei afirma que dois elementos X e Y são idênticos se, e somente se, eles compartilham todas e somente as mesmas propriedades. Respeitando esta lei é possível determinar se dois indivíduos são equivalentes em tempos diferentes.

4.2 Desenvolvimento da Linguagem

A primeira versão da linguagem tOWL [32] foi construída em três camadas no topo da OWL-DL^{-}: camada de domínios concretos, camada de referência temporal e camada de *4D fluents*.

A atual versão da OWL utiliza *Datatype Maps* para adicionar papéis concretos e tipos de dados à linguagem, ao contrário da primeira versão da OWL, que utilizava uma versão simplificada de domínios concretos para representar tipos de dados [34].

Considerando isto, a tOWL permanece com as camadas: **Camada de Referência Temporal** e **Camada de *4D fluents*** e adicionamos a **Camada de *Concrete Path***

4.2.1 Camada de Concrete Path

Na primeira versão da tOWL a primeira camada é a de domínios concretos, que adicionava à linguagem o conjunto $\mathbb{Q}$ de números racionais e o conjunto de predicados $\{<, \leqslant, =, \neq, \geqslant, >\}$. Na linguagem OWL 2, o conjunto de números racionais já é nativo à linguagem, através do *Datatype Map*, sendo nomeado `owl:rational`.

O tipo de dados `owl:rational` acompanha os *facets* apresentados na Tabela 4.1, onde v pertence ao *value space* de `owl:real` e DT é do tipo `owl:rational`. A primeira coluna da Tabela 4.1 mostra a sintaxe utilizada pelos *facets* e a segunda coluna mostra para onde cada par representado na primeira coluna é mapeado. A partir destes *facets*, é possível obter todos os predicados utilizados na primeira versão da tOWL.

A primeira versão da tOWL continha os seguintes construtores na primeira camada: `ConcreteFeatureChain`, `dataSomeValuesFrom` e `dataAllValuesFrom`. Para a criação da extensão da tOWL, foi necessário traduzir cada uma destas construções para a OWL 2.

Tabela 4.1: *Facets* do tipo de dados `owl:rational` [35].

Sintaxe	**Mapeamento**
`( xsd:minInclusive , v )`	Todo x no *value space* de DT, tal que $x = v$ ou $x > v$
`( xsd:maxInclusive , v )`	Todo x no *value space* de DT, tal que $x = v$ ou $x < v$
`( xsd:minExclusive , v )`	Todo x no *value space* de DT, tal que $x > v$
`( xsd:maxExclusive , v )`	Todo x no *value space* de DT, tal que $x < v$

ConcreteFeatureChain

Este construtor foi criado na primeira versão da tOWL com a função de permitir o uso do *Concrete Feature Path* (CFP). A tOWL utiliza *features*, estes são papéis que podem tomar valores do domínio da interpretação, ou seja, valores abstratos e levarem para o mesmo domínio (*abstract features*), ou leva para o domínio concreto (*concrete features*). Por exemplo, o *concrete feature* `idade` leva os indivíduos do domínio da interpretação aos valores de idade contidos no domínio concreto de números inteiros. Já o *abstract feature* `progenitorDe`, toma valores dos indivíduos do domínio de interpretação, os pais, e leva até o mesmo domínio para indicar quais indivíduos são os filhos.

O CFP é uma construção que permite realizar a composição destas *features* da seguinte forma:

$$f_1 \circ f_2 \circ ... \circ f_n \circ g, \tag{4.3}$$

onde f_i, ... , f_n são *abstract features*, g é um *concrete feature* e $n \in \mathbb{N}$. Sendo que para $n = 0$, o conjunto de *abstract features* é vazio.

Este construtor é fundamental para a linguagem tOWL, já que precisamos desta composição de *features* para representar construções relacionadas ao tempo, tais como:

$$time \circ start. \tag{4.4}$$

A construção na Equação 4.4 denota o ponto de início de um intervalo, primeiro aplicando o *abstract feature* `time`, para obter o intervalo associado com um indivíduo e então aplicando o *concrete feature* `start` para obter o ponto de início daquele intervalo.

Como a OWL 2 não utiliza domínios concretos, o conceito de *feature* não pôde ser utilizado para o desenvolvimento da extensão da tOWL. Porém, em OWL 2, temos definições parecidas às dos *features*: papéis concretos e papéis abstratos [21]. Os papéis abstratos (ou *object property*) conectam indivíduos à indivíduos, os papéis concretos (ou *datatype*

property) conectam indivíduos à valores de dados, ou seja, elementos que possuem tipos de dados.

Além disso, na OWL 2 pode se realizar a composição de papéis abstratos, conforme apresentado na Tabela 4.2, onde R_1 ... R_n são papéis concretos, com $n \in \mathbb{N}$.

Tabela 4.2: Composição de papéis concretos em OWL 2.

Construtor OWL 2	**Sintaxe LD**	**Semântica LD**
`ObjectPropertyChain(`R_1 ... R_n`)`	$R_1 \circ R_2 \circ ... \circ R_n$	$\{(a,b) \in \Delta^I \times \Delta^I \mid$ $\exists c \in \Delta^I.(a,x_1) \in R_1^{\mathcal{I}} \wedge (x_1,x_2) \in$ $R_2^{\mathcal{I}} \wedge$ ... $\wedge (x_{(n-1)},b) \in R_n^{\mathcal{I}}\}$

Baseando-se nos construtores já existem na OWL 2 e na primeira versão da tOWL, esta extensão apresenta um novo construtor com função equivalente ao `ConcreteFeatureChain`: a `DatatypePropertyChain`. Com esta função podemos criar algo equivalente à CFP, uma composição de n papéis abstratos e um papel concreto.

Tabela 4.3: Semântica e Sintaxe da `DatatypePropertyChain`.

Construtor OWL 2	**Sintaxe LD**	**Semântica LD**
`DataPropertyChain(`R_1 ... R_n S`)`	$R_1 \circ R_2 \circ ... \circ R_n \circ S$	$\{(a,b) \in \Delta^I \times \Delta^D \mid$ $\exists c \in \Delta^I.(a,x_1) \in R_1^{\mathcal{I}} \wedge (x_1,x_2) \in$ $R_2^{\mathcal{I}} \wedge$ ... $\wedge (x_{n-1},b) \in S^{\mathcal{I}}\}$

Com a adição da construção mostrada na Tabela 4.3 podemos também adicionar uma outra construção interessante para aumentar a expressividade da linguagem. Com a expressividade da $\mathcal{SROIQ}(D)$ podemos ter axiomas de papéis da seguinte forma:

$$R_1 \circ R_2 \sqsubseteq R_3, \tag{4.5}$$

onde R_1, R_2 e R_3 são papéis abstratos.

Utilizando o construtor `DataPropertyChain`, podemos estender a construção mostrada na Equação 4.5, chamada de inclusão de papéis complexos, para também utilizar papéis concretos. A Tabela 4.4 mostra este novo construtor, chamado `SubDatatypePropertyOf`, onde R_1 ... R_n são papéis abstratos e S_1 e S_2 são papéis concretos.

Tabela 4.4: Semântica e Sintaxe da `SubDatatypePropertyOf`.

Construtor OWL 2	**Sintaxe LD**	**Semântica LD**
`SubDatatypePropertyOf(` `DataPropertyChain(`R_1 ... R_n S_1) S_2)	$R_1 \circ R_2 \circ ...$ $\circ R_n \circ S_1 \sqsubseteq S_2$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} \circ ...$ $\circ R_n^{\mathcal{I}} \circ S_1^{\mathcal{I}} \subseteq S_2^{\mathcal{I}}$

Com o construtor `SubDataPropertyOf` é possível criar novos papéis concretos, realizando a composição de outros papéis. Podemos criar um novo papel para denotar o início de um intervalo, utilizando a composição da Equação 4.4 da seguinte forma:ç

$$time \circ start \sqsubseteq startOfInterval \tag{4.6}$$

É necessário tomar cuidado ao elaborar a ontologia utilizando tais construções, pois as propriedades de transitividade, simetria e inversão não podem ser utilizadas para papéis concretos, assim como as propriedades de assimetria, reflexividade e irreflexividade. O mesmo algoritmo criado para avaliar a composição de papéis abstratos pode ser adaptado para avaliar a *concrete feature chain* [23].

Os construtores `dataSomeValuesFrom` e `dataAllValuesFrom`

Alguns dos construtores da primeira versão da tOWL nesta camada já foram adicionados à OWL 2, desta forma evitamos a introdução de indecidibilidade à linguagem. A Tabela 4.5 apresenta estes construtores e suas respectivas versões na OWL 2, onde u_1 e u_2 são *feature chains*, p_d é um predicado do domínio concreto, S_1 ... S_2 são papéis concretos e dr é uma *data range*.

Tabela 4.5: Construtores tOWL vs Construtores OWL 2 [32].

Construtor tOWL	**Construtor OWL 2**
`dataSomeValuesFrom`(u_1 u_2 p_d)	`DataSomeValuesFrom`(S_1 ... S_n dr)
`dataAllValuesFrom`(u_1 u_2 p_d)	`DataAllValuesFrom`(S_1 ... S_n dr)

Podemos criar papéis concretos através de *feature chains* utilizando a construção de papéis concretos complexos, conforme apresentado na Tabela 4.4.

A *data range* é uma construção que surgiu na OWL 2, os *data ranges* mais básicos são os tipos de dados. A construção de *data ranges* admite apenas o uso de *data ranges* unários, ou seja, não poderíamos comparar duas propriedades para gerar uma *data range*. Porém, esse problema pode ser resolvido baseando-se em domínios concretos, po-

demos utilizá-la para representar intervalos de dados, por exemplo. Podemos representar o fato de o início do intervalo ser menor que o final de um intervalo da seguinte forma: `DataAllValuesFrom(startOfInterval endOfInterval DataComparison(Arguments(x y) le( x y ))))`, onde `DataComparison` é uma extensão à OWL 2 utilizada para comparar dois atributos de dados [38].

4.2.2 Camada de referência temporal

A camada de referência temporal utiliza as 13 relações de intervalos de Allen [1] para estabelecer relações entre intervalos, como sugerido no trabalho original à respeito da linguagem tOWL [32]. Um intervalo i é considerado um par de números reais ordenados, (x_1, x_2), com $x_1 \leqslant x_2$. Desta forma, um intervalo pode ser definido da seguinte forma:

$$Interval = (start \leqslant end) \tag{4.7}$$

Um par de intervalos agrupa dois intervalos em *first* e *second*:

$$Pair = \exists first.Interval \sqcap \exists second.Interval \tag{4.8}$$

A Tabela 4.6 apresenta os construtores e a sintaxe da lógica de descrição correspondente.

4.2.3 Camada de *4D fluents*

A camada de *4D fluents* permaneceu a mesma que na primeira versão da linguagem. Conforme mostra a Tabela 3.3

4.3 Decidibilidade da tOWL extendida

Esta seção apresenta a análise sobre a decidibilidade da linguagem tOWL estendida neste trabalho. A Seção 4.3.1 apresenta o algoritmo para raciocinar com o fragmento $\mathcal{SROIQ}$ da Lógica de Descrição. A Seção 4.3.2 apresenta como é possível tornar as construções adicionadas à tOWL compatíveis com este algoritmo de raciocínio.

4.3.1 Raciocinando com $\mathcal{SROIQ}$

O fragmento $\mathcal{SROIQ}$ da Lógica de Descrição já foi provado ser decidível em [23]. Nesta Seção apresentamos o algoritmo utilizado para raciocinar com $\mathcal{SROIQ}$ e, consequentemente, com a OWL 2. No algoritmo a seguir [23], C e D representam conceitos,

Tabela 4.6: Construtores da camada de referência temporal

Construtor tOWL	**Sintaxe LD**
`equalsInterval`$(pair)$	$Pair \sqcap (first \circ left \ = \ second \circ left) \sqcap$ $(first \circ right \ = \ second \circ right)$
`beforeInterval`$(pair)$	$Pair \sqcap (first \circ right \ < \ second \circ left)$
`afterInterval`$(pair)$	$Pair \sqcap (first \circ left \ > \ second \circ right)$
`meetsInterval`$(pair)$	$Pair \sqcap (first \circ right \ = \ second \circ left)$
`metByInterval`$(pair)$	$Pair \sqcap (first \circ left \ = \ second \circ right)$
`overlapsInterval`$(pair)$	$Pair \sqcap (first \circ left \ < \ second \circ left) \sqcap$ $(first \circ right \ < \ second \circ right) \sqcap$ $(first \circ right \ > \ second \circ left)$
`overlappedbyInterval`$(pair)$	$Pair \sqcap (first \circ right \ > \ second \circ right) \sqcap$ $(first \circ left \ > \ second \circ left) \sqcap$ $(first \circ left \ < \ second \circ right)$
`duringInterval`$(pair)$	$Pair \sqcap (first \circ left \ > \ second \circ left) \sqcap$ $(first \circ right \ < \ second \circ right)$
`containsInterval`$(pair)$	$Pair \sqcap (first \circ right \ > \ second \circ right) \sqcap$ $(first \circ left \ < \ second \circ left$
`startsInterval`$(pair)$	$Pair \sqcap (first \circ left \ = \ second \circ left) \sqcap$ $(first \circ right \ < \ second \circ right)$
`startedByInterval`$(pair)$	$Pair \sqcap (first \circ left \ = \ second \circ left) \sqcap$ $(first \circ right \ > \ second \circ right)$
`finishesInterval`$(pair)$	$Pair \sqcap (first \circ left \ > \ second \circ left) \sqcap$ $(first \circ right \ = \ second \circ right)$
`finishedByInterval`$(pair)$	$Pair \sqcap (first \circ right \ = \ second \circ right) \sqcap$ $(first \circ left \ < \ second \circ left)$

R e S representam papéis, $\mathcal{R}$ juntos com seus inversos. $\mathcal{R}_h \cup \mathcal{R}_a$ é a Rbox e $\mathbf{R}_C$ é o conjunto de papéis ocorrendo em C e $\mathcal{R}$ juntos com seus inversos. Primeiramente definimos $\texttt{fclos}(C_0, \mathcal{R})$, o fecho de um conceito C_0 em relação à uma hierarquia de papéis $\mathcal{R}$. Um autômato finito não-determinístico [24] (AFND) é utilizado em restrições universais de valor para memorizar o caminho entre um objeto que deve satisfazer uma restrição de valor e outros objetos. Desta forma, o algoritmo irá "empilhar"a restrição de valor junto com todos os caminhos, enquanto o AFND é atualizado com o caminho tomado até um certo ponto.

O algoritmo irá gerar um grafo de conclusão, esta é uma estrutura que se for completa e livre de *clashs*, pode ser traduzida como um *tableau* para os conceitos e Rbox de entrada.

Definição 1. ***Grafo de Conclusão -*** *Seja $\mathcal{R}$ uma RBox, C_0 um conceito na forma normal da negação e seja N um conjunto de nominais. Um grafo de conclusão para C_0 com respeito à $\mathcal{R}$ é um grafo direcionado $G = (V, E, \mathcal{L}, \neq)$ onde cada nó $x \in V$ é rotulado com um conjunto*

$$\mathcal{L}(x) \subseteq \texttt{fclos}(C_0, \mathcal{R}) \cup N \cup \{(\leqslant mR.C) | (\leqslant nR.C) \in \texttt{fclos}(C_0, \mathcal{R}) \, e \, m \leqslant n\} \tag{4.9}$$

e cada vértice $< x, y > \in E$ é rotulado com um conjunto de papéis de nome $\mathcal{L}(< x, y >)$ contendo papéis ocorrendo em C_0 ou $\mathcal{R}$. Neste contexto, $R \in \mathcal{L}(< x, y >)$ é usada como abreviação de $< x, y > \in E$ e $R \in \mathcal{L}(< x, y >)$.

Se $< x, y > \in E$, então y é chamado de sucessor de x e x é chamado de antecessor de y. Ancestral é o fechamento transitivo de antecessor e descendente é o fechamento transitivo de sucessor. Um nó y é chamado de R-sucessor de um nó x se para algum R' com $R' \sqsubseteq R$, $R' \in \mathcal{L}(< x, y >)$. Um nó y é chamado de vizinho (R-vizinho) de um nó x se y é um sucessor de x ou se x é um sucessor de y

Para um papel S e um nó x em $\boldsymbol{G}$, definimos um conjnto de x's S-vizinhos com C em seu rótulo, $S^{\boldsymbol{G}}(x, C)$, descrito na Equação 4.10:

$$S^{\boldsymbol{G}} := \{y \mid y \text{ é um } S\text{-vizinho de } x \text{ e } C \in \mathcal{L}(y)\} \tag{4.10}$$

*Dizemos que $\boldsymbol{G}$ contém um **clash** se existem nós x e y tais que:*

1. $\bot \in \mathcal{L}(x)$, *ou*
2. *para um nome de conceito A, $\{A, \neg A\} \subseteq \mathcal{L}(x)$, ou*
3. *x é um S-vizinho de x e $\neg \exists S.Self \in \mathcal{L}(\S)$, ou*
4. *Existe algum $\texttt{Dis}(R, S) \in R_a$ e y é um R-vizinho e $S-$vizinho de x, ou*

5. *Existe algum* $\texttt{Asy}(R) \in R_a$ *e* y *é um* R*-vizinho de* x *e* x *é um* R*-vizinho de* y*, ou*

6. *Existe algum conceito* $(\leqslant nS.C) \in \mathcal{L}(x)$ *e* $\{y_0, ..., y_n\} \subseteq S^{\mathbf{G}}(x, C)$ *com* $y_i \neq y_j$ *para todo* $0 \leqslant i < j \leqslant n$*, ou*

7. *Para algum* $o \in N$*,* $x \neq y$ *e* $o \in \mathcal{L}(\S) \cap \mathcal{L}(\dagger)$

Se $o_1, ..., o_l$ são todos os nominais ocorrendo em C_0 então o algoritmo *tableau* é inicializado com o grafo de conclusão $\mathbf{G} = (R_0, r_1, ..., r_l, \varnothing, \mathcal{L}, \varnothing)$ com $\mathcal{L}(r_0) = C_0$ e $\mathcal{L}(r_i) = \{o_i\}$ para $1 \leqslant i \leqslant l$. $\mathbf{G}$ é então expandido repetidamente aplicando as regras de expansão mostradas na Tabela 4.7, parando a expansão caso ocorra um *clash*. É possível observar a similaridade desta tabela com a Tabela 2.1, o algoritmo é similar, porém com adição de novas regras. A Tabela 4.7 apresenta apenas as principais regras de expansão, o restante das regras podem ser encontradas em [23]. A complexidade deste algoritmo é de N2ExpTime-complete ($O(2^{2^{p(n)}})$, onde $p(n)$ é um polinômio) [25] .

4.3.2 Raciocinando com os construtores adicionados à tOWL

Nesta Seção apresentamos como é realizado o raciocínio automático para os construtores adicionados à tOWL neste trabalho: `DatatypePropertyChain` e `SubDatatypePropertyOf`. Para raciocinar utilizando estas construções é necessário adicionar a capacidade de lidar com axiomas de inclusão de papéis concretos. O método que permite isso é a transformação de uma hierarquia de papéis (tal como $time \circ start \sqsubseteq startOfInterval$) em um autômato finito não-determinístico. O método apresentado aqui se baseia no algoritmo utilizado para avaliar a expressão `ObjectPropertyChain`, tal método já possui prova de sua decidibilidade, apresentada em [23]. Após a transformação da hierarquia de papéis em um autômato, este pode ser avaliado por uma das regras do algoritmo apresentado na Tabela 4.7.

Considere a seguinte expressão:

$$R_1 \circ ... \circ R_n \circ ..S_1 \sqsubseteq S_2, \tag{4.11}$$

onde $R_1..R_n$ são papéis abstratos e S_1, S_2 são papéis concretos.

Define-se para uma hierarquia de papéis $\mathcal{R}_h$ um AFND $\mathcal{B}_S$ que interpreta todas as implicações entre os papéis $R_1...R_n$ S_1 e S_2 que são consequências de $\mathcal{R}_h$.

O autômato $\mathcal{B}_S$ é definido como:

Definição 2. *Seja* C *um conceito e* $\mathcal{R}$ *uma Rbox. Para cada nome de papel* R *ocorrendo em* $\mathcal{R}$ *ou* C*, definimos o AFND* $\mathcal{A}_S$ *da seguinte forma:* $\mathcal{A}_S$ *contém um estado* i_S *e um estado* f_S *com a transição* $i_S \xrightarrow{R} f_S$*. O estado* i_S *é o único estado inicial e* f_S *é o único*

Tabela 4.7: Regras de expansão do *tableau* para $\mathcal{SROIQ}$ [23].

$\sqcap$-rule:	se $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x não está bloqueado diretamente, e $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$ então $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	se $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x não está indiretamente bloqueado, e $\{C_1, C_2\} \cap \mathcal{L}(x) = \varnothing$ então $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{E\}$ para algum $E \in \{C_1, C_2\}$
$\exists$-rule:	se $\exists S.C \in \mathcal{L}(x)$, x não está bloqueado, e x não possui S-vizinho y com $C \in \mathcal{L}(y)$ então crie um novo nó y com $\mathcal{L}(\langle x, y\rangle) := \{S\}$ e $\mathcal{L}(y) := \{C\}$
$\mathtt{Self} - \mathtt{Ref}$-rule:	se $\exists S.\mathtt{Self} \in \mathcal{L}(x)$ ou $\mathtt{Ref}(S) \in R_a$, x não está bloqueado e $S \notin \mathcal{L}(\langle x, y\rangle)$ existe adicione um vértice $\langle x, x\rangle$ se o mesmo ainda não existir, e defina $\mathcal{L}(\langle x, x\rangle) \longrightarrow \mathcal{L}(\langle x, x\rangle) \cup \{S\}$
$\forall_1$-rule:	se $\forall S.C \in \mathcal{L}(x)$, x não está bloqueado, e $\forall \mathcal{B}_S.C \notin \mathcal{L}(x)$ então defina $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\forall \mathcal{B}_S.C\}$
$\forall_2$-rule:	se $\forall \mathcal{B}.C \in \mathcal{L}(x)$, x não está indiretamente bloqueado p $\underrightarrow{s}$ q em $\mathcal{B}(p)$, e existe um S-vizinho y de x com $\forall \mathcal{B}(q).C \notin \mathcal{L}(y)$ então $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall \mathcal{B}(q).C\}$
$\forall_3$-rule:	se $\forall \mathcal{B}.C \in \mathcal{L}(x)$, x não está indiretamente bloqueado, $\epsilon \in \mathcal{L}(\mathcal{B})$ e $C \notin \mathcal{L}(x)$ então $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$
choose-rule:	se $(\geqslant nS.C) \in \mathcal{L}(x)$, x não está indiretamente bloqueado e existe um S-vizinho y de x com $\{C, \neg C\} \cap \mathcal{L}(y) = \varnothing$ então $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{E\}$ para algum $E \in \{C, \neg C\}$
$\geqslant$-rule:	se 1. $(\geqslant nS.C) \in \mathcal{L}(x)$, x não está bloqueado 2. não existem n S-vizinhos seguros $y_1, ..., y_n$ de x com $C \in \mathcal{L}(y_i)$ e $y_i \neq y_j$ para $1 \leqslant i < j \leqslant n$ então criar n novos nos $y_1, ..., y_n$ com $\mathcal{L}(\langle x_i, y_i\rangle) = \{S\}, \mathcal{L}(y_i) = C$ e $y_i \neq y_j$ para $1 \leqslant i < j \leqslant n$ então defina $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \neg C_1\}$
$\leqslant$-rule:	se 1. $(\leqslant nS.C) \in \mathcal{L}(z)$, z não está indiretamente bloqueado 2. $\#S^{\mathbf{G}}(z, C) > n$ e existem dois S-vizinhos x, y, de z com $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$, e não $x \neq y$ então 1. se y é um nó nominal então $\mathtt{Merge}(y, x)$ 2. senão, se y é um nó nominal ou um ancestral de x, então $\mathtt{Merge}(x, y)$ 3. senão $\mathtt{Merge}(y, x)$

estado final. Além disso, para cada $w \sqsubseteq S \in \mathcal{R}$, $\mathcal{A}_S$ *contém os seguintes estados e transições:*

- *se* $w = R_1 S_1$, *então* $\mathcal{A}_R$ *contém* $f_S \xrightarrow{\epsilon} i_S$, *e*
- *se* $w = R_1 \cdots R_n\, S_1$ *e* $R_1 \neq \cdots \neq R_n$, *então* $\mathcal{A}_S$ *contém* $i_S \xrightarrow{\epsilon} i_w \xrightarrow{R_1} f_w^1 \xrightarrow{R_2} f_w^1 \cdots f_w^n \xrightarrow{S_1} f_w^{n+1} \xrightarrow{\epsilon} f_S$

  onde assume-se que todos os f_w^i, i_w *são distintos.*

Como trata-se de papéis concretos, não há a necessidade de considerar com o caminho inverso das transições pois papéis concretos não podem ser invertidos [2].

Então define-se $\mathcal{B}_S$ *de forma indutiva em cima de* $\prec$*:*

- *se* R *é mínimo em relação a* $\prec$, *ou seja, não existe* R' *tal que* $R' \prec R$, *definimos* $\mathcal{B}_S := \mathcal{A}_S$
- *senão,* $\mathcal{B}_S$ *é a união disjunta de* $\mathcal{A}_S$ *com a cópia* $\mathcal{A}'_S$ *de* $\mathcal{A}_S$ *para cada transição* $p \xrightarrow{R_i} q$ *com* $S \neq R$. *Para cada transição, adicionamos transições-*ϵ *de* p *para o estado inicial em* $\mathcal{A}'_S$ *e do estado final em* $\mathcal{A}'_S$ *para* q, *então fazemos* i_s *o único estado inicial e* f_s *o único estado final em* $\mathcal{B}_S$.

Como existem algoritmos para se raciocinar com a linguagem OWL 2 e com as construções adicionadas à linguagem tOWL, podemos utilizar uma ontologia construída com a versão modificada da linguagem tOWL e raciocinar com esta ontologia para obtermos conhecimentos implícitos e verificarmos consistência da ontologia.

5

Estudo de caso: Utilizando uma ontologia temporal para análise de ocorrências aéreas

"The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge."

– Stephen Hawking

Este Capítulo apresentará um estudo de caso para auxiliar na validação da linguagem estendida no Capítulo 4, sendo o objeto de estudo a representação de ocorrências aéreas, tais como acidentes, incidentes, etc encontradas em bases de dados públicas disponíveis em sítios na internet.

Para contextualização e esclarecimento oportuno, foram realizados durante o processo de construção da extensão da ontologia proposta, testes exaustivos com fragmentos de *benchmarks* específicos para análise e avaliação de funcionamento e desempenho de ontologias. Finalizada esta etapa de testes iniciais, optou-se por validar os resultados obtidos nesta proposta em uma situação real, permitindo analisar sua aplicabilidade na área de Web Semântica em geral no futuro.

O Capítulo foi dividido como: a Seção 5.1 apresenta como foi realizada a aquisição de dados, bem como a concepção da base de dados. Os dados foram obtidos a partir de uma base de dados aberta ao público que contém informações a respeito de acidentes, incidentes e ocorrências aéreas em geral. Na Seção 5.2 veremos como é realizada a criação da ontologia, quais aspectos importantes a serem considerados e como validar se a ontologia alcançou os requisitos esperados. A Seção 5.3 apresenta a ontologia construída e detalhes de sua implementação. A Seção 5.4 descreve respostas às *competency questions* criadas

para verificar a eficiência da ontologia. A Seção 5.5 apresenta exemplos de como raciocinar com esta ontologia.

5.1 Aquisição de Dados

Para o estudo de caso deste trabalho, serão utilizados dados de ocorrências na aviação relacionadas à segurança. Esta base de dados foi escolhida devido ao fato de serem dados abertos e extremamente relevantes para aviação civil mundial.

5.1.1 Formato dos dados

Os dados foram coletados a partir do sítio *Aviation Safety Network*[1]. Esta base de dados possui mais de 15 mil ocorrências, ocorridas entre os anos de 1919 a 2015. A Tabela 5.1 apresenta os atributos contidos para cada ocorrência contida na base de dados e seu respectivo significado. Além disso, cada voo possui informações sobre as suas fases de voo e o período associado à cada fase, estas informações foram retiradas do sítio *Flight Radar 24*[2]. A Tabela 5.2 apresenta as fases do voo e as características de cada fase.

Categorias das Ocorrências

O número da categoria indica o dano ocorrido na aeronave, o número 1 representa perda total e o número 2 representa que o dano é reparável. A perda total ocorre quando a aeronave é danificada e não pode ser consertada devido ao custo envolvido ser muito elevado. Também se considera perda total quando a aeronave está desaparecida, ou a busca pelos destroços foi encerrada sem encontrar a aeronave, ou, onde a aeronave foi danificada substancialmente e está inacessível. Por exemplo, uma ocorrência A1 significa um acidente que resultou em perda total da aeronave (ex.: Acidente da *Germanwings*[3] em março de 2015.).

Para a realização da coleta dos dados, foi desenvolvido um *web crawler* automático capaz de capturar cada ocorrência, e em seguida, separar cada um dos atributos contidos em cada ocorrência.

5.2 Metodologia para construção da ontologia

Algumas regras fundamentais para a criação de ontologias, de acordo com [36], são designadas como:

[1] http://aviation-safety.net/
[2] http://www.flightradar24.com
[3] https://pt.wikipedia.org/wiki/Voo_Germanwings_9525

Tabela 5.1: Atributos contidos na base de dados do *Aviation Safety Network*.

Campo	**Significado**
Data	Data da ocorrência, de acordo com o horário local
Horário	Horário da ocorrência
Tipo	Fabricante e modelo exato envolvido na ocorrência
Operador	Companhia aérea, organização ou indivíduo operando a aeronave no horário da ocorrência
Registro	A marca de registro da aeronave
C/n / msn	Número de série do fabricante/produção
Primeiro Voo	Data em que a aeronave envolvida na ocorrência realizou seu primeiro voo.
Tripulação	Número de membros da tripulação e número de fatalidades entre os mesmos devido à ocorrência
Passageiros	Número de passageiros e número de fatalidades entre os mesmos devido à ocorrência
Total	Soma do número de passageiros e número de membros da tripulação
Casualidades em terra	Número de casualidades em terra, resultadas direto da ocorrência
Casualidades por colisão	Número de casualidades por colisão, em outra aeronave envolvida na ocorrência
Dano	O dano causado à aeronave. Este campo pode possuir os seguintes valores: (i) Perda Total, (ii) Substancial, (iii) Mínimo, (iv) Nenhum e (v) Desconhecido
Local	Cidade ou aeroporto onde houve a ocorrência. Em alguns casos, o nome de uma montanha ou mar/oceano é utilizado.
Fase	Fase do Voo
Natureza	Natureza do Voo
Aeroporto de partida	Último aeroporto de partida antes da ocorrência
Aeroporto destino	Aeroporto de destino agendado.
Número do Voo	Número do voo, determinado pela companhia aé rea
Destino da aeronave	Define qual o destino na aeronave após a ocorrência, por exemplo, se a aeronave será reparada ou descartada.

Tabela 5.2: Descrição das fases típicas de um voo.

Fase do voo	**Descrição**
Takeoff	Decolagem (*takeoff)* é a fase do voo em que a aeronave realiza a transição entre o movimento o solo e o voo em ar, geralmente se iniciando na pista. A velocidade de decolagem varia de acordo com a densidade do ar, peso da aeronave e configuração da aeronave. Geralmente, a velocidade para aeronaves do tipo *jetliners* está na faixa de 130 a 155 nós (240-285 km/h).
Climb	Após a decolagem, a aeronave deve subir até uma certa altitude (aproximadamente 30.000 pés ou 10 km) para que possa iniciar a fase de cruzeiro de forma segura e econômica. O ângulo de ataque e o empuxo do motor são simultaneamente aumentados para produzir a subida (*climb*).
Cruise	O cruzeiro (*cruise*) é a fase em que o voo é mais eficiente em relação ao uso de combustível. Geralmente esta fase corresponde à maior parte do voo. A velocidade típica para voos comerciais está entre 475 e 500 nós (878-926 km/h).
Descent	Nesta etapa a aeronave diminui sua altitude. A descida (*descent*) é uma etapa essencial ao se aproximar do momento de aterrissagem. O piloto controla o ângulo de descida variando a potência do motor e o ângulo de inclinação.
Landing	A aterrisagem (*landing*) é a última parte do voo, quando a aeronave retorna ao solo. A velocidade deve ser reduzida para esta fase ser realizada com sucesso, isto é realizado diminuindo o empuxo e/ou induzindo a um maior arrasto usando os *flaps*, freios e o trem de pouso.

1. Não existe uma maneira correta de modelar um domínio, existem sempre outras alternativas. A melhor solução depende da aplicação que se pretende construir e as possíveis extensões que poderão ser implementadas futuramente;

2. O desenvolvimento de uma ontologia é um processo iterativo;

3. Conceitos na ontologia devem ser próximos aos objetos, físicos ou lógicos, e relacionamentos no domínio de interesse. Geralmente estes são representados por substantivos (objetos) ou verbos (relacionamentos) em frases que se quer descrever o domínio.

Existem vários métodos para se criar uma ontologia [49], neste trabalho, a ontologia foi criada seguindo alguns passos, de acordo com a proposta de [36], como mostrado a seguir.

5.2.1 Determinando o domínio e escopo da ontologia

Antes de iniciar o desenvolvimento da ontologia, é necessário responder as seguintes perguntas:

a) Qual o domínio será abrangido pela ontologia? R: O domínio da ontologia deste trabalho são as ocorrências aéreas.

b) Para que a ontologia será usada? R: A ontologia será usada para representar tais ocorrências aéreas e raciocinar em cima delas.

c) Que tipos de perguntas a ontologia deve ser capaz de responder? R: Deverá responder perguntas das quais estejam envolvida alguma característica correlacionada de tempo. Estas questões devem envolver uma maior complexidade e que ainda não seriam atendidas por métodos semelhantes de obtenção de informação (ex.: Linguagens de consultas estruturadas para banco de dados). A seguir são listadas algumas possibilidades como exemplo para estas perguntas, a saber:

   1 Quais acidentes ocorreram simultaneamente em um mesmo período de tempo?

   2 Onde ocorreu o maior número de ocorrências em um determinado período?

   3 Qual tipo de aeronave esteve envolvida na maior quantidade de ocorrências?

   4 Em qual fase do voo ocorrem mais ocorrências?

   5 É possível encontrar algum tipo de padrão entre período e número de acidentes aéreos?

Para construir uma ontologia para as ocorrências aéreas é necessário determinar quais relacionamentos e atributos são relevantes para o estudo de caso e para construir a ontologia temporal devemos analisar quais aspectos temporais devem ser considerados relevantes e quais devem ser desconsiderados para evitar uma complexidade desnecessária.

Desta feita, neste trabalho foram elaboradas *competency questions*, como apresentadas em [17] e descritas anteriormente, como sendo estas uma lista de questões que em uma base de conhecimento, utiliza uma ontologia que deve ser capaz de responder. Estas perguntas servem como um teste para avaliar se uma ontologia é apropriada ou não para um determinado domínio. Por exemplo, em um domínio que abrange vinho e comida, alguns exemplos de *competency questions* são: "Qual a melhor escolha de vinho para carne grelhada?" "Quais características de um vinho afetam sua adequação à um determinado prato?". Baseando-se nesta questões é possível determinar quais características/relacionamentos são fundamentais para a ontologia desejada.

5.2.2 Reutilizar ontologias existentes

Um dos objetivos da Web Semântica é facilitar a busca de informações na Web, para facilitar este processo é importante que as ontologias sejam reutilizadas para que o formato em que a informação se encontra seja o mais unificado possível. Não existe uma ontologia já pronta que satisfaça todos os requisitos do nosso domínio, mas existem algumas ontologias que contém conceitos de aviação e que podem servir de inspiração para este trabalho [39, 40]. A ontologia CRIISTAL [39], por exemplo, é utilizada para descrever alguns conceitos da aeronáutica, como fases do voo, condições climáticas, nível de combustível, etc., entretanto não atende às demandas temporais das quais são necessárias para o caso em análise.

5.2.3 Enumerar termos importantes para a ontologia

Nesta etapa da construção da ontologia devemos listar todos os termos que gostaríamos de utilizar em afirmações ou explicar à um usuário.

Estes termos foram construídos a partir dos atributos das Tabelas 5.1 e 5.2 e das *competency questions* elaboradas na primeira etapa do desenvolvimento da ontologia.

5.2.4 Definir classes e a hierarquia de classes

Existem diversos meios de se desenvolver a hierarquia de classes. Alguns deles são apresentados em [50] e listados a seguir:

Top-Down Processo de desenvolvimento que inicia com a definição de conceitos mais gerais do domínio e em seguida realiza a especialização dos conceitos.

Bottom-up Processo de desenvolvimento que começa com a definição de casos mais específicos, com as folhas da hierarquia, e em seguida realiza o agrupamento destas classes em conceitos mais gerais.

Combinação Um processo de desenvolvimento é uma combinação quando utiliza a abordagem *top-down* e a *bottom-up* na mesma construção. Os conceitos mais notáveis são definidos primeiro e então eles são generalizados ou especializados apropriadamente. O desenvolvimento se inicia tanto com termos *top-down*, quanto termos mais especializados.

Neste trabalho foi utilizado a método de combinação, para que fosse possível selecionar as classes que são consideradas mais importantes e então expandir ou especializar a partir das mesmas.

5.2.5 Definir as propriedades de classes — *slots*

Após definir as classes, se faz necessário definir as diferentes estruturas internas dos conceitos, ou seja, as propriedades destas classes. Diversos tipos de propriedades de objetos podem se tornar *slots* em uma ontologia: propriedades intrínsecas (ex: o sabor de um vinho), propriedades extrínsecas (ex: a área de origem de um vinho), partes de um objeto, caso este seja estruturado (ex: os pratos de uma refeição), e relações com outros indivíduos (ex: o fabricante de um vinho).

Todas as subclasses de uma classe herda os *slots* da classe pai. Um *slot* deve ser ligado à classe mais geral que pode ter tal propriedade.

5.2.6 Definir os aspectos dos *slots*

Slots possuem diferentes aspectos para descrever tipos de valores, valores permitidos, o número de valores (cardinalidade) e outras características que o *slot* pode possuir.

Cardinalidade: A cardinalidade define quantos valores um *slot* pode possuir. Alguns sistemas distinguem apenas entre cardinalidade única (no máximo um valor) e cardinalidade múltipla (qualquer número de valores) [36].

Alguns sistemas permitem a especificação de uma cardinalidade mínima e máxima para descrever o número de valores do *slot* [36]. Cardinalidade mínima de N significa que um *slot* deve possuir pelo menos N valores. A cardinalidade máxima de M indica que o *slot* deve possuir no máximo M valores. Algumas vezes é interessante

definir a cardinalidade máxima para 0, indicando que um *slot* não possui nenhum valor para uma subclasse particular.

Tipo de valor: Determina que tipos de valores podem preencher os *slots*. Os tipos mais comuns usados são: *String*, *Number*, *Boolean*, *Enumerated*. Existe também o *slot* de tipo *Instance*, que permite a definição de relações entre indivíduos.

Domínio e abrangência do *slot*: As classes permitidas por *slots* do tipo *Instance* são referidas pela abrangência do *slot*. As classes das quais um *slot* está vinculado ou as classes cujas propriedades são descritas por um *slot* são chamadas o domínio de um *slot* [36]. Ao definir um domínio ou abrangência de um *slot*, é necessário encontrar a classe ou classes mais gerais que podem representar o domínio ou abrangência de um *slot*. Por outro lado, não é recomendado que um domínio ou abrangência seja muito geral, todas as classes no domínio de um *slot* devem ser descritas pelo *slot* e todas as instâncias de todas as classes na abrangência de um *slot* devem ser potencialmente utilizadas para preencher um *slot*.

5.2.7 Criar instâncias

A última etapa é a criação de instâncias das classes na hierarquia. Cada classe foi instanciada com dados da base de dados construída a partir de ocorrências do transporte aéreo.

5.3 A construção da ontologia

A Figura 5.1 mostra a ontologia temporal para representar ocorrências aéreas, seguindo a proposta apresentada como referência, adicionando temporalidade à OWL 2.

Os atributos representados por retângulos completos são do tipo *Instância* e os representados por retângulos pontilhados possuem *tipos de dados* (*String*, *Number*, *Boolean*, *Enumerated*...). Existem três grandes classes: a *Ocorrência*, o *Voo* e a *Aeronave*. A *Ocorrência* é ligada à classe *Voo* e a classe *Voo* é ligada à classe *Aeronave*. A Figura 5.1 também apresenta todas as propriedades que conectam as classes aos seus predicados. Para facilitar a visualização, na Figura 5.1, foi omitido detalhes das classes de menor importância para esse contexto, como as classes *Aeroporto de Destino*, *Aeroporto de Partida*, *Local*, etc.

Os aspectos temporais desta ontologia são: a data do primeiro voo de uma aeronave, o horário previsto de chegada e de partida, a data e o horário da ocorrências e o histórico do voo. O histórico do voo é divido em fases, conforme mostra a Figura 5.2. Este diagrama foi construído baseado na Tabela 5.1. Cada estágio possui duas possíveis saídas: Sucesso

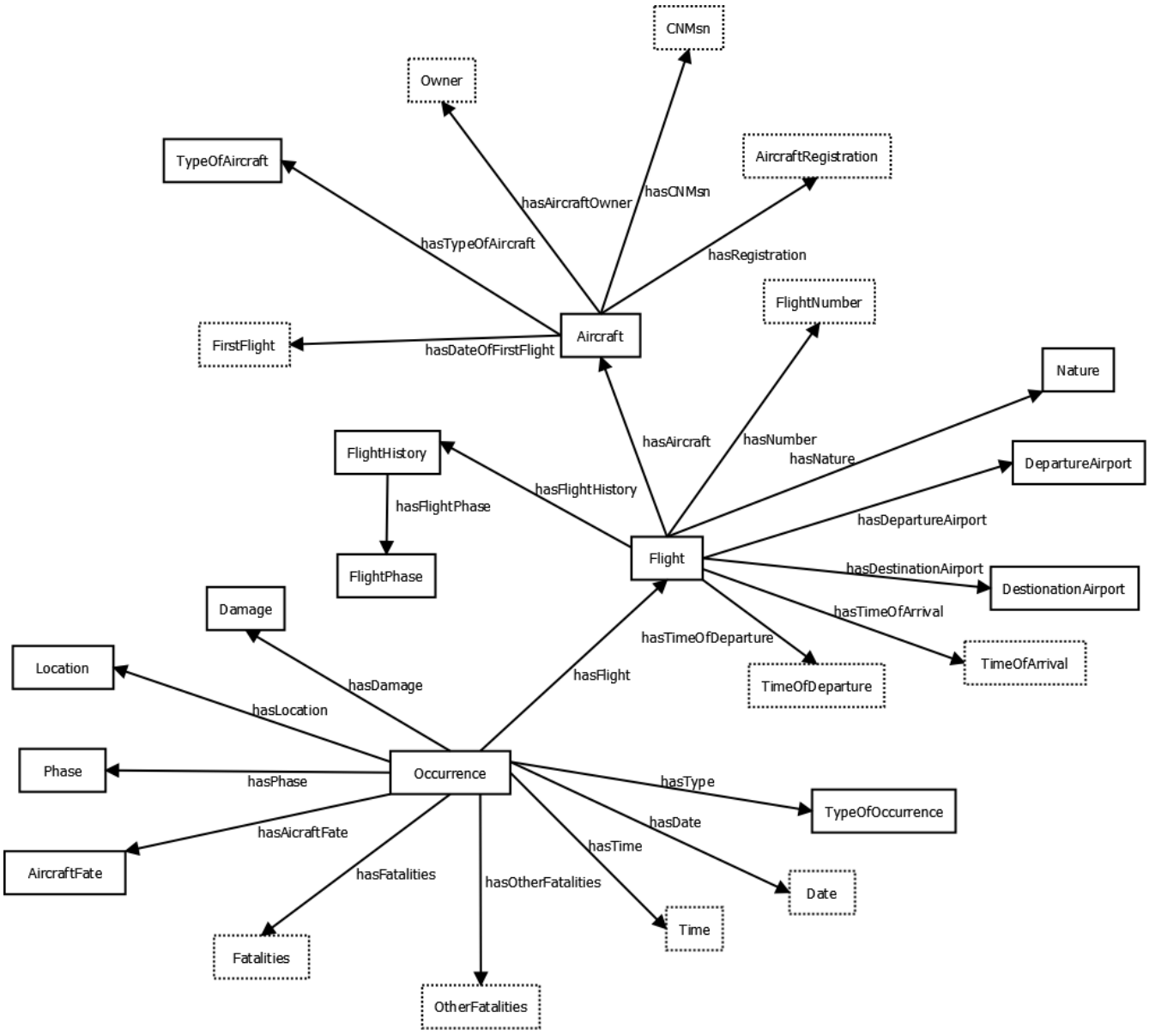

Figura 5.1: Ontologia construída para ocorrências aéreas.

ou Falha. Quando ocorre o Sucesso, o voo segue as suas fases normais. Ao ocorrer Falha, o voo segue para uma situação de emergência. Quando uma situação de emergência falha, o voo muda para um estágio *Abort*, este estágio significa que a recuperação da situação de emergência falhou e um acidente/incidente pode ter ocorrido. Cada estágio deste diagrama é representado na ontologia como uma *TimeSlice*.

5.3.1 Implementação da TBox

Conforme mencionado anteriormente, a TBox permite que se crie as terminologias para a ontologia. Neste estudo de caso, a TBox representa as informações conceituais sobre as ocorrências aéreas. Primeiro declara-se todas as classes da ontologia:

```
Class( :Occurrence )
Class( :Flight )
Class( :Aircraft )
Class( :TypeofOccurence )
Class( :AircraftFate )
Class( :Phase )
Class( :Location )
Class( :Damage )
Class( :Airport )
Class( :Nature )
Class( :FlightHistory )
Class( :FlightPhase )
Class( :TypeOfAircraft )
Class( :Fatalities )
```

Em seguida define-se os domínios e as abrangências das propriedades que ligam classes às instâncias:

```
ObjectPropertyDomain( :hasDamage :Occurrence )
ObjectPropertyRange( :hasDamage :Damage )
ObjectPropertyDomain( :hasLocation :Occurrence )
ObjectPropertyRange( :hasLocation :Location )
ObjectPropertyDomain( :hasPhase :Occurrence )
ObjectPropertyRange( :hasPhase :FlightPhase )
ObjectPropertyDomain( :hasAircraftFate :Occurrence )
ObjectPropertyRange( :hasAircraftFate :AircraftFate )
ObjectPropertyDomain( :hasType :Occurrence )
```

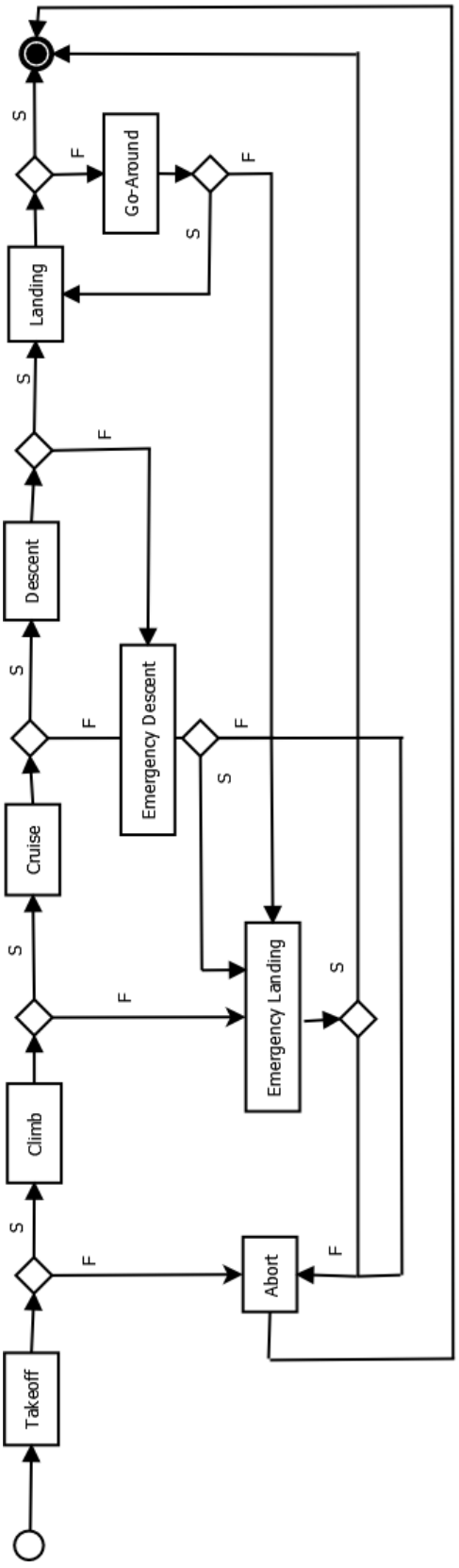

Figura 5.2: Diagrama de sequência para as fases do voo.

```
ObjectPropertyRange( :hasType :TypeOfOccurrence )
ObjectPropertyDomain( :hasFlight :Occurrence )
ObjectPropertyRange( :hasFlight :Flight )
ObjectPropertyDomain( :hasFatalities :Occurrence )
ObjectPropertyRange( :hasFatalities :Fatalities )
ObjectPropertyDomain( :hasOtherFatalities :Occurrence )
ObjectPropertyRange( :hasOtherFatalities :Fatalities )
...
```

Define-se também os domínios e as abrangências das propriedades que ligam classes à tipo de dados:

```
DataPropertyDomain( :hasTime :Occurence )
DataPropertyRange( :hasTime xsd:time )
DataPropertyDomain( :hasDate :Occurence )
DataPropertyRange( :hasDate xsd:date )

DataPropertyDomain( :hasTimeOfDeparture :Flight )
DataPropertyRange( :hasTimeOfDeparture xsd:dateTime )
DataPropertyDomain( :hasTimeOfArrival :Flight )
DataPropertyRange( :hasTimeOfArrival xsd:dateTime )
DataPropertyDomain( :hasNumber :Flight )
...
```

A maioria das propriedades de objetos possuem cardinalidade igual a um, nesta ontologia proposta. Isso é definido da seguinte forma:

```
ClassAssertion( ObjectMaxCardinality( 1 :hasLocation ) :Occurrence )
ClassAssertion( ObjectMaxCardinality( 1 :hasDamage ) :Occurrence )
ClassAssertion( ObjectMaxCardinality( 1 :hasPhase ) :Occurrence )
ClassAssertion( ObjectMaxCardinality( 1 :hasAircraft ) :Occurrence )
ClassAssertion( ObjectMaxCardinality( 1 :hasType ) :Occurrence )
ClassAssertion( ObjectMaxCardinality( 1 :hasFlight ) :Occurrence )
...
```

Adicionamos a seguinte construção para indicar que as fatalidades podem ser passageiros, tripulação ou outros (pessoas no solo, por exemplo):

```
SubClassOf ( :Passenger :Fatalities )
SubClassOf ( :Crew :Fatalities )
SubClassOf ( :OtherFatalities :Fatalities )
```

Cada fase do voo é representada como subclasse da classe *FlightPhase*:

```
Class( :TakeOff partial :FlightPhase )
Class( :Climb partial :FlightPhase )
Class( :Cruise partial :FlightPhase )
Class( :Descent partial :FlightPhase )
Class( :Landing partial :FlightPhase )
Class( :Abort partial :FlightPhase )
Class( :Go-Around partial :FlightPhase )
Class( :EmergencyLanding partial :FlightPhase )
Class( :EmergencyDescent partial :FlightPhase )
```

E todas as fases são disjuntas entre si:

```
DisjointClasses( :TakeOff, :Climb, :Cruise, :Descent, :Lading, :Abort,
 :Go-Around, :EmergencyDescent, :EmergencyLanding )
```

As classes de todas as *timeslices* do histórico de voo são definidas como:

```
Class(FlighHistory_TS complete
 restriction(timeSliceOf(someValuesFrom FlightHistory)))
```

Cada fase do voo também possui sua própria *timeslice*, definida como:

```
Class(TakeOff_TS complete restriction(timeSliceOf(someValuesFrom TakeOff)))
Class(Climb_TS complete restriction(timeSliceOf(someValuesFrom Climb)))
Class(Cruise_TS complete restriction(timeSliceOf(someValuesFrom Cruise)))
Class(Descent_TS complete restriction(timeSliceOf(someValuesFrom Descent)))
Class(Landing_TS complete restriction(timeSliceOf(someValuesFrom Landing)))
Class(Abort_TS complete restriction(timeSliceOf(someValuesFrom Abort)))
Class(Go-Around_TS complete restriction(timeSliceOf(someValuesFrom Go-Around)))
Class(EmergencyLanding_TS complete
 restriction(timeSliceOf(someValuesFrom EmergencyLanding)))
Class(EmergencyDescent_TS complete
 restriction(timeSliceOf(someValuesFrom EmergencyDescent)))
```

Para cada estágio, define-se uma propriedade funcional para ligar cada *timeslice* do histórico de voo à fase que pertence a este histórico:

```
ObjectProperty(:takeOff domain(:FlighHistory_TS) range(:TakeOff_TS))
 FunctionalObjectProperty(takeOff)
  ObjectProperty(:climb domain(:FlighHistory_TS) range(:Climb_TS))
 FunctionalObjectProperty(climb)
  ObjectProperty(:cruise domain(:FlighHistory_TS) range(:Cruise_TS))
 FunctionalObjectProperty(cruise)
  ObjectProperty(:descent domain(:FlighHistory_TS) range(:Descent_TS))
 FunctionalObjectProperty(descent)
 ...
```

Em seguida, definiu-se o *fluent* `inPhase`, que aponta cada *timeslice* de um voo para a fase em que este se encontra.

```
 FluentObjectProperty(inPhase domain(
 restriction (timeSliceOf(someValuesFrom Flight))) range(
 restriction (timeSliceOf (someValuesFrom FlightPhase))))
```

As *timeslices* do `Flight History` são definidas pela sequência de fases que um voo pode seguir. Isto depende do uso de uma cadeia de composição de papéis. A seguinte construção define que a fase inicial do voo deve sempre ser a fase de *take off*.

```
Class(FlightHistory_TS partial restriction(
DataSomeValuesFrom(DataPropertyChain(takeOff time), time, starts))))
```

5.3.2 Implementação da ABox

Na ABox são representadas as informações a respeito de um voo específico, sendo que para fins de exemplo foi escolhido o voo 9525 da *Germanwings* (Figura 5.3).

Inicialmente, cria-se a instância do voo 4U9525:

```
Individual (:iFlight4U9525 type(:Flight))
```

Podemos utilizar uma restrição de cardinalidade qualificada para representar que o número máximo possível de fatalidades neste acidente é de 150 pessoas e de 2 tripulantes, pois esta é a capacidade máxima de uma aeronave do tipo Airbus 320 [15]. Tal representação não era possível na tOWL original:

```
 ClassAssertion(
   ObjectMaxCardinality( 2 :hasFatalities :Crew )
   :iFlight4U9525
```

Criminal Occurrence description

Status:	Preliminary
Date:	**Tuesday 24 March 2015**
Time:	ca 10:41

Type:	Airbus A320-211
Operator:	Germanwings
Registration:	D-AIPX
C/n / msn:	147
First flight:	1990-11-29 (24 years 4 months)
Total airframe hrs:	58300
Cycles:	46700
Engines:	2 CFMI CFM56-5A1
Crew:	Fatalities: 6 / Occupants: 6
Passengers:	Fatalities: 144 / Occupants: 144
Total:	Fatalities: 150 / Occupants: 150
Airplane damage:	Damaged beyond repair
Location:	17 km (10.6 mls) SW of Barcelonnette (France)
Phase:	En route (ENR)
Nature:	International Scheduled Passenger
Departure airport:	Barcelona-El Prat Airport (BCN/LEBL), Spain
Destination airport:	Düsseldorf International Airport (DUS/EDDL), Germany
Flightnumber:	4U9525

Narrative:

An Airbus A320 operated by Germanwings was destroyed in an accident in a mountainous area in southern France. All 144 passengers and six crew members were killed.

Flight 4U9525 departed Barcelona, Spain at 10:00 hours local time (09:00 UTC) on a regular passenger service to Düsseldorf, Germany. The flight reached its cruising altitude of FL380 at 10:27 hours. At 10:30 hours the flight was cleared direct to the IRMAR waypoint, which was confirmed by the flight: "Direct IRMAR, Merci 18G." This was the last radio contact with the flight.

Figura 5.3: Dados do voo 9525 da Germanwings.

```
)

ClassAssertion(
  ObjectMaxCardinality( 150 :hasFatalities :Passenger )
  :iFlight4U9525
)
```

Em seguida foram criadas todas as fases que esse voo possui, em formato de *timeslices*, sofrendo um acidente na fase de Cruzeiro (*En route*), consideramos que ele foi para *Emergency Descent* antes de colidir.

```
Individual (:iFlightHistory_TS1 type (:FlightHistory_TS)
value (:timeSliceOf :iFlightHistory_TS1)
value (:takeOff :iTakeOff1_TS1)
value (:climb :iClimb1_TS1)
value (:cruise :iCruise1_TS1)
value (:descent :iDescent1_TS1)
value (:abort :iAbort1_TS1)
value (:goAround :iGoAround1_TS1)
value (:emergencyDescent :iEmergencyDescent1_TS1))
```

Criou-se a *timeslice* associada a primeira fase do voo, *take off*, conforme mostra a Figura 5.4, descrita a seguir:

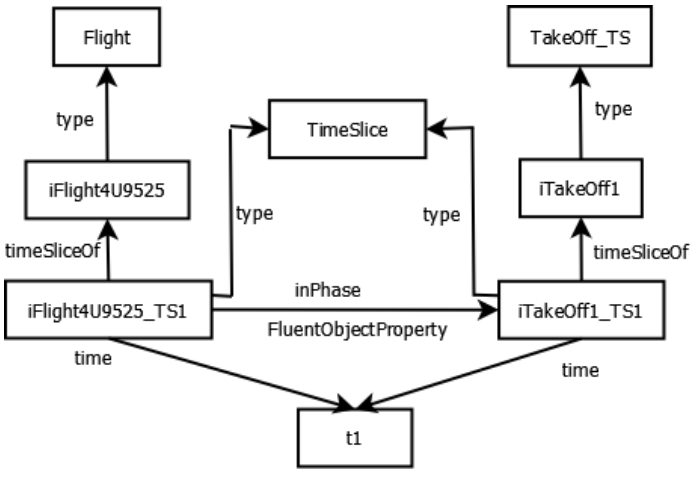

Figura 5.4: Associação do voo 4U9525 à fase de *take off*. Informações retiradas do *Aviation Safety*.

```
Individual(:t1 type(:Interval))
```

```
Individual(:iTakeOff1
type(:TakeOff_TS))
Individual(:iTakeOff1_TS1
type(:TimeSlice)
value(:timeSliceOf :iTakeOff1)
value(:time :t1))
Individual(:iFlight4U9525_TS1
type(:TimeSlice)
value(:timeSliceOf :iFlight4U9525)
value(:time :t1)
value(:inPhase :iTakeOff1_TS1))
```

Da mesma forma, deve-se criar as outras *timeslices* para cada fase do voo, até a fase em que ocorre o acidente.

5.4 Análise da Ontologia

Para analisar se a ontologia é adequada, devemos verificar se ela é capaz de responder as *competency questions* mencionadas anteriormente:

1 Quais acidentes ocorreram simultaneamente em um mesmo período?

R: A ontologia representa o tempo em que um acidente ocorreu, é possível isolar períodos em semanas e meses, analisando quais acidentes ocorrem em um mesmo período. Os Gráficos das Figuras 5.5 e 5.6 foram gerados a partir de dados da ontologia implementada e mostram o número de acidentes por mês e por dia da semana, respectivamente. É possível observar que o número de acidentes não apresenta variações significativas entre os meses e os dias da semana, portanto, não é possível chegar e conclusões relacionadas a estes aspectos.

2 Onde ocorreu o maior número de ocorrências em um determinado período?

R: A ontologia possui uma propriedade para representar o local onde ocorreram os incidentes e acidentes, sendo assim é possível analisar localização e período das ocorrências. A Tabela 5.3 exibe os locais em que ocorreram a maior quantidade de acidentes no mês de abril desde 1919.

3 Qual tipo de aeronave esteve envolvida na maior quantidade de ocorrências?

R: A ontologia armazena o tipo de aeronave envolvida no acidente através da classe `Aircraft`, então é possível encontrar os tipos de aeronaves que foram envolvidas em

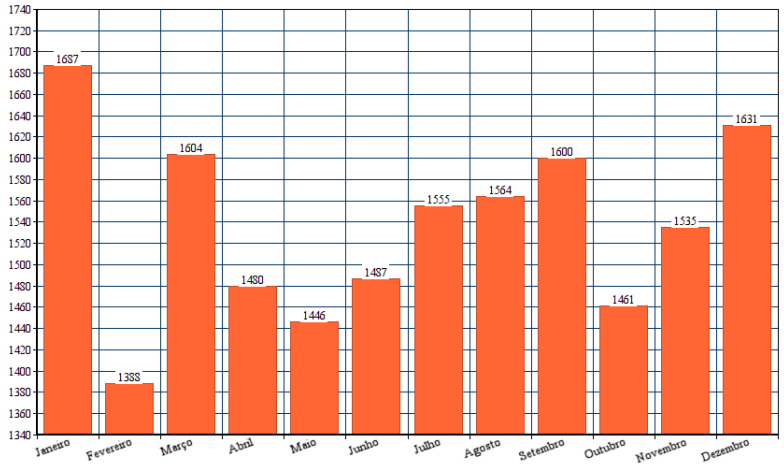

Figura 5.5: Número de acidentes aéreos por mês.

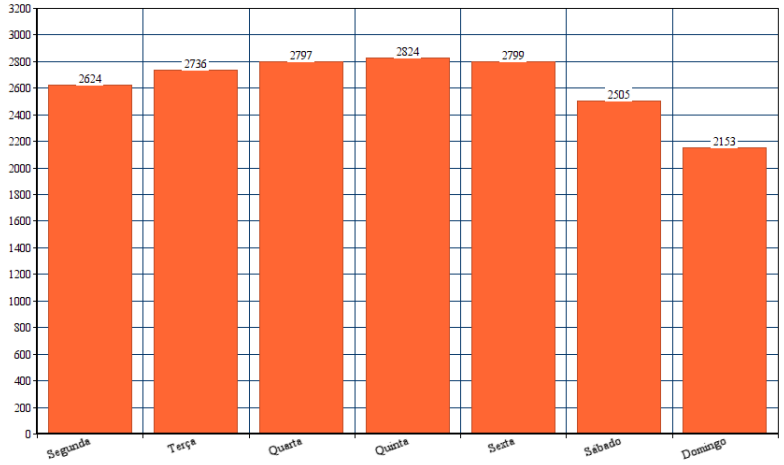

Figura 5.6: Número de acidentes aéreos por dia da semana.

Tabela 5.3: Locais com maior número de acidentes no mês de abril.

Local	**Número de acidentes**
Estados Unidos	264
Russia	84
Reino Unido	49
India	49

um maior número de ocorrências. A Tabela 5.4 exibe esta informação, exibindo os tipos de aeronaves mais envolvidas em acidentes. A aeronave com o maior número de acidentes é a Douglas C-47-DL (DC-3), utilizada durante a Segunda Guerra Mundial, o que explica o elevado número de acidentes. A Figura 5.7 apresenta o número de acidentes por ano para a aeronave Douglas C-47-DL (DC-3) e para as aeronaves das fabricantes Boeing e Airbus. É possível observar um pico no número de acidentes da aeronave Douglas C-47-DL (DC-3) na época em que ocorreu a Segunda Guerra Mundial.

Tabela 5.4: Aeronaves envolvidas no maior número de acidentes.

Tipo da aeronave	**Número de acidentes**
Douglas C-47-DL (DC-3)	411
Antonov 2R	212
Consolidated PBY-5A Catalina	210
de Havilland Canada DHC-6 Twin Otter 300	204
Douglas Dakota III (DC-3)	185

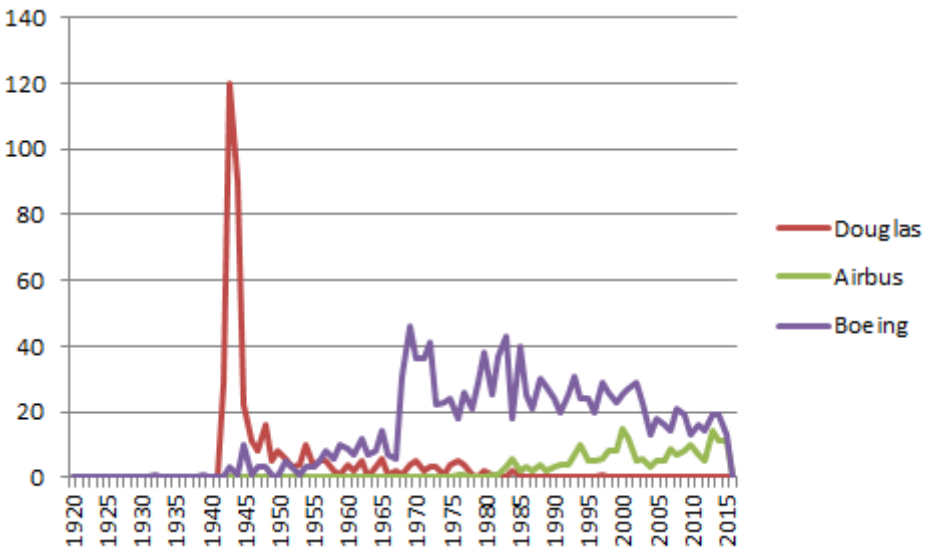

Figura 5.7: Número de acidentes por ano, considerando os tipos das aeronaves.

4 Em qual fase do voo ocorrem mais acidentes?

R: Esta ontologia armazena as fases do voo e em qual fase do voo aconteceu o acidente, desta forma podemos responder tal pergunta. A Figura 5.8 apresenta o gráfico relacionando o número de acidentes aéreos com a fase do voo respectiva. A fase de cruzeiro (*En Route*) é a fase com o maior número de acidentes e a aterrisagem (*Landing*) possui mais acidentes que a decolagem (*takeoff*).

5 É possível encontrar algum tipo de padrão entre período e número de acidentes aéreos?

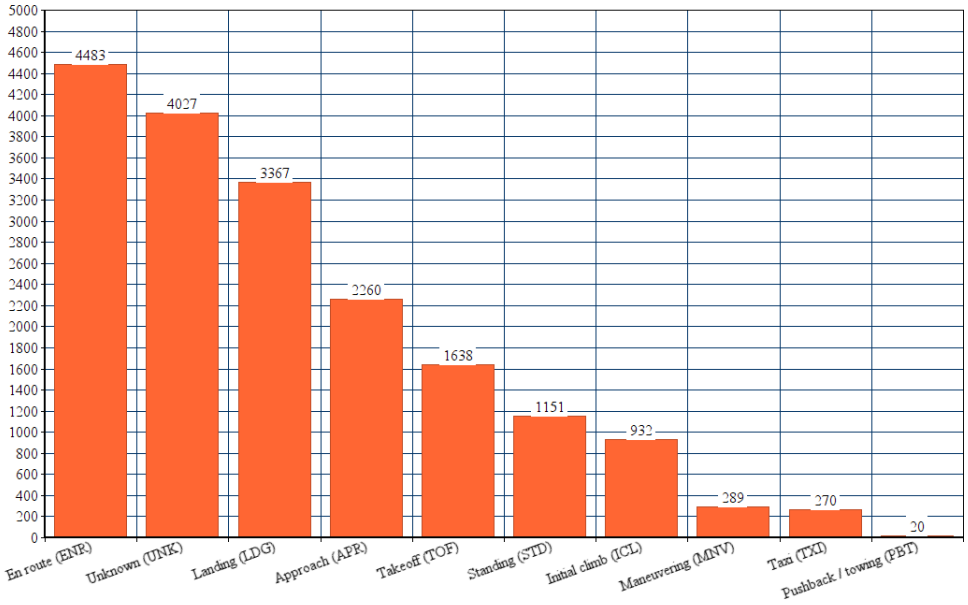

Figura 5.8: Número de acidentes aéreos por fase do voo.

R: Com esta ontologia sabemos quais os acidentes que ocorreram em um determinado período, portanto podemos verificar se existe uma relação entre o período analisado e o número de acidentes aéreos. Pela resposta da primeira pergunta, observamos que não é possível traçar tal relação em relação aos meses e semanas, porém podemos observar uma relação entre número de acidentes e diferentes fases do voo.

5.5 Raciocinando com a ontologia temporal

Conforme apresentado na Seção 4.3, a versão estendida da linguagem tOWL é decidível, isso significa que é possível raciocinar com a mesma. Nesta Seção apresentamos um exemplo de como é possível raciocinar com o tempo, utilizando a ontologia temporal para ocorrências aéreas apresentada neste trabalho. Na Seção 5.5.1 apresentamos como é possível raciocinar com as fases de um voo, já na Seção 5.5.2 mostramos como este raciocínio pode ser feito com elementos que não envolvem temporalidade.

5.5.1 Raciocinando com as fases de voo

Nesta seção, apresentamos como é possível inferir a fase de voo em que uma aeronave se encontra em um determinado momento. As diferentes fases e as possibilidades de

mudança de fase foram descritas na Figura 5.2. Conforme apresentado anteriormente, estas fases foram introduzidas à linguagem na forma de *timeslices*. Baseando-se nestas informações, quando não se tem os dados completos sobre as fases de voo, pode-se deduzi-las a partir das informações contidas na base de conhecimento utilizando mecanismos de raciocínio automático.

Para esta análise, será usado novamente o voo 4U9525 como exemplo (Figura 5.3). Este voo passou pelas seguintes fases de voo: *take off* → *climb* → *cruise* → *emergency landing* → *abort*. Supondo que tenhamos apenas informação a respeito dos intervalos de *take off* e *cruise*.

Representou-se o *timeslice* de *takeoff* do voo 4U9525 da seguinte forma (Esta representação já foi apresentada anteriormente, porém foi incluída novamente para permitir a comparação entre a representação de diferentes fases de voo):

```
Individual(:t1 type(:Interval))
Individual(:iTakeOff1
type(:TakeOff_TS))
Individual(:iTakeOff1_TS1
type(:TimeSlice)
value(:timeSliceOf :iTakeOff1)
value(:time :t1))
Individual(:iFlight4U9525_TS1
type(:TimeSlice)
value(:timeSliceOf :iFlight4U9525)
value(:time :t1)
value(:inPhase :iTakeOff1_TS1))
```

A fase de *takeoff* está associada a um intervalo $t1$. Uma *timeslice* de nome `iFlight4U9525_TS1` é criada para representar o voo neste intervalo, este voo se liga à *timeslice* `iTakeOff1_TS1` que representa a fase de *takeoff* no instante $t1$.

Temos também dados sobre a fase de *cruise*, podemos representar este conhecimento da mesma forma, associando esta fase ao intervalo $t3$:

```
Individual(:t3 type(:Interval))
Individual(:iCruise1
type(:Cruise_TS))
Individual(:iCruise1_TS1
type(:TimeSlice)
value(:timeSliceOf :iCruise1)
value(:time :t3))
```

```
Individual(:iFlight4U9525_TS3
type(:TimeSlice)
value(:timeSliceOf :iFlight4U9525)
value(:time :t3)
value(:inPhase :iCruise1_TS1))
```

O conhecimento contido nesta base de conhecimento é representado na Figura 5.9. Conforme observado na Figura 5.2, a única fase de voo que pode estar entre *takeoff* e *cruise* é a fase de *climb*. A base de conhecimento já possui informação à respeito do diagrama de sequência das fases de voo, portanto, é possível gerar o conhecimento implícito de que entre os intervalos $t1$ e $t3$ existe um intervalo $t2$ em que ocorreu a fase de *climb*. Este conhecimento implícito é demonstrado pela Figura 5.10.

Como também sabemos que ocorreu um acidente na fase de cruzeiro neste voo, é possível concluir que a próxima fase é a de *Emergency Descent*, que ocorre quando há algum problema na fase de cruzeiro. Da mesma forma, como se sabe que não houveram sobreviventes no acidente é possível concluir que a fase de *Emergency Landing* também falhou, portanto o voo encerrou na fase *Abort*.

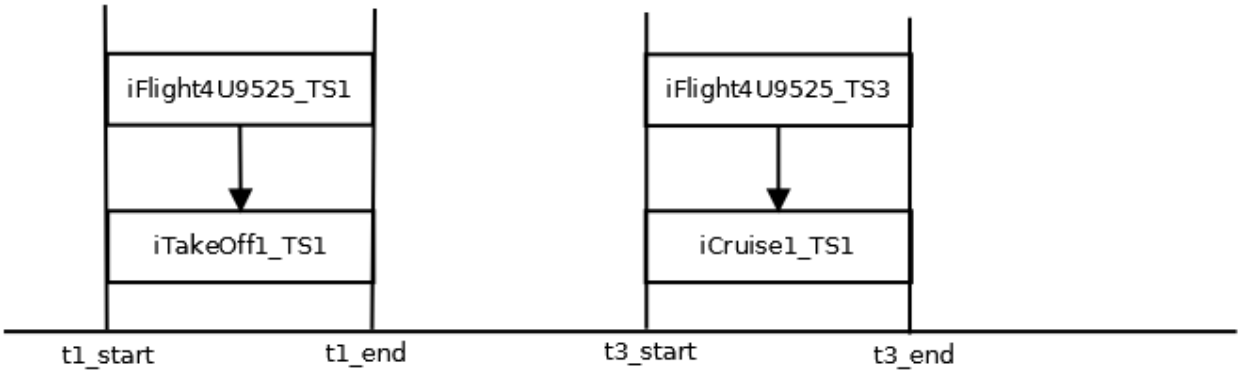

Figura 5.9: Conhecimento explícito contido na base de conhecimento à respeito do voo 4U9525.

Tal abordagem é interessante para casos em que não temos informações completas à respeito das fases de voo, fato que acontece bastante em informações contidas na *Web*, devido ao grande volume e baixa organização. Tal abordagem não seria possível com o uso de banco de dados pois o mesmo se baseia na Hipótese de Mundo Fechado. O gráfico da Figura 5.8 mostra que existe uma relação significativa entre o número de acidentes e a fase em que ocorreu o acidente, portanto é importante que uma base de conhecimento contenha esta informação, para que seja possível analisar quais aspectos tornam uma fase de voo mais propensa à acidentes que outras e tentar melhorar tais aspectos.

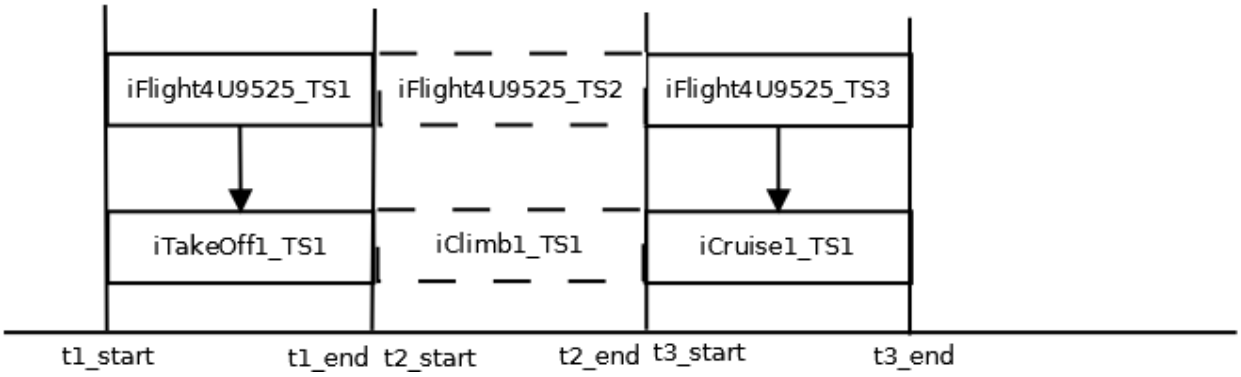

Figura 5.10: Conhecimento implícito extraído da base de conhecimento.

5.5.2 Raciocinando com outros aspectos da ontologia

Esta Seção apresenta outros conhecimentos implícitos que podem ser obtidos à partir da ontologia construída neste trabalho.

É possível criar regras na ontologia proposta para categorizar cada tipo de ocorrência. A criação da categoria *Acidente* é feita:

```
Individual (:Accident type(:TypeOfOccurrence))
```

Conforme mencionado anteriormente, um dos critérios que caracteriza um acidente é o fato da aeronave sofrer um dano grave/irreversível, como é o caso do voo 4U9525:

```
ObjectPropertyAssertion( :hasDamage :iFlight4U9525
:DamagedBeyondRepair )
```

Pela característica do dano à aeronave, é possível concluir que se trata de uma Ocorrência do tipo *Acidente*, portanto o seguinte conhecimento implícito poderia ser obtido, caso esta informação esteja faltando na base de conhecimento:

```
ObjectPropertyAssertion( :hasTypeOfOccurrence :iFlight4U9525
:Accident )
```

Desta forma, pode-se gerar bases de conhecimento completas a partir de fontes de informação que possam estar incompletas. De forma similar, podemos associar o tipo de Ocorrência com o número de fatalidades.

Outro elemento encontrado nesta ontologia é o *Destino da Aeronave* (*Aircraft Fate*) que determina o que irá acontecer com a aeronave após a ocorrência. Quando o dano que a aeronave recebe é tão grave que não permite conserto, assim como ocorreu no voo 4U9525, o destino da aeronave é *Written off* (Eliminada). Caso não tenhamos em nossa base de conhecimento informações sobre o destino da aeronave, pode-se facilmente gerar este conhecimento implícito ao obter informações sobre o dano sofrido pela aeronave.

```
ObjectPropertyAssertion( :hasAircraftFate :iFlight4U9525
:WrittenOff )
```

6

Considerações Finais

"We must die as egos and be born again in the swarm, not separate and self-hypnotized, but individual and related."

– Henry Miller

Neste trabalho, apresentamos um estudo da compatibilidade entre a linguagem temporal tOWL e a linguagem da Web Semântica OWL 2. A linguagem tOWL é baseada no fragmento $\mathcal{SHIND}(D)$ da Lógica de Descrição, enquanto a linguagem OWL 2 é baseada no fragmento $\mathcal{SROIQ}(D)$. Foi concluído que estas linguagens são incompatíveis entre si, isto significa que utilizar a linguagem tOWL junto à OWL 2, resultaria em uma perda de decidibilidade. O fato de uma linguagem ser indecidível, significa que não é possível raciocinar de forma precisa com a mesma, ou seja, podemos não ser capazes de obter conhecimentos implícitos e verificar a consistência utilizando uma ontologia construída com tal linguagem.

O fato que torna ambas linguagens incompatíveis é o uso de Domínios Concretos na linguagem tOWL. O uso de Domínios Concretos junto com o uso de nominais torna uma linguagem indecidível. Na OWL 2 temos o uso de nominais, como a linguagem tOWL exige o uso de Domínios Concretos, isto tornaria a linguagem indecidível. A solução encontrada neste trabalho foi substituir o uso de Domínios Concretos pelo uso de *Datatype Maps*, que permite o uso de tipos de dados, porém com uma abordagem diferente. A linguagem OWL 2 utiliza *Datatype Maps* para representar tipos de dados, portanto, foi possível fazer tal substituição sem perda da decidibilidade.

Para que essa substituição fosse possível, analisamos todas as estruturas que utilizavam Domínios Concretos e verificamos a possibilidade de modificar seu uso. Como resultado, adicionamos uma nova camada à tOWL, substituindo a camada de Domínios Concretos. Esta camada serviu como uma cola para tornar tOWL e OWL 2 compatíveis.

Apresentamos também os algoritmos para lidar com as construções desta nova camada, provando ser possível raciocinar com a versão modificada da linguagem tOWL.

Para verificar o funcionamento das modificações feitas na linguagem tOWL, apresentamos uma ontologia para representar ocorrências aéreas. Esta ontologia utiliza a capacidade de representação temporal da tOWL para representar os instantes em que houve a ocorrência aérea e as fases de voo correspondeste ao voo. Apresentamos também exemplos de como é possível raciocinar com tais aspectos temporais.

Existem diversas vantagens nesta adaptação feita da linguagem tOWL para a linguagem OWL 2: temos uma maior expressividade na linguagem, podendo representar construções complexas, tal como restrição de cardinalidade qualificada e podemos utilizar ferramentas desenvolvidas especialmente para a OWL 2.

6.1 Trabalhos Futuros

Com as modificações apresentadas neste trabalho à linguagem tOWL, é possível implementar um raciocinador automático capaz de raciocinar com o tempo, adaptando raciocinadores que já existem para a $\mathcal{SROIQ}$. Também é possível analisar novos aspectos do tempo a serem adicionados à linguagem tOWL, tal como a cardinalidade temporal. A cardinalidade temporal permitiria limitar em que instantes do tempo uma propriedade é válida, por exemplo, podemos definir que uma aeronave possui um limite em que pode ficar em atividade.

Além disso, é interessante verificar a possibilidade de se criar ontologias temporais para diferentes áreas, por exemplo, na área de economia, analisando como a situação de uma empresa varia em relação ao tempo. Também pretendemos comparar o método apresentado neste trabalho com algoritmos de mineração de processos. Com uma base de conhecimento sobre processos, podemos utilizar um raciocinador automático para montar os processos contidos em uma empresa.

Referências

[1] James F Allen. An interval-based representation of temporal knowledge. In *IJCAI*, volume 81, pages 221–226, 1981. 47

[2] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004. x, 21, 22, 24, 27, 52

[3] Franz Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003. x, 3, 10, 11, 12, 13, 14, 15, 18, 20

[4] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001. 2, 20, 22

[5] Max Black. The identity of indiscernibles. *Mind*, pages 153–164, 1952. 42

[6] Ronald Brachman and Hector Levesque. *Knowledge representation and reasoning*. Elsevier, 2004. 8

[7] Arthur Buchsbaum and Tarcisio Pequeno. O método dos tableaux generalizado e sua aplicação ao raciocínio automático em lógicas não clássicas. *Revista "O que nos faz pensar", Cadernos do Departamento de Filosofia da PUC-Rio*, 3:81–96, 1990. 18

[8] Francis M Cornford. *Plato's theory of knowledge: The theaetetus and the sophist*. Courier Dover Publications, 2003. 7

[9] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI magazine*, 14(1):17, 1993. 2, 8

[10] Giuseppe De Giacomo and Maurizio Lenzerini. Tbox and abox reasoning in expressive description logics. *KR*, 96:316–327, 1996. 10

[11] Martin A Fischler and Oscar Firschein. Intelligence: the eye, the brain, and the computer. 1987. 8

[12] F Gargiulo, G Zazzaro, G Romano, G Gigante, A Raggioli, and R Fusco. Aerospace information system based on semantic technonogies and ontology management. 2014. 5

[13] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014. 8

[14] Edmund L Gettier. Is justified true belief knowledge? *analysis*, pages 121–123, 1963. 7

[15] Moshe Givoni and Piet Rietveld. The environmental implications of airlines' choice of aircraft size. *Journal of Air Transport Management*, 16(3):159–167, 2010. 66

[16] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008. 3, 4, 24, 32, 33, 39

[17] Michael Grüninger and Mark S Fox. Methodology for the design and evaluation of ontologies. 1995. 58

[18] Steven D Hales and Timothy A Johnson. Endurantism, perdurantism and special relativity. *The Philosophical Quarterly*, 53(213):524–539, 2003. 40

[19] Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015. 2

[20] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, and Sebastian Rudolph. Owl 2 web ontology language primer. *W3C recommendation*, 27(1):123, 2009. 24, 25

[21] Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. CRC Press, 2009. 44

[22] Ian Horrocks. Ontologies and databases. *Presentation at the Semantic Days*, 2008. 4

[23] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. 2006. xi, 3, 39, 46, 47, 50, 51

[24] Ian Horrocks and Ulrike Sattler. Decidability of shiq with complex role inclusion axioms. *Artificial Intelligence*, 160(1):79–104, 2004. 49

[25] Yevgeny Kazakov. Riq and sroiq shoiq. 2008. 50

[26] Markus Krötzsch. *OWL 2 Profiles: An introduction to lightweight ontology languages*. Springer, 2012. xi, 20, 29

[27] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. A description logic primer. *arXiv preprint arXiv:1201.4089*, 2012. 28

[28] Peter B Ladkin. Time representation: A taxonomy of internal relations. In *AAAI*, pages 360–366, 1986. 3

[29] Marcio Cardoso Machado, Ligia Maria Soto Urbina, and Michelle Aparecida Gomes Eller. Brazilian aeronautical maintenance: technical and geographical distribution. *Gestão & Produção*, 22(2):243–253, 2015. 2

[30] Deborah L McGuinness. *Explaining reasoning in description logics*. PhD thesis, Rutgers, The State University of New Jersey, 1996. 10, 15

[31] Deborah L McGuinness and Alexander Borgida. Explaining subsumption in description logics. In *IJCAI (1)*, pages 816–821. Citeseer, 1995. 15

[32] Viorel Milea, Flavius Frasincar, and Uzay Kaymak. towl: A temporal web ontology language. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(1):268–281, 2012. xi, 3, 33, 34, 35, 37, 38, 40, 43, 46, 47

[33] Jack Minker. On indefinite databases and the closed world assumption. In *6th Conference on Automated Deduction*, pages 292–308. Springer, 1982. 4

[34] Boris Motik and Ian Horrocks. Owl datatypes: Design and implementation. *The Semantic Web-ISWC 2008*, pages 307–322, 2008. 22, 24, 39, 43

[35] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. Owl 2 web ontology language: Structural specification and functional-style syntax. *W3C recommendation*, 27(65):159, 2009. xi, 44

[36] Natalya F Noy. Ontology development 101: A guide to creating your first ontology: Knowldege systems laboratory, stanford university. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, 2001. 54, 57, 59, 60

[37] David S Oderberg. The metaphysics of identity over time. 1993. 42

[38] Bijan Parsia and Uli Sattler. Owl 2 web ontology language data range extension: linear equations, 2009. 47

[39] Bart-Jan van Putten, SR Wolfe, and MV Dignum. An ontology for traffic flow management. 2008. 58

[40] Matthias Reiss, Marie Moal, Yvonne Barnard, Jean-Philippe Ramu, and Agnes Froger. Using ontologies to conceptualize the aeronautic domain. In *Proceedings of the International Conference on Human-Computer Interaction in Aeronautics. Cépaduès-Editions, Toulouse, France*, pages 56–63. Citeseer, 2006. 58

[41] Carlos Henrique Marques da Rocha, Robert Ramon de Carvalho Sousa, and Nilo de Souza Campos. Uma análise da situação financeira da indústria brasileira de aviação civil. *Journal of Transport Literature*, 10(3):35–39, 2016. 3

[42] Uli Sattler, Robert Stevens, and Phillip Lord. How does a reasoner work? *Ontogenesis*, 2014. 18

[43] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *OWLED*, volume 432, 2008. 28

[44] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007. 28

[45] John F Sowa. *Knowledge representation: logical, philosophical, and computational foundations*, volume 511. Brooks/Cole, 1994. 8

[46] John F Sowa. Knowledge representation: logical, philosophical, and computational foundations. 1999. 8

[47] Steffen Staab and Rudi Studer. *Handbook on ontologies*. Springer Science & Business Media, 2013. 2

[48] Kristin Thornburg. The perdurantist's commitments. *Res Cogitans*, 1(1):4, 2010. 40

[49] Michael Uschold and Martin King. *Towards a methodology for building ontologies*. Citeseer, 1995. 57

[50] Mike Uschold and Michael Gruninger. Ontologies: Principles, methods and applications. *The knowledge engineering review*, 11(02):93–136, 1996. 58

[51] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*, volume 1. Elsevier, 2008. xi, 16, 18

[52] Yuan Wang, Qing Li, Yong Sun, and Jinliang Chen. Aviation equipment fault information fusion based on ontology. In *2014 International Conference on Computer, Communications and Information Technology (CCIT 2014)*. Atlantis Press, 2014. 5

[53] Chris Welty, Richard Fikes, and Selene Makarios. A reusable ontology for fluents in owl. In *FOIS*, volume 150, pages 226–236, 2006. x, 40, 41