UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Isauflânia Suelen Ribeiro Timóteo

Uma ontologia de Código-Fonte Funcional construída com o método SABiOS

Vitória, ES

2023

Isauflânia Suelen Ribeiro Timóteo

Uma ontologia de Código-Fonte Funcional construída com o método SABiOS

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Ciência da Computação

Orientador: Camila Zache de Aguiar

Coorientador: Prof. Vítor E. Silva Souza

Vitória, ES

2023

Isauflânia Suelen Ribeiro Timóteo
Uma ontologia de Código-Fonte Funcional construída com o método SABiOS/ Isauflânia Suelen Ribeiro Timóteo. – Vitória, ES, 2023-
58 p. : il. (algumas color.) ; 30 cm.

Orientador: Camila Zache de Aguiar

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Colegiado do Curso de Ciência da Computação, 2023.

1. Ontologia. 2. Programação Funcional. 3. SABiOS I. Timóteo, Isauflânia Suelen Ribeiro. II. Universidade Federal do Espírito Santo. IV. Uma ontologia de Código-Fonte Funcional construída com o método SABiOS

CDU 02:141:005.7

Isauflânia Suelen Ribeiro Timóteo

Uma ontologia de Código-Fonte Funcional construída com o método SABiOS

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, 08 de Fevereiro de 2023:

Camila Zache de Aguiar
Orientador

Prof. Vitor E. Silva Souza
Coorientador

Veruska Carretta Zamborlini
Universidade Federal do Espírito Santo

Alexandre Adler Cunha de Freitas
Universidade Federal do Espírito Santo

Vitória, ES
2023

Agradecimentos

Agradeço a Deus por ter me proporcionado saúde e determinação para continuar me esforçando e alcançar meus objetivos.

Aos meus familiares pelo apoio incondicional durante toda a minha formação acadêmica, em especial meus pais Luiz Carlos e Maria Aparecida, a minha irmã Ellen e meu companheiro Vinícius.

Aos meus amigos da graduação e do trabalho que compartilharam comigo todos os momentos difíceis e também momentos felizes, me dando apoio e sempre prontos para me ajudar: Leandro, Brida, Athus, Franco, Ana Paula e Thayza.

Também agradeço a todos os docentes que participaram da minha graduação, ao meu coorientador Prof. Vitor pela confiança, pela paciência e por prontamente me ajudar sempre que o procurei. À minha orientadora, Camila, por ter me dado a oportunidade de trabalhar neste projeto, pela dedicação, paciência, pelo compartilhamento de seu conhecimento, tempo e amizade.

"Mais do que máquinas, necessitamos de humanidades.
(Charles Chaplin)

Resumo

Em conjunto com o constante avanço tecnológico, o mercado de desenvolvimento de software tem crescido proporcionalmente. Com base nas linguagens de programação "clássicas" têm sido desenvolvidas inúmeras linguagens de programação voltadas para aplicações de diversas áreas, desde aplicações científicas até comerciais. Devido a esse avanço progressivo, estudar uma linguagem de programação tem requerido uma maior dedicação, pois mesmo fazendo parte de uma mesma categoria, cada linguagem possui a sua particularidade. Assim, faz-se necessário que estejamos sempre atualizados e tenhamos um conhecimento mais especializado das linguagens de programação.

Uma vez que as ontologias têm sido utilizadas para representação clara de conceitos e relações nos mais variados domínios, neste trabalho, fazemos uso da ontologia para representar o domínio de código-fonte funcional, a fim de explorar a semântica dos conceitos desse domínio e oferecer aos usuários uma fonte de conhecimento clara e bem fundamentada.

Para isso, utilizamos a engenharia de ontologias como meio de possibilitar a construção de uma ontologia de qualidade, aplicando o método SABiOS *Systematic Approach for Building Ontologies, Supplemented*. Em conjunto, foi aplicada análise ontológica com a ontologia de fundamentação UFO para a construção da FUNC-O - *Functional Code Ontology*, que reflete o domínio de código-fonte de programação funcional. Durante a construção do modelo buscou-se aplicar o passo a passo proposto pelo método SABiOS para o desenvolvimento da ontologia de referência e sua validação, perpassando as fases de *conception*, *pre-reference* e *reference*.

Por fim, a ontologia foi verificada respondendo as questões de competência levantadas para o domínio e validada instanciando seus conceitos com situações do mundo real, através do código-fonte de cada linguagem de programação estudada.

Palavras-chaves: Ontologia, Engenharia de Ontologia, Método SABiOS, Linguagem de Programação, Programação Funcional e Interoperabilidade.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFO	Unified Foundational Ontology
FUNC-O	Functional Code Ontology
SCO	Source Code Ontology
SPO	Software Process Ontology
SwO	Software Ontology
OOC-O	Object-Oriented Code Ontology
SABiO	Systematic Approach for Building Ontologies
SABiOS	Systematic Approach for Building Ontologies, Supplemented
SCON	Source Code Ontology Network
FNO	Functional Ontology
MOCA	Modelagem de uma Ontologia para Construção de Algoritmos

Sumário

1 Introdução

Programar é uma atividade central no meio computacional realizada pelos profissionais da área. O papel fundamental e que possibilita esse desempenho é exercido pelas linguagens de programação (VAREJÃO, 2004).

Com o constante avanço tecnológico, têm sido desenvolvidas inumeras linguagens de programação voltadas para aplicações de diversas áreas, desde aplicações científicas até comerciais. Com essa variedade de linguagens, estudar uma linguagem de programação tem requerido uma maior dedicação, pois mesmo fazendo parte de uma mesma categoria, cada linguagem tem a sua particularidade.

Assim, através do estudo de linguagens de programação é possível usar meios computacionais para encontrar uma solução de problemas que podem surgir em situações específicas. Tais linguagens são classificadas de acordo com suas características, chamadas como paradigmas. Segundo Varejão (2004), em seu último nível, os paradigmas das linguagens são: estruturado, orientado a objetos, concorrente, funcional e lógico. O paradigma funcional é o tema do presente trabalho.

Neste contexto, o paradigma funcional está voltado diretamente para funções e soluções de problemas matemáticos, não possuindo efeitos colaterais e utilizando recursividade como técnica. Essas funções tratam problemas específicos, recebendo um ou mais dados como entrada e retornando esses dados como uma saída e possível solução. Esse paradigma é muito utilizado em aplicações matemáticas ou estatísticas e inteligência artificial. Visto que este paradigma obtém muitos conceitos e relações complexas que dificultam sua compreensão, este trabalho busca através do uso de ontologias solucionar este e outros problemas relacionados ao domínio de programação funcional.

Os modelos ontológicos foram explorados por muitos anos e seu estudo tem se expandido por diversas áreas. As ontologias têm sido utilizadas para organização de conceitos e relações para um melhor entendimento de um determinado domínio (SWARTOUT; TATE, 1999). Elas possibilitam o compartilhamento de informações de forma clara e objetiva, quando elaboradas a partir de uma Engenharia de Ontologias que permite uma construção mais eficaz e com qualidade.

Assim, o método de Engenharia de Ontologias SABiOS é utilizado neste trabalho para a construção da ontologia de programação funcional. O SABiOS é baseado no método SABiO (FALBO, 2014) e objetiva auxiliar a construção de ontologias, tanto de referência quanto operacional. O SABiOS busca facilitar o processo de desenvolvimento de ontologia, baseado em princípios ágeis (AGUIAR, 2021).

Desse modo, o presente trabalho teve como objetivo aplicar o método SABiOS para a construção de uma ontologia de código-fonte funcional bem fundamentada e que apresentasse uma conceituação consensual e compartilhada deste domínio.

1.1 Motivação e Justificativa

Nos dias atuais, a variedade de linguagens de programação tem se tornado cada vez mais ampla para atender a constante evolução do desenvolvimento de software. Por isso, é necessário que estejamos sempre atualizados e tenhamos um conhecimento mais especializado de linguagens de programação destinadas a determinados contextos. Existem ainda diferentes maneiras de representar um código fonte dependendo do tipo de linguagem utilizada ou de seu paradigma.

Por conta dos diferentes cenários de representação do código-fonte, podemos observar que existe um problema da interoperabilidade semântica em código-fonte, onde pode haver comprometimento da capacidade de interpretação correta de informações (FARINELLI; MELO; ALMEIDA, 2013). Desse modo, a Programação Funcional, assim como outras linguagens de programação que não estão fundamentadas numa representação semântica, apresentam informações não claras e muitas vezes inconsistentes. As informações são disponibilizadas por meio de textos, sites, livros, artigos, especificações e ainda não há uma representação unificada dessas informações, causando desafios para a interpretação de conceitos, uso de construtos e interoperabilidade semântica (AGUIAR, 2021).

Esses desafios podem ser mitigados através do uso de ontologia (FALBO, 2014). Atualmente, existe o desenvolvimento de diversos projetos ontológicos em diferentes áreas de estudo, que buscam disseminar o conhecimento, informações e conceitos sobre um domínio específico. A ontologia proposta neste trabalho, pretende tornar explícito o entendimento sobre programação funcional, disseminando o conhecimento sobre o domínio e auxiliando programadores em sua aplicação. A ontologia foi desenvolvida e validada utilizando o método SABiOS, de maneira que, com sua criação, pudesse ser aplicada para código fonte representado em linguagem de programação funcional.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver um modelo conceitual ontológico, por meio da aplicação do método SABiOS, refletindo o domínio de código-fonte funcional.

Os objetivos específicos do trabalho são:

1. Investigar o uso de ontologias e sua aplicação no domínio de programação funcional.

2. Definir uma conceituação consensual e compartilhada sobre o domínio de código-fonte funcional.
3. Criar uma ontologia que possibilite a compreensão do domínio de código-fonte funcional, de forma clara e unificada.
4. Analisar o método SABiOS, aplicado ao domínio de código-fonte funcional, como forma de validar o seu uso.

1.3 Método de Desenvolvimento do Trabalho

Para cumprir os objetivos estabelecidos na Seção 1.2, foi realizada uma pesquisa bibliográfica explorando os temas de: Linguagens de Programação, Programação Funcional, Modelagem Conceitual baseada em Ontologias e Engenharia de Ontologias. Após esta etapa, foram realizados estudos e discussões a respeito do método a ser utilizado para a construção da ontologia, utilizado o método SABiOS.

Assim, sendo realizado um estudo aprofundado do método SABiOS, aplicando os passos para a construção e validação, tanto da ontologia de referência, quanto do uso do método. Nesse sentido, o método foi aplicado seguindo as fases *Conception*, *Pre-reference* e *Reference*, sendo as fases *Pre-operational* e *Operational* a serem desenvolvidas em trabalhos futuros. Foi realizado um levantamento de todos os conceitos que faziam parte do domínio de cada linguagem selecionada, um catálogo para análise e comparativo dos conceitos e elaboração de uma definição consensual, conforme apresentado no capítulo 3. A ontologia elaborada foi verificada a partir de questões de competência e validada com dados reais de códigos representados em linguagem funcional.

1.4 Organização da Monografia

Além desta introdução, esta monografia é composta por outros cinco capítulos:

- O Capítulo 2 apresenta os aspectos relativos ao conteúdo teórico relevante para o trabalho;
- O Capítulo 3 apresenta como o método SABiOS foi aplicado para a construção da ontologia de código-fonte funcional;
- O Capítulo 4 apresenta a ontologia FUNC-O - *Functional Code Ontology*, ontologia resultado neste trabalho e suas análises;
- O Capítulo 5 apresenta a avaliação da ontologia FUNC-O em resposta às questões de competência, instanciação do código-fonte de diferentes linguagens de programação e mapeamento das diferentes linguagens para os conceitos da ontologia;

- e o Capítulo 6 apresenta as considerações finais do trabalho, assim como as possibilidades de trabalhos futuros.

2 Referencial Teórico e Tecnologias Utilizadas

Este capítulo traz um conjunto de referências que serviram como fundamento para uma melhor compreensão do presente trabalho. A Seção 2.1 apresenta uma breve introdução ao estudo de Linguagens de Programação, que a partir de algumas características levam à Programação Funcional apresentada na Seção 2.2, base para pesquisa e desenvolvimento do trabalho. A Seção 2.3 apresenta os principais conceitos que se fazem necessários para uma modelagem baseada em ontologias. Nela é definido o que é Ontologia e Engenharia de Ontologias, um pouco sobre seus principais métodos, aplicações e finalidade na Ciência da Computação. Por fim, a Seção 2.4 apresenta o método utilizado para a construção da ontologia de referência.

2.1 Linguagens de Programação

Uma Linguagem de Programação é um recurso que possibilita a interação do homem com o computador. É uma ferramenta usada pelos profissionais da área de Ciência da Computação para escrever programas, isto é, um conjunto de instruções que compõem um programa e são seguidas pelo computador para resolver um determinado problema ou realizar um processo (COUTINHO, 2020; VAREJÃO, 2004).

Existem diferentes níveis de linguagens de programação. Elas podem ser classificadas em: **linguagens de baixo nível** que são mais próximas da linguagem de máquina, conectando diretamente programas e hardware; e **linguagens de alto nível** que são mais intuitivas e de mais fácil entendimento por serem mais próximas da linguagem humana (COUTINHO, 2020).

Assim, as primeiras linguagens de programação modernas e os computadores digitais começaram a surgir a partir de 1940 e eram usados para aplicações científicas. Os primeiros computadores criados nesta década eram programados na linguagem Assembly, considerada uma linguagem de máquina ou de baixo nível (SEBESTA, 2011; BERTOLINI et al., 2019).

As linguagens de programação surgiram da lógica matemática e apesar das aplicações terem estruturas simples, requeriam diversas computações de aritmética de ponto flutuante (SEBESTA, 2011; BERTOLINI et al., 2019). Nos anos 50, começaram a ser desenvolvidos computadores para uso comercial e com eles as linguagens de programação de alto nível. A primeira linguagem de alto nível desenvolvida e voltada para os negócios foi o COBOL, que ainda hoje é muito utilizada nas aplicações empresariais e sistemas financeiros (SEBESTA, 2011).

Aplicações computacionais de Inteligência Artificial começaram a aparecer a partir de 1959 e a primeira linguagem de programação desenvolvida para aplicações de IA foi a linguagem funcional LISP (*list processing*) (MCCARTHY et al., 1965). Outras abordagens começaram a surgir a partir daí, como a programação lógica usando a linguagem Prolog (CLOCKSIN; MELLISH, 2003).

Nos anos 60 e 70 foram desenvolvidas diversas linguagens que são utilizadas até os dias de hoje, como por exemplo a linguagem C. A partir desse período foram estabelecidos os paradigmas de linguagens de programação, separando-as em categorias de linguagens (BERTOLINI et al., 2019).

Um paradigma pode ser definido como uma coleção de atributos que servem para categorizar um grupo de linguagens (VAREJÃO, 2004). As linguagens de programação, segundo alguns autores, podem receber quatro classificações: paradigma imperativo, orientado a objetos, lógico e funcional. A Figura 1, ilustra a classificação adotada por Varejão (2004), que aponta as classificações dos paradigmas subdivididos em duas categorias principais: imperativo e declarativo.

O paradigma imperativo abrange os paradigmas que descrevem alterações de estados na memória. Este paradigma é subdividido em: **estruturado** que organiza o processo de controle de execução dos programas, reduzindo-os em estruturas mais simples; **orientado a objetos** que baseia-se e descreve a composição de objetos. Neles são definidas as classes e os comportamentos relacionados a elas, que são chamados de métodos; e **concorrente** que descreve a concorrência de processos, que são executados de forma síncrona, por mais recursos.

O paradigma declarativo abrange os paradigmas que descrevem como a tarefa será resolvida de maneira abstrata. Esse paradigma é subdividido em: **funcional** que define funções que retornam como valor de saída uma solução de um problema; e **lógico** que é constituído por cláusulas que definem relações de entrada e saída, e relações baseadas em fatos e regras.

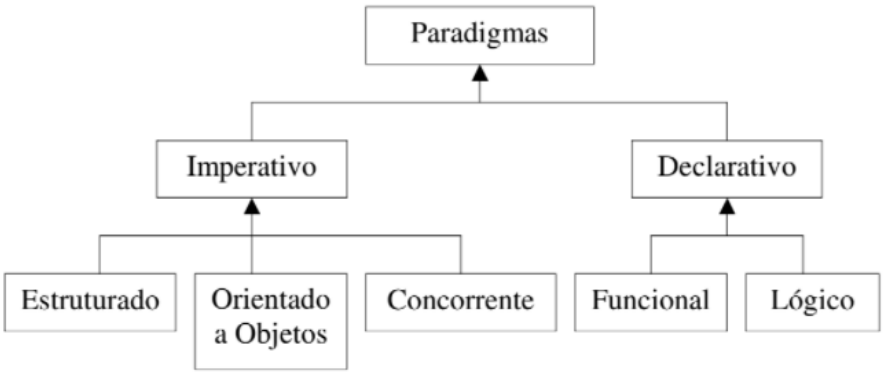

Figura 1 – Paradigmas de Linguagens de programação. Fonte: Varejão (2004).

Em virtude do avanço progressivo do desenvolvimento de linguagens de programação para áreas de aplicações distintas, algumas linguagens foram desenvolvidas com uma abordagem multiparadigma, que suportam mais de um paradigma de programação a fim de se adequar aos diversos tipos de problemas. Dessa forma, uma linguagem de programação que está sendo desenvolvida, conforme suas necessidades e características, pode se enquadrar em um ou mais paradigmas, e estes definem como ela irá operar e resolver um determinado problema. Como exemplo de uma linguagem de programação multiparadigma, poderia citar a linguagem Python, que se enquadra no paradigma orientado a objetos, funcional e procedural.

No presente trabalho, o enfoque deste estudo é o paradigma de programação funcional, que veremos a seguir.

2.2 Programação Funcional

A programação funcional é um paradigma que trata, em sua essência, funções e problemas matemáticos, não possuindo efeito colateral e utilizando recursividade. Essas funções são procedimentos que executam tarefas que, por meio de conjunto de dados de uma ou mais entradas, produzirão um conjunto de dados de saída, do ponto de vista matemático, mapeando membros de um conjunto domínio para um conjunto imagem (SCHMITT; SILVA et al., 2003; SEBESTA, 2011).

O cálculo lambda, de uma maneira formal, pode-se dizer que descreve o cálculo de uma função. Ele pode ser considerado como uma modesta linguagem de programação funcional, pois nele foi realizado o estudo de propriedades e funções estimáveis, podendo ser considerado um modelo computacional. Esse cálculo foi desenvolvido em 1930 por Alonzo Church. Dentro da Ciência da Computação, esse cálculo faz parte da área de estudo de teoria da computação, sendo a linguagem funcional muito utilizada no meio acadêmico (SCHMITT; SILVA et al., 2003; JUCA, 2018).

A primeira linguagem de programação funcional desenvolvida foi a LISP, por John McCarthy em 1959, para uso em aplicações de Inteligência Artificial. Embora não seja uma linguagem puramente funcional e que não tenha agregado conceitos funcionais atuais, a LISP em sua originalidade, representa bem os conceitos fundamentais da programação funcional de forma que muitas linguagens atuais herdaram suas características funcionais (SCHMITT; SILVA et al., 2003; SEBESTA, 2011). Existe uma grande variedade de linguagens de programação funcional ou que fazem uso do paradigma funcional, tal como Python, Haskell, R e Scheme.

Quando se fala em paradigma funcional, é intuitivo comparar ao paradigma imperativo. No caso do paradigma funcional, uma vez que essa é uma programação baseada nas funções matemáticas, a linguagem de programação é utilizada para lidar com abstrações e

sistemas complexos, evitando dados mutáveis ou compartilhamento de estados, ou seja, ela dita "o que" deve ser feito (NASCIMENTO, 2019; SCHMITT; SILVA et al., 2003). No paradigma imperativo, a linguagem dita "como" um processo ou execução deve ser feito. Nesse paradigma são especificadas as alterações de estados, descrevendo variáveis e ações que os manipulam. Essas variáveis são associadas a um valor que pode ser alterado através de operações atribuídas. Em uma linguagem imperativa, o resultado da análise de uma expressão é armazenado em uma variável, diferentemente de uma linguagem funcional que não usa variáveis e nem sentenças de atribuição (VAREJÃO, 2004; SEBESTA, 2011).

As principais características e vantagens da programação funcional vão além do tratamento de funções para uma resolução de um problema: **(i)** realiza uma análise das funções matemáticas, onde uma função pode ser manipulada de maneiras distintas; **(ii)** não possui efeitos colaterais, isto é, a ordem de execução é irrelevante para o seu resultado; **(iii)** não possui comandos de atribuição; **(iv)** utiliza a recursividade como técnica, sendo esta considerada melhor que os laços utilizados pela imperativa; e **(v)** possui gerenciamento de memória automática (SILVEIRA et al., 2021; SCHMITT; SILVA et al., 2003).

Programar em uma linguagem de programação funcional compreende um conjunto de diferentes conceitos e características, abrangendo a construção de definições e avaliação de expressões (SCHMITT; SILVA et al., 2003). Dentre as diversas abordagens existentes, ainda há a necessidade de buscar técnicas de aprendizado em diferentes artefatos, para se familiarizar com conceitos como funções de ordem superior, recursividade, funções lambda, existindo a possibilidade de uma compreensão distinta de notações e conceitos básicos. A seguir, na Seção 2.3 a seguir, veremos como representar essas características em um modelo conceitual ontológico de forma a definir e unificar as conceituações do domínio de programação funcional.

2.3 Ontologias

As ontologias permitem a representação formal de conceitos fundamentais interpretados pelos usuários, de forma que facilite a comunicação e compreensão de um determinado domínio, podendo ter diversos significados quando utilizadas em áreas de estudo distintas (ISOTANI; BITTENCOURT, 2015; MACIEL, 2019).

O termo ontologia tem sua origem na filosofia, mais precisamente na disciplina de Metafísica (ISOTANI; BITTENCOURT, 2015). Em seu contexto, a ontologia tenta responder questões relacionadas à existência, o estudo do ser, assim como descrita por Aristóteles: "a ciência do ser enquanto ser". O termo em sua definição, existe desde o século IV a.C, contudo, só foi especificado na filosofia no século XVII (ZAMBORLINI, 2011), onde a palavra "ontologia" aparece pela primeira vez no livro "Ogdoas Scholastica" de Jacob Lorhard publicado em 1606 (FRANÇA, 2009).

Apesar de ter sua origem na Filosofia e ao longo dos anos o termo ser utilizado pelos filósofos para se referir a elementos naturais do mundo, ainda não há um consenso sobre a semântica da palavra “ontologia”, principalmente na Ciência da Computação (GUIZZARDI, 2000), pois a expressão "ontologia"tem sido aplicada em diversas áreas de conhecimento e pesquisa, onde adquiriu diferentes significados. Dentre as diversas definições, a que mais é citada e de mais fácil entendimento é a de Thomas R. Gruber: “uma ontologia é uma especificação explícita de uma conceituação”, onde é entendido que conceituação é uma visão do mundo abstrata e simplificada, que desejamos representar para algum propósito (GRUBER, 1993). Essa definição foi adaptada por Studer, Benjamins e Fensel (1998) que a completa: “uma ontologia é uma especificação explícita e formal de uma conceitualização compartilhada”.

Em Ciência da Computação o termo foi empregado pela primeira vez em 1967, por G.H. Mealy, na área de processamento de dados. Com o decorrer dos anos, na área de Inteligência Artificial, houve a necessidade da representação e compartilhamento de um domínio específico da área, fazendo com que ocorresse uma demanda maior no estudo do assunto (GUIZZARDI, 2005). Desde então a ontologia tomou grande destaque nas áreas computacionais de Inteligência Artificial, Engenharia de Software, Banco de Dados e Web Semântica, sendo que esta última mostrou-se extremamente dependente de ontologias em seu desenvolvimento de aplicações e implementação de soluções. A partir daí vemos o quão crucial as ontologias se tornaram nos sistemas de software, que de alguma maneira integram informações.

Uma ontologia é composta por **conceitos**, que podem representar qualquer coisa em um domínio específico; **relações** que representam as interações existentes entre os conceitos e cardinalidades no domínio; **funções** que são um retrato especial de relações; **axiomas** que são sentenças, que sempre são verdadeiras em qualquer situação; e as **instâncias** que são utilizadas para representar os elementos que compõem o domínio (GRUBER, 1993; PASINI, 2009).

2.3.1 Classificação de Ontologias

Na bibliografia é possível encontrar diferentes classificações quanto à aplicação e estrutura das ontologias, uma bastante conhecida foi apresentada por Guarino (1998), mostrada na Figura 2. As ontologias são classificadas em quatro níveis de generalidade, que são: 1. Ontologias de Fundamentação ou Topo (Nível superior) que descrevem conceitos mais gerais e independem de um problema ou domínio específico; 2. Ontologias de domínio que descrevem um tipo próprio e específico de ontologia, para representar conceitos e relações existentes em uma determinada área; 3. Ontologias de tarefas especificam os termos introduzidos na ontologia, descrevem atividades ou tarefas; e 4. Ontologias de aplicação que descrevem os conceitos definidos conforme o domínio e tarefa especificados.

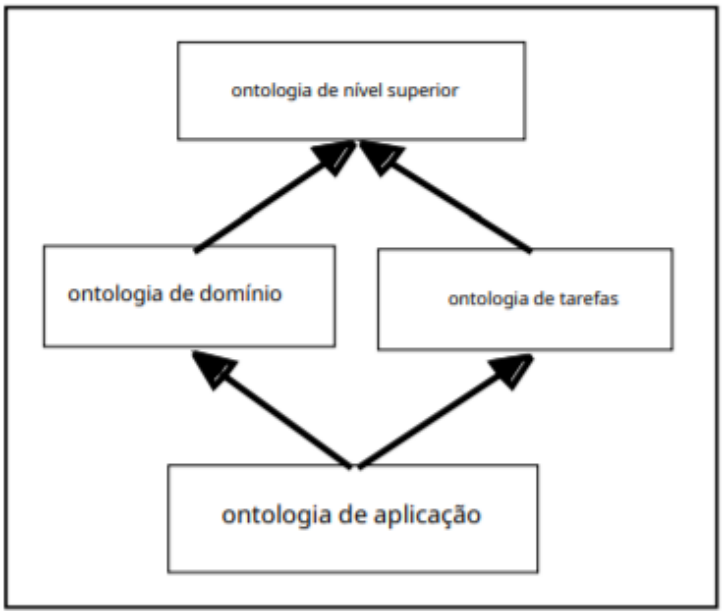

Figura 2 – Tipos de ontologias, de acordo com seu nível de dependência. Fonte: Guarino (1998)

.

Esses conceitos das ontologias de domínio e tarefas são termos introduzidos na ontologia de nível superior, sendo os termos da ontologia de aplicação um modo de especializar ambas as ontologias de domínio e tarefas relacionadas (GUARINO, 1998). Uma classificação proposta por Falbo et al. (2013), define três tipos de ontologias: 1. **Ontologias de Fundamentação**, que descrevem conceitos que independem de um domínio específico; 2. **Ontologias de núcleo**, essas descrevem uma definição da estrutura da ontologia em uma área específica, cobrindo diversos domínios de uma aplicação; e 3. **Ontologias de domínio**, que especificam os termos de um determinado domínio.

Falbo (2014) ainda propõe as seguintes linhas quanto à classificação de ontologias e seus aspectos: **Ontologia de Referência** que é desenvolvida para descrever o domínio de maneira clara e objetiva, para uma melhor compreensão e resolução de problemas; e **Ontologia Operacional** que é desenvolvida para alcançar os objetivos satisfatórios quanto ao seu processamento e propriedades computacionais.

Assim, como Guarino (1998) e outros trabalhos mencionam, o desenvolvimento de uma ontologia de domínio deve ser sustentada por ontologias de fundamentação (nível superior), trazendo uma base melhor para modelagem e representação do universo estudado.

Desse modo, para o desenvolvimento deste projeto, são relevantes as ontologias de fundamentação. Nele será usada a UFO (Unified Foundational Ontology), uma ontologia de fundamentação que busca melhorar e dar suporte a construção de um modelo conceitual, ou seja, no desenvolvimento de ontologias de domínio e aplicações baseadas nestas ontologias. Ela é firmada em teorias ontológicas formais, lógica filosófica e ciência cognitiva, abordando resultados de outras ontologias de fundamentação (GUIZZARDI, 2005). Apesar de essas

ontologias que ela sumariza compreenderem propriedades significativas, elas têm algumas limitações notáveis no processo de captura de conceitos básicos na modelagem conceitual. Assim, a UFO aborda aspectos primordiais para a modelagem conceitual, propondo a unificação dessas ontologias (ZAMBORLINI, 2011).

Ela é subdividida em três partes principais, que são chamadas de fragmentos: UFO-A (uma ontologia focada em objetos) define o núcleo da ontologia, ela é composta de categorias de elementos e representa tipos de indivíduos chamadas de *endurants*; UFO-B (uma ontologia focada em eventos) define relações temporais e conceitos que acontecem como estados, eventos, processos, entre outros; e UFO-C (uma ontologia de âmbitos sociais, construída com base a partir de A e B) define conceitos como compromissos sociais, ação, estados internacionais, entre outros (GUIZZARDI, 2005; GUIZZARDI et al., 2011; BARCELLOS, 2009).

2.3.2 OntoUML

A abordagem selecionada para ser utilizada no presente trabalho foi a UFO e entre as suas principais partes, o fragmento da UFO-A, ontologia focada em objetos. A UFO-A pode ser representada em OntoUML *(Ontological Unified Modeling Language)*, é uma linguagem ontologicamente bem fundamentada para Modelagem Conceitual sendo construída como uma extensão da UML (GUIZZARDI et al., 2015). A OntoUML evidência distinções ontológicas fundamentais de UFO-A, diretamente na sintaxe, por meio de estereótipos aplicados a classes e relações de um diagrama de classes. Um dos benefícios em sua aplicação é que ela provê uma semântica formal e restrita (GUARINO, 1998), sendo adotada por diversas instituições como forma de modelagem conceitual em vários domínios e em seu desenvolvimento.

A OntoUML realiza uma distinção fundamental entre **tipos**, uma abstração que geramos para nos auxiliar a classificar o mundo real, e **indivíduos** que são particularidades identificadas através das características encontradas neste mundo. Ela trabalha com *princípios de identidade* impondo restrições para que possamos entender como serão combinados os conceitos para projeção do modelo conceitual (SUCHANEK, 2018). A Tabela 1 mostra algumas dessas esteriotipações.

Tabela 1 – Estereótipos de OntoUML.

Esteriótipo	**Definição**
«kind»	Representa conceitos rígidos, sendo um todo que cujas partes (complexo funcional) contribuem de formas distintas para outras funcionalidades.
«subkind»	Representa uma especialização sortal rígido cujas instâncias herdam o princípio de identidade por meio de um kind.
«category»	Representa um mixin rígido que não requer a especificação de uma dependência, pode representar uma característica comum entre os indivíduos.
«quality»	Representa conceitos rígidos que dão um princípio de identidade a suas instâncias, geralmente refere-se a matérias incontáveis.
«role»	Representa um conceito anti-rígido, refere-se a instâncias que possuem um determinado papel.
«relator»	Representa um conceito rígido que realiza a conexão de outros indivíduos para que possa existir.

2.3.3 Engenharia de Ontologias

A Engenharia de Ontologias permite o desenvolvimento de uma ontologia com maior qualidade. Ela estuda o processo de construção de uma ontologia e busca sanar possíveis restrições e problemas que possam surgir nesse caminho, oferecendo métodos e ferramentas para solucioná-los. Possibilita também o compartilhamento de uma base de conhecimentos construída para outras ontologias em criação (ISOTANI; BITTENCOURT, 2015).

A construção de uma ontologia envolve diversas atividades, como identificar seu propósito, conceitos e realizar a modelagem do domínio. Um dos desafios durante esse processo de construção é fazer com que exista uma proximidade da ontologia desenvolvida com a realidade (ISOTANI; BITTENCOURT, 2015).

Na literatura é possível encontrar diversos métodos para o desenvolvimento de ontologias, como a proposta apresentada por Noy, McGuinness et al. (2001) em Ontology Development 101, que apresenta sete passos para a construção de uma ontologia, envolvendo atividades desde a sua análise conceitual até a fase de instanciação. Ontology Development 101 é um método simples e bastante utilizado.

A Methontology, proposta por Fernández-López, Gómez-Pérez e Juristo (1997), assim como a Ontology Development 101, é inspirada em Engenharia de Software. Foi desenvolvida em um laboratório de Inteligência Artificial e utiliza o método para construção de outras ontologias por reengenharia. Suas atividades principais são: especificação, conceitualização, formalização, implementação e manutenção (ISOTANI; BITTENCOURT,

2015; PASINI, 2009). A On-to-knowledge proposta por Fensel et al. (2000) foi desenvolvida para dar suporte em contextos organizacionais, na administração de sistemas por meio de provedores sustentando documentos eletrônicos. O método SABiO proposto por Falbo (2014), foi desenvolvido para a construção de ontologias de referência e operacionais que possam ser integradas a UFO, sendo composto por cinco fases principais, que são: identificação de propósito, definição dos requisitos, captura e formalização da ontologia, design, implementação e testes.

Os métodos consideram atividades presentes no ciclo de vida de uma ontologia, onde este ciclo estabelece cada estágio que será seguido ao longo do desenvolvimento desta ontologia. Cada método tem a sua forma de conduzir essas atividades, como algumas mencionadas anteriormente. Desse modo, o ciclo de vida de uma ontologia pode variar de acordo com o método que está sendo utilizado para sua construção (ISOTANI; BITTENCOURT, 2015).

A Figura 3 mostra uma generalização do ciclo de vida de uma ontologia, onde cada etapa é descrita como: **especificação** que estabelece o motivo da construção da ontologia, onde será usada e quais os usuários interessados; **conceitualização** que organiza e estrutura todo o conhecimento acerca do domínio escolhido para a construção da ontologia; **formalização** estabelece todos os conceitos e relações da ontologia construindo o modelo conceitual; **implementação** elabora a ontologia em uma determinada linguagem; e **manutenção** que repara a ontologia que já passou pela etapa de implementação (ISOTANI; BITTENCOURT, 2015).

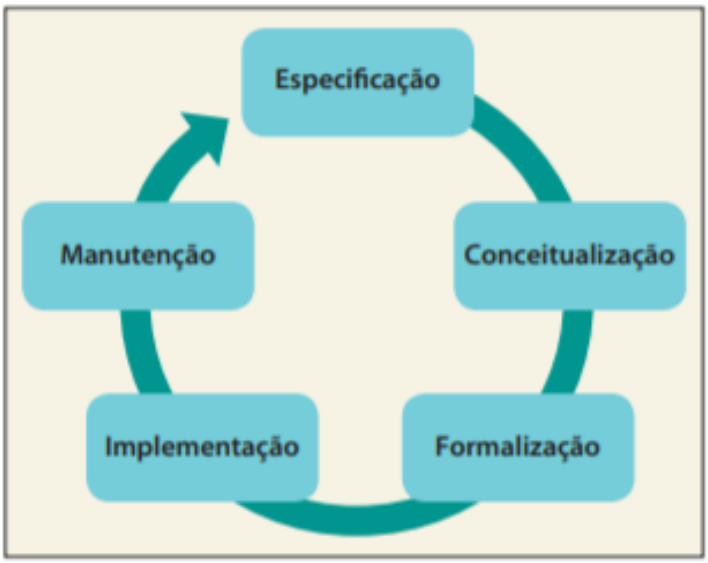

Figura 3 – Ciclo de vida de uma ontologia. Fonte: Isotani e Bittencourt (2015)

.

Mesmo sem determinar qual método será utilizado, todas apresentam um ciclo de vida que serve como guia na construção da ontologia (ISOTANI; BITTENCOURT, 2015).

Nesse contexto, quanto aos aspectos de Engenharia de Ontologias, veremos na próxima seção, o método SABiOS.

2.4 Método SABiOS

Esta seção apresenta o método escolhido para o processo de desenvolvimento do modelo conceitual baseado em ontologia, o método SABiOS – *Systematic Approach for Building Ontologies, Scrum* (AGUIAR, 2021). SABiOS é um método proposto para a construção de ontologias, que cobre tanto o desenvolvimento de ontologias de referência quanto o operacional, independente de seu domínio. Tem como base o método SABiO *(Systematic Approach for Building Ontologies)* proposto por Falbo (2014), apresentado na Seção 2.3.3.

SABiOS foi definido a partir da análise dos resultados obtidos pela aplicação do método SABiO, observando uma brecha quanto ao nível de detalhes de sua atividades e processos sugeridos. Dessa forma, o SABiOS tem a intenção de ser um método de fácil utilização e guiar mais detalhadamente o processo de desenvolvimento da ontologia, baseado em princípios ágeis (AGUIAR, 2021).

Em SABiOS são definidos o ciclo de vida de desenvolvimento das ontologias, os comportamentos, artefatos e atividades. A abordagem adotada pelo SABiOS utiliza como direcionamento questões de competência e indiretamente orientação de desenvolvimento por usuário (especialista e engenheiros) e por dados (fontes de dados consistentes).

O ciclo de vida do método SABiOS é subdividido em dois ciclos: ciclo de vida da ontologia e ciclo de vida da fase, que estão relacionadas, uma apresentando as fases e a outra as atividades dessas fases. O **ciclo de vida da ontologia** caracteriza o processo de construção da ontologia por meio de fases e iterações. A Figura 4 apresenta a composição desse ciclo que é constituído por cinco fases: **conception** que destaca o propósito da ontologia acordado entre as partes de interesse; **pre reference** que destaca as questões quanto a modelagem, que influenciarão o desenvolvimento da ontologia de referência; **reference** que destaca a construção da ontologia de referência através do conhecimento adquirido na fase **conception**, sendo considerada a fase mais crítica; **pre operational** que destaca questões quanto a codificação, que influenciarão o desenvolvimento da ontologia operacional; e **operational** que destaca a construção da ontologia em uma linguagem operacional.

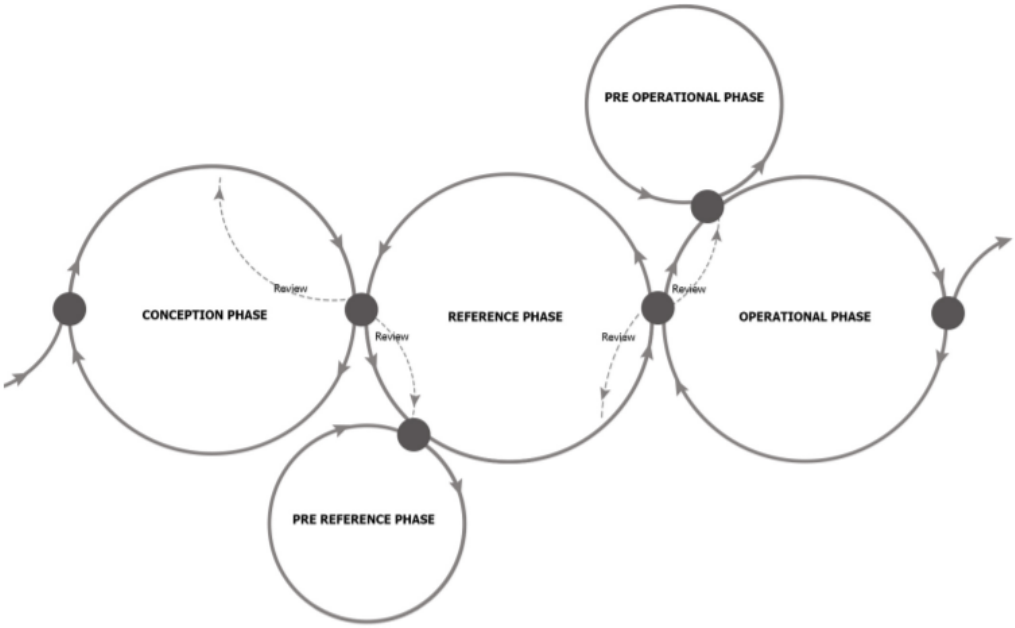

Figura 4 – Ciclo de vida da ontologia. Fonte: (AGUIAR, 2021).

.

O **ciclo de vida da fase** caracteriza o processo de construção da ontologia através das atividades, processos e iterações que integram o ciclo de vida da ontologia. Esse ciclo é constituído pelas atividades que integram dois tipos de processos. A Figura 5 apresenta uma visão geral do método, com destaque aos processos e suas atividades, que são: **processos principais** que descrevem os procedimentos essenciais para o desenvolvimento da ontologia, a saber: os processos de requisitos (em amarelo), captura (azul e verde), design (rosa) e implementação (roxa); e os **processos de suporte** que descrevem os procedimentos que apoiam os processos principais e dentre eles estão: o processo de aquisição de conhecimento (círculo pontilhado rosa), documentação, gerenciamento de configuração, avaliação, reuso (círculo pontilhado azul) e publicação.

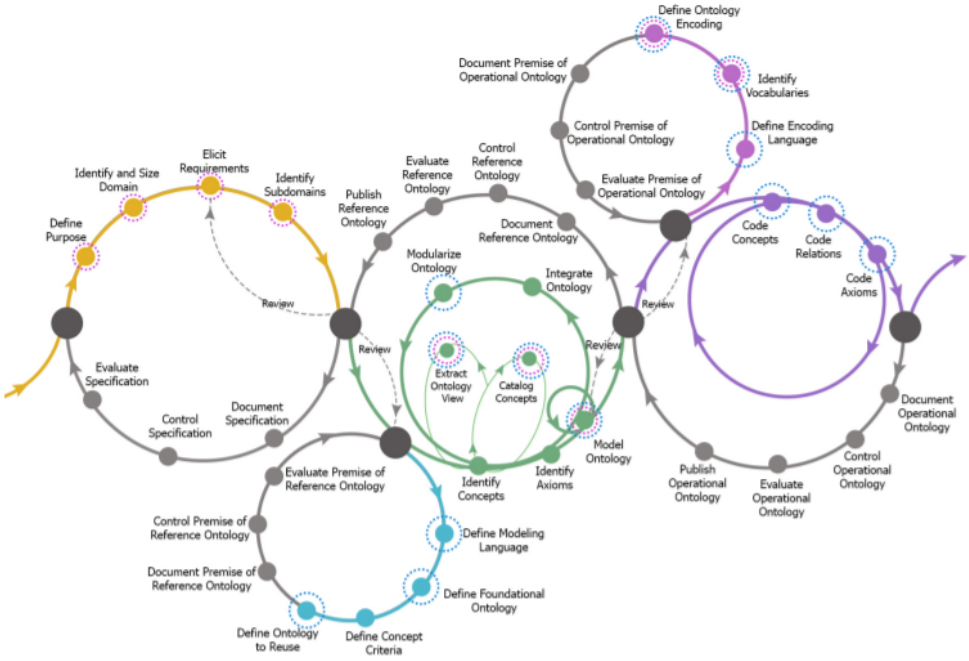

Figura 5 – Visão geral do método destacando os processos e suas atividades. Fonte: (AGUIAR, 2021).

.

Desse modo, esse trabalho objetiva aplicar o método SABiOS para realizar as atividades dos processos principais e de suporte para a construção da ontologia, discutida mais detalhadamente no próximo capítulo para a construção da ontologia de código-fonte.

3 Aplicação do Método SABiOS

Este capítulo objetiva apresentar resultados da aplicação do método SABiOS para a construção da ontologia de código funcional, denominada FUNC-O - *Functional Code Ontology*. Nesse sentido, o método foi aplicado seguindo as fases *Conception*, *Pre-reference* e *Reference*, sendo as fases *Pre-operational* e *Operational* a serem desenvolvidas em trabalhos futuros. Embora o processo seja iterativo e cada fase seja executada diversas vezes até o resultado final, apresentamos a seguir os resultados das atividades de cada fase, cujos artefatos estão publicados no repositório compartilhado do projeto <https://drive.google.com/drive/folders/1RH0qoJuB34lrjZ_lmbyFTBcYDyjy76Rc?usp=share_link>

3.1 Fase 1 - Conception

Nesta atividade, o método é aplicado para a definição das questões gerais relacionadas à ontologia e partes interessadas. Embora o processo seja iterativo e cada fase seja executada diversas vezes até o resultado final, a seguir apresentamos os resultados das atividades de cada fase.

- **[REQI-PURP] Define Purpose**

  Esta atividade tem como objetivo principal descrever o propósito da ontologia, respondendo: (i) **como** a ontologia pretende representar o domínio, (ii) **para que** a ontologia será útil e (iii) **por que** a ontologia deve ser construída.

  Assim, o presente trabalho adota como propósito da ontologia: **Como** apresentar características identificadas a partir de estudos sobre o paradigma de programação funcional. **Para** disseminar o conhecimento acerca de Programação Funcional de forma clara e unificada, evitando diferentes entendimentos sobre um mesmo conceito. **Porque** as linguagens de programação definem suas próprias maneiras de declarar e descrever semanticamente as funções, o resulta em códigos heterogêneos e dificulta na interoperabilidade de seus conceitos.

- **[REQI-DOMN] Identify and Size Domain**

  A finalidade desta atividade é identificar o domínio que a ontologia irá representar, de acordo com o propósito estabelecido na atividade anterior e dimensioná-lo.

  O presente trabalho identificou o domínio de Programação Funcional, um paradigma que trata, principalmente, funções e problemas matemáticos. Essas funções são procedimentos que executam tarefas que, por meio de conjunto de informações de uma ou mais entradas, irão produzir uma saída, do ponto de vista matemático, mapeando

membros de um conjunto domínio para um conjunto imagem. No paradigma funcional essas funções são utilizadas para solucionar problemas, que são descrições gerais de situações específicas, que podem ter diversas soluções ou nenhuma. Uma vez que essa é uma linguagem baseada nas funções matemáticas, ela é utilizada para lidar com abstrações e sistemas complexos, evitando dados mutáveis ou compartilhamento de estados.

O paradigma funcional é bastante utilizado em aplicações matemáticas ou estatísticas e de Inteligência Artificial, em linguagens como Python, também em outras linguagens multiparadigmas como R, Scheme, entre outras. Dentre as suas principais vantagens estão: soluções e notação concisas, gerenciamento de memória automática, semântica clara e objetiva, facilidade na realização de testes e busca de problemas. Para a dimensão do domínio, observa-se que muitas das linguagens de programação existentes não são puramente funcionais.

Algumas das linguagens que foram projetadas para essa abordagem de desenvolvimento funcional são multiparadigmas, ou seja, suportam diversos paradigmas de programação. No entanto, esta ontologia limita-se à representação dos conceitos dos paradigmas funcionais e não da orientação a objetos, como métodos e classes ou procedurais, como goto e jump.

- **[REQI-ELIC] Elicit Requirements**

  Esta atividade objetiva elicitar requisitos funcionais e não funcionais, realizando esse passo por meio do conhecimento adquirido a respeito do domínio segundo as atividades anteriores. Para os requisitos funcionais foram elicitadas as seguintes questões de competência:
  Subdomínio: Função
  RF01 – Quais os principais conceitos de um código-fonte funcional?
  RF02 – Quais os tipos de funções existentes em um código-fonte funcional?
  RF03 – Quais os principais componentes de uma função?

  Subdomínio: Variável
  RF04 – Quais variáveis compõem um código-fonte funcional?
  RF05 – Quais variáveis compõem uma função?

  Para os requisitos não funcionais:
  RNF01 – Ser fundamentada por uma ontologia de fundamentação para reutilizar o que já é consolidado na literatura.
  RNF02 – Ser definida a partir de uma fonte de conhecimento base.
  RNF03 – Estar em um padrão que facilite a interoperabilidade dos conceitos com outras ontologias existentes.

- **[REQI-SUBD] Identify Subdomains**

Esta atividade objetiva identificar os subdomínios por meio das questões de competências definidas na atividade anterior, facilitando o desenvolvimento da ontologia. Assim, uma vez que a programação funcional se utiliza da composição de funções para a construção do código-fonte, para a ontologia FUNC-O foram definidos os subdomínios "Função"para representar os conceitos relacionados a funções e "Variável"para representar os conceitos relacionados a variáveis. Tais subdomínios auxiliam na construção da ontologia restringindo o que é mais relevante no domínio de programação funcional conforme as questões de competência.

- **[DOCM-SPEC] Document Specification**

  Esta atividade utiliza o Documento de Especificação de Ontologia como meio para documentar os passos desenvolvidos nas atividades da Fase 1. O documento de Especificação de Ontologia produzido na fase de *Conception* para a ontologia FUNC-O é apresentado no repositório do projeto compartilhado DOCM-SPEC-FUNC-O

- **[CONF-SPEC] Control Specification**

  Atividade que controla as versões e alterações registradas no Documento de Especificação, disponível no repositório do projeto. O controle do documento foi realizado através do cabeçalho do documento e o armazenamento das versões no repositório compartilhado do projeto.

- **[EVAL-SPEC] Evaluate Specification**

  Atividade que avalia se as informações registradas no Documento de Especificação atendem as expectativas esperadas. Por fim, o documento foi validado, seguindo para a fase *Pre Reference*.

3.2 Fase 2 - Pre-Reference

Nesta atividade o método destaca as questões quanto a modelagem, que influenciarão o desenvolvimento da ontologia de referência.

- **[CAPT-LANG] Define Modeling Language**

  Esta atividade define a linguagem de representação que será utilizada para o desenvolvimento da ontologia de referência, definida neste trabalho como a OntoUML, já abordada na Seção 2.3.2.

- **[CAPT-FOUN] Define Foundational Ontology**

  Esta atividade objetiva especificar a ontologia de fundamentação escolhida para orientar no processo de modelagem da ontologia, definida neste trabalho como *Unified Foundational Ontology* - UFO, apresentada na Seção 2.3. UFO é uma ontologia de

fundamentação firmada em teorias ontológicas formais, lógica filosófica e ciência cognitiva que busca dar suporte a construção de um modelo conceitual. Para a construção da ontologia FUNC-O, será adotada a ontologia de fundamentação UFO-A, focada em objetos.

- **[CAPT-CRIT] Define Concept Criteria**

  Esta atividade especifica os critérios para identificar se um conceito atende ao domínio da ontologia em desenvolvimento.

  Para a ontologia FUNC-O foram definidos os seguintes critérios:
  CR01: Conceito presente em mais de 50% das linguagens de programação analisadas.
  CR02: Conceito relacionado aos principais conceitos da programação funcional (função e variável).

- **[CAPT-REUS] Define Ontology to Reuse**

  Esta atividade define as ontologias de referência que poderão ser reusadas. Para o domínio de Código Funcional foram analisadas as ontologias apresentadas na Tabela 2:

Tabela 2 – Ontologias analisadas para Reuso.

Prefixo	**Definição**	**Análise**
FNO	Ontologia de função que descreve as funções independente de suas relações com conceitos e implementação (MEESTER, 2021).	Representa resumidamente a estrutura das funções, seus parâmetros tanto de entrada quanto de saída.
M.O.C.A	Ontologia elaborada a partir de conteúdos da disciplina de Algoritmos em linguagem de programação C (TERRA et al., 2017).	Apresenta informações de consultas e o contexto de desenvolvimento destes algoritmos.
SCO	Ontologia de código fonte fundamentada em UFO e elaborada a partir de diversas linguagens de programação (AGUIAR; ZANETTI; SOUZA, 2021).	Apresenta conceitos base como função e variável que podem ancorar os conceitos do paradigma funcional.

As ontologias FNO e M.O.C.A não foram utilizadas, pois a cobertura apontada não está dentro do dimensionamento da ontologia FUNC-O. Apesar de algumas ontologias estabelecerem conceitos relacionados a programação, na ontologia FNO foi discutido problemas relacionados a algoritmos, implementações e execuções para os processos em seu desenvolvimento, na ontologia M.O.C.A apresentou-se a construção e desenvolvimento de algoritmos na linguagem, visando abordar informações utilizadas para sua implementação, com uma perspectiva de execução, não servindo de apoio para a construção do domínio deste trabalho. Por fim, a ontologia SCO foi utilizada

para apoiar o desenvolvimento da ontologia FUNC-O, uma vez que a ontologia foi desenvolvida adotando a mesma ontologia de fundamentação e apresenta os conceitos base de um código-fonte que ancoram os conceitos de código-fonte funcional.

- **[DOCM-REFE] Document Premise of Reference Ontology**

  Esta atividade utiliza o Documento de Ontologia de Referência como meio para documentar as premissas desenvolvidas nas atividades. O documento de Ontologia de Referência produzido na fase de Pre Reference para a ontologia FUNC-O é apresentado no repositório do projeto compartilhado DOCM-SPEC-FUNC-O.

- **[CONF-REFE] Control Premise of Reference Ontology**

  Atividade que controla as versões e alterações registradas no Documento de Referência. O controle do documento foi realizado através do cabeçalho do documento e o armazenamento das versões no repositório compartilhado do projeto.

- **[SAVA01] Evaluate Premise of Reference Ontology**

  Atividade que avalia se as informações registradas no Documento de Referência atendem as expectativas esperadas. Por fim, o documento foi validado seguindo para a fase *Reference*.

3.3 Fase 3 - Reference

Nesta atividade o método representa a conceituação do domínio com base na especificação elaborada na fase *Conception* e nas questões definidas na fase *Pre-Reference*.

- **[CAPT-CONC] Identify Concepts**
  Atividade que define os conceitos mais importantes do domínio e que compõem o propósito da ontologia de acordo com as fontes de conhecimento, neste trabalho, fontes bibliográficas. A seguir são apresentadas as sub atividades desenvolvidas para a identificação dos conceitos:

- **[CAPT-CATA] Catalog Concepts**
  Atividade que lista e cataloga os conceitos conforme os critérios definidos na atividade [CAPT-CRIT], verificando se atende ao domínio da ontologia em desenvolvimento e seus relacionamentos.

  Foram utilizados como fonte de conhecimento: livros, documentos de especificação, artigos e sites das linguagens de programação que suportam programação funcional. Durante o processo de aquisição de conhecimento, foram identificadas diversas linguagens relacionadas ao domínio, contudo a investigação por fontes de informação foi direcionada às linguagens: Python, R, Lisp e Haskell, pois essas linguagens são as

mais utilizadas no mercado e meio acadêmico, possuem atualizações constantes e são as que possuem maior banco de informações.

As descrições dos conceitos levantados seguiram as informações definidas nas fontes de conhecimento, conforme a atividade [REUS-DATA] Reuse Data Source. As tabelas 3, 4, 5 e 6 apresentam o catálogo de conceitos elaborado para cada uma das linguagens de programação estudadas no domínio de código funcional.

Tabela 3 – Catálogo de conceitos da linguagem Python

Conceito	**Definição**	**Instância**
Função	Uma função define um nome, com ou sem argumentos e um corpo. Executa uma série de instruções e pode ou não retornar um valor. (REFERENCE, 2020), (MEESTER, 2021)	def NOME(PARAMETROS): COMANDOS
Variável	Variável é um nome que se refere a um valor. Quando ela assume o valor de um objeto dizemos que ela está instanciada, quando ainda não assumiu o valor de nenhum objeto ela está não instanciada. (ALURA, 2020), (IME-USP, 2016)	NOMEVARIAVEL = VALOR
Função Lambda	A função lambda é uma função que não precisa ser definida, ela é uma função de uma linha, que funciona como se houvesse o retorno antes do comando, ou seja, pode ser utilizada dentro do código como se estivesse sido declarada como variável. (REFERENCE, 2020), (VAZ,)	def <lambda>(parameters): return expression
Função Matemática	Função matemática que realiza operações matemáticas simples e complexas. Como essa função pertence a um módulo, para utilizá-la é necessário realizar a importação. (DOWNEY, 2016)	import math math.pow(x, y)

Catálogo de conceitos da linguagem Python

Conceito	**Definição**	**Instância**
Retorno	É o resultado da execução de uma função. (DOWNEY, 2016)	return
Parâmetro	Parâmetro é um nome utilizado dentro de uma função para se referir ao valor passado como argumento. (DOWNEY, 2016), (REFERENCE, 2020)	def func(foo, bar=None)
Argumento	Argumento é um valor concedido a uma função quando a função é chamada. Este valor é atribuído ao parâmetro correspondente na função. (DOWNEY, 2016), (REFERENCE, 2020)	complex(real=3, imag=5)
Corpo	A sequência de instruções dentro de uma definição de função. (DOWNEY, 2016)	def func (...)(corpo da função)
Condição	Uma exigência/condição que deve ser satisfeita por quem chama a função, antes de executá-la. (DOWNEY, 2016)	def calculaarea(raio): if raio > 5: print("A área é: ",3.14 * raio**2) calculaarea(2) A precondição é o valor do raio passado para a função ser maior que 5.
Variável Local	Variável que é criada dentro de uma função, ou seja, ela só existe dentro da função. (DOWNEY, 2016)	def cattwice(part1, part2): cat = part1 + part2 print twice(cat) cat é a variável local
Variável Global	As variáveis globais são variáveis que uma vez definidas podem ser acessadas de qualquer lugar do código. (DOWNEY, 2016)	verbose = True def example1(): if verbose: print('Running example1') verbose é a variável global

Catálogo de conceitos da linguagem Python

Tabela 4 – Catálogo de conceitos da linguagem R

Conceito	**Definição**	**Instância**
Função	Uma função pode ou não conter uma palavra chave inicial, com ou sem argumentos e um corpo. Executa uma série de instruções e pode ou não retornar um valor. (REFERENCE, 2000), (FIGUEIRA, 2016)	nome = function (argumento1 , ... , argumento n) (Comandos da função)
Variável	Uma variável é um nome que nos permite armazenar um valor qualquer. (REIS,)	nomevar <- valor
Argumento	O argumento é um nome que obtém um valor fornecido quando a função é chamada, ele define a variável e pode ser usado no corpo da função. (REFERENCE, 2000)	nome = expressão
Constante	Em R qualquer número digitado diretamente no prompt é uma constante e é avaliado. Existem cinco tipos de constantes: inteiro, lógico, numérico, complexo e string. (REFERENCE, 2000)	> 1 [1] 1 TRUE ou FALSE
Função matemática	Uma função matemática é composta por um nome e os dados que serão processados pela operação matemática entre parênteses. (SAMUEL-ROSA, 2020)	Sum( )
Função anônima	Uma função anônima é uma função que não possui nome e são usadas com mais frequência como argumentos para outras funções. (REFERENCE, 2000)	Sintaxe de função anônima (function(x) x * 10)(10) equivalente (normal) fun<-function(x) x * 10 fun(10)
Corpo	O corpo de uma função é qualquer expressão válida contida entre chaves. (REFERENCE, 2000)	{ ' e ' }

Retorno	Retorno do resultado ao final da função. (FIGUEIRA, 2016)	return

Catálogo de conceitos da linguagem R

Tabela 5 – Catálogo de conceitos da linguagem Lisp

Conceito	**Definição**	**Instância**
Função	Uma função define um nome, com ou sem argumentos e um corpo. Executa uma série de instruções e pode ou não retornar um valor. (LEITAO; CACHOPO, 1995), (MCCARTHY et al., 1965)	(defun nome (parâmetro-1 ...parâmetro-n) corpo)
Variável	Uma variável é um símbolo usado para representar o valor em uma função. (MCCARTHY et al., 1965)	fn: (LAMBDA (X Y) (CONS X Y))
Função lambda	Uma lambda é uma função com todas as características de uma função, mas que não está associada a nenhum símbolo. Pode ser vista como uma função sem nome. (LEITAO; CACHOPO, 1995)	> ((lambda (z) (+ z 3)) 2) 5
Predicado	Predicado é quando uma expressão condicional assume a forma de "se-então-caso contrário", resultando em verdadeiro ou falso. Os predicados devem ter concatenação da letra “p”. (LEITAO; CACHOPO, 1995)	> (> 4 3) t > (< 4 3) nil > (zerop 1) nil > (zerop 0) t
Expressão Condicional	Expressões cujo valor depende de um ou mais testes a realizar previamente, permitindo escolher vias diferentes para a obtenção do resultado. (LEITAO; CACHOPO, 1995)	Predicados Operadores Lógicos Selecção simples ..

Catálogo de conceitos da linguagem Lisp

Conceito	**Definição**	**Instância**
Função Recursiva	Uma função recursiva é uma função que se refere a si própria. (LEITAO; CACHOPO, 1995)	Na função factorial, o caso básico é o teste de igualdade a zero (zerop n), o resultado imediato é 1, e o passo recursivo é (* n (fact (- n 1)))
Variável Local	São variáveis locais, temporárias, para guardarem resultados parciais em outros lugares. Lisp providencia uma forma especial designada *let* que é convertida para uma lambda. (LEITAO; CACHOPO, 1995)	(let ((var-1 exp-1)
Função Local	Funções locais com a forma especial *flet*. A sua sintaxe é extremamente parecida com a do *let*, só que o valor de cada variável é a definição de uma função. (LEITAO; CACHOPO, 1995)	(defun teste (x)(flet ((f-local1 (y) (+ x y))
Átomos	Nome dos elementos que não podem ser decompostos. Para se testar se um elemento é atômico pode-se usar a função atom. (LEITAO; CACHOPO, 1995)	> (atom 1) T
Retorno	Forma especial que é o valor a retornar do ciclo da função. (LEITAO; CACHOPO, 1995)	return

Catálogo de conceitos da linguagem Lisp

Tabela 6 – Catálogo de conceitos da linguagem Haskell

Conceito	**Definição**	**Instância**

Catálogo de conceitos da linguagem Haskell

Conceito	**Definição**	**Instância**
Função	Uma função é uma definição de nome composta por argumentos separados ou não por vírgulas ou parênteses e que tem um retorno. (BOIS, 2008), (PIFFER; KROTH, 2002)	< nome da função > = < corpo > - x :: Int -> Int -> Float -> Bool -> Int
Variável	Uma variável é um nome que é atribuído a um valor uma vez e sua definição não muda de valor no mesmo problema. (MARLOW et al., 2010)	r = 2.0 (area = pi * r ^{2})
Identificador	Um nome identificador consiste em uma letra seguida por zero ou mais letras, dígitos, sublinhados e aspas simples. Os identificadores são sensíveis a maiúsculas (variável) e minúsculas (átomo) (MARLOW et al., 2010)	varid → (Pequeno pequena | grande | dígitos | ’) ⟨ reservedid ⟩
Função Recursiva	Uma função recursiva é uma função que chama a si mesma. (BOIS, 2008)	fatorial :: Int -> Int fatorial 0 = 1 fatorial n = n * fatorial (n-1)
Expressão Lambda	Definição de função em que a função não precisa ter um nome, apenas são armazenadas em uma variável. (BOIS, 2008)	sucessor :: Int -> Int sucessor x = x+1 Haskell > (x -> x + 1) 10 11
Retorno	Não realiza nenhuma interação. Apenas retorna um valor conforme a execução da função. (MALAQUIAS, 2016)	return

Catálogo de conceitos da linguagem Haskell

- **[CAPT-VIEW] Extract Ontology View**
  Atividade que auxilia a elaborar e extrair a visão das ontologias reusadas como parte da ontologia em construção, trazendo um conjunto de conceitos extraídos a partir delas.

Conforme atividade [CAPT-REUS] as ontologias apresentadas, exceto a SCO, não foram utilizadas, pois a cobertura apontada não está dentro do dimensionamento da ontologia FUNC-O. Já a ontologia SCO foi utilizada para apoiar o desenvolvimento da FUNC-O, uma vez que o modelo apresenta os conceitos base de um código-fonte e que pode ancorar os conceitos do código funcional. A *Software Process Ontology* (SPO) e a *Software Ontology* (SwO), identificadas com os acrônimos correspondentes (SPO:: e SwO::, respectivamente) são destacadas usando cores diferentes, a fim de reutilizar conceitos de engenharia de software relacionados a código-fonte. A Figura 6 apresenta a visão da ontologia SCO reusada na construção da ontologia FUNC-O.

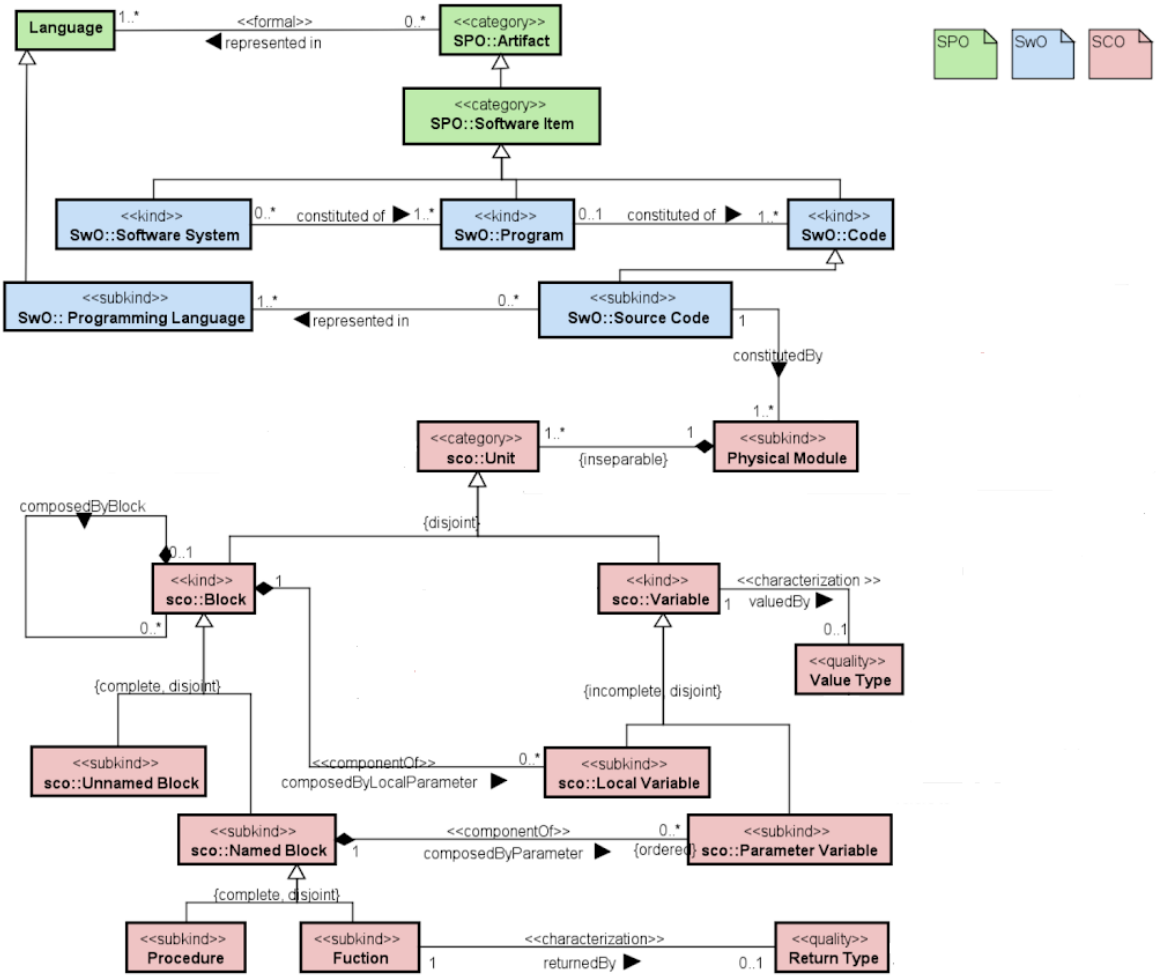

Figura 6 – Visão da Ontologia SCO. Fonte:(AGUIAR, 2021)

.

- **[CAPT-AXIM] Identify Axioms** Nesta atividade o propósito é identificar as restrições que o modelo conceitual não consegue representar e transformá-las em axiomas da ontologia em construção. Os axiomas da FUNC-O serão desenvolvidos em trabalhos futuros.

- **[CAPT-MODE] Model Ontology**
  A principal intenção desta atividade é a modelagem da ontologia através dos conceitos e relações identificados a partir das fontes de conhecimento e das ontologias reusadas. Essa modelagem deve seguir os padrões ontológicos e ser coberta pelo propósito definido independente da tecnologia utilizada. O modelo conceitual foi construído considerando um direcionamento conforme os subdomínios identificados e aplicado padrões ontológicos conceituais para garantir a qualidade do modelo. Através da atividade [KNOW-CONS] Concepts Consensus, que tem por objetivo estabelecer consenso semântico, foi definido o consenso entre os conceitos do domínio da ontologia em construção, com a intenção de obter clareza de possíveis dúvidas, inconsistências ou conflitos.

  A Tabela 7 apresenta o consenso de conceitos, com conceitos definidos na ontologia FUNC-O prefixados com **func::**, conceitos reutilizados da ontologia SCO-O prefixados com **sco::** e conceitos adicionados à ontologia FUNC-O, mas sugeridos para ser incorporados na ontologia SCO-O não são prefixados. A Figura 7 apresenta o modelo conceitual a partir do consenso dos conceitos, que será melhor descrito no capítulo 4.

Tabela 7 – Consenso dos conceitos

Conceito	**Definição**
func::Anonymous Block	Expressão anônima que não precisa ser definida, não precisa ter um nome, e pode ser utilizada dentro do código como se estivesse sido declarada como variável. São usadas com mais frequência como argumentos para outras funções.
sco::Mutability	Característica que permite ou não alterar o conteúdo ou estado de uma variável após sua inicialização. No código-fonte funcional, a mutabilidade é sempre falsa, porque uma vez que instanciamos um valor para uma variável, ela terá esse valor para sempre.
Block Name	Nome identificar de um dado bloco. No código-fonte funcional, especifica o nome da função.
func::Determinism	Característica que define sempre o mesmo valor de retorno para uma determinada entrada, ou seja, variáveis de parâmetro.
func::Math Function	É um tipo de função composta por um nome e pelos parâmetros que serão processados pela operação matemática entre parênteses. Esses dados realizam operações matemáticas simples. Quando são cálculos mais complexos as funções podem pertencer a um módulo, que deve ser importado para utilização.

Consenso dos conceitos

Conceito	**Definição**
func::Pure Function	É um tipo de função onde independente da quantidade de vezes que ela for chamada, o retorno sempre será o mesmo, ela não altera os parâmetros que são passados para ela, nem tem outros efeitos colaterais.
func::Impure Function	É um tipo de função que tem efeitos colaterais, nem sempre retorna o valor esperado, dado que esta função pode conter variáveis mutáveis, permitindo que essas variáveis possam receber atualizações a qualquer momento.
func::Recursive Function	É um tipo de função que ao ser executada chama a si mesma até o momento de parada definido.
func::Return Value	Valor retornado por uma função, referente ao tipo de retorno definido ou esperado.
Variable Name	Nome identificar de uma dada variável, utilizado para declaração e identificação.
Variable Value	Valor de um determinado tipo armazenado em uma variável para processamento e execução de um programa.
sco::Variable Type	É o tipo da variável que determina quais valores/conteúdos poderão ser armazenados nela.
Global Variable	É um tipo de variável que é declarada fora de um bloco, ou seja, pode ser utilizada por diferentes blocos e funções.

Consenso dos conceitos

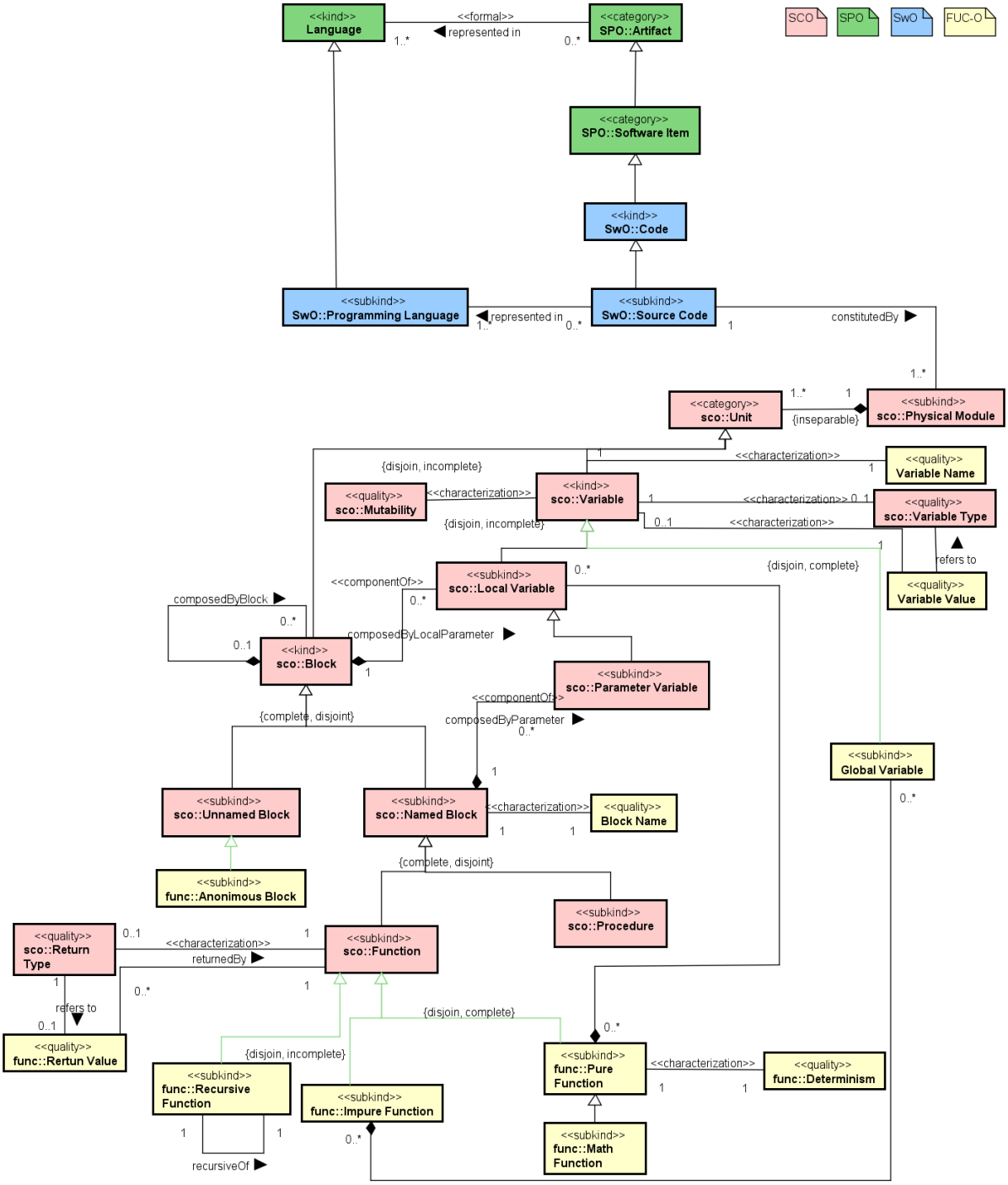

Figura 7 – Modelo conceitual de FUNC-O. Fonte: autor.

- **[CAPT-INTE] Integrate Ontology**
  Atividade que realiza a integração da ontologia em construção com as ontologias reusadas. Essa atividade é realizada em conjunto a atividade [CAPT-MODE] Model Ontology, pois a sua intenção é criar um modelo coerente e completo com todas as informações importantes para a linguagem. Para a integração da ontologia foi realizada uma ligação de modo que a visão da ontologia SCO reusada (cor rosa) apoiasse os conceitos da ontologia FUNC-O (cor amarela) com relações de generali-

zação (setas de cores verdes), especificando as características que eles impõem em suas instâncias, como apresentado na Figura 7.

- **[CAPT-MODU] Modularize Ontology**
  A Atividade de modularização tem por finalidade reconhecer os módulos da ontologia que possam ser separados mesmo estando associados a outros módulos. Essa atividade tem três objetivos principais para sua utilização: **1**- melhorar o comportamento da ontologia permitindo formas distintas de processamento, **2**- facilitar a construção e manutenção do modelo ontológico por permitir sua separação e **3**- facilitar o reuso de fragmentos da ontologia dado que todas os ajustes relevantes foram aplicados. Como a ontologia de código funcional é formada por poucos conceitos, FUNC-O foi construída por um único módulo, que se relaciona com as demais ontologias reusadas, como apresentado na Figura 8.

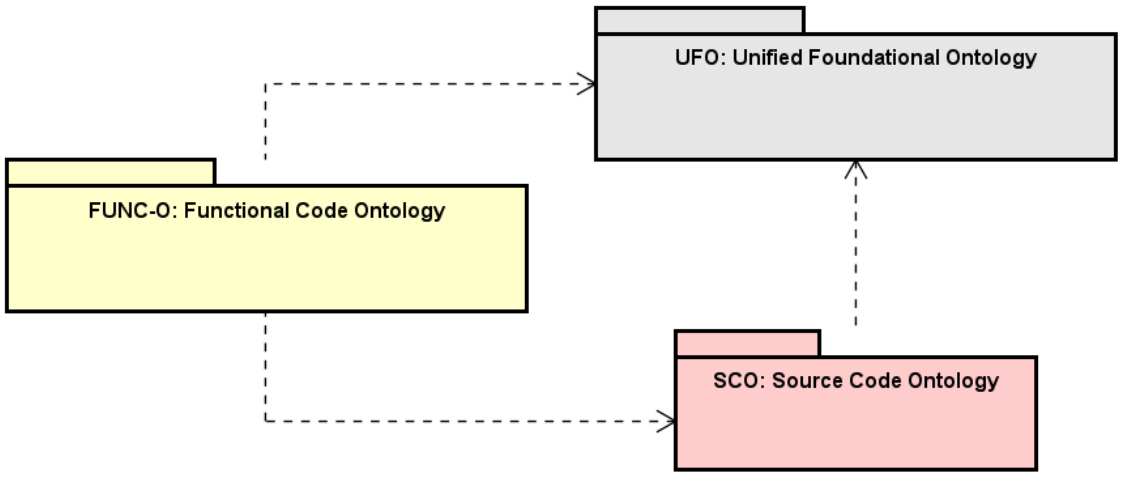

Figura 8 – Modularização da ontologia. Fonte: autor

.

- **[DOCM-MODE] Document Reference Ontology**
  Esta atividade utiliza o Documento de Ontologia de Referência como meio para documentar as premissas desenvolvidas nas atividades complementando o documento elaborado na atividade [DOCM-REFE] Document Premise of Reference Ontology. O documento de Ontologia de Referência produzido na fase de Reference para a ontologia FUNC-O é disponibilizado no repositório compartilhado DOCM-SPEC-FUNC-O.

- **[CONF-MODE] Control Reference Ontology**
  Atividade que controla as versões e alterações registradas no Documento de Referência. O controle do documento foi realizado através do cabeçalho do documento complementando os controles registrados na atividade [CONF-REFE] Control Premise of Reference Ontology, conforme apresentado na atividade [DOCM-MODE] Document Reference Ontology.

- **[EVAL-MODE] Evaluate Reference Ontology**
  Essa atividade busca realizar a verificação e validação da ontologia utilizando as informações registradas no Documento de Ontologia de Referência. Para verificação é necessário analisar se os requisitos levantados respondem as questões de competência com os conceitos e relações representados no modelo da ontologia, apresentados na Seção 5.1. Para validação é necessário analisar se todos os conceitos levantados atendem a situações da realidade, apresentado na Seção 5.2.

- **[PUBL-REFE] Publish Reference Ontology**
  Atividade que visa a publicação da Ontologia de Referência, definida na atividade [DOCM-MODE] Document Reference Ontology. A Ontologia de Referência FUNC-O está publicada no repositório compartilhado do projeto DOCM-SPEC-FUNC-O e no site do projeto no NEMO.

4 Ontologia de Código Funcional (FUNC-O)

A Ontologia de Código Funcional (FUNC-O) tem a intenção de apresentar características identificadas a partir de estudos sobre o paradigma de programação funcional, reconhecendo suas entidades de forma semântica e tornando explícito o entendimento sobre código-fonte funcional, disseminando o conhecimento sobre o domínio.

Dado esse escopo, a programação funcional é um paradigma que trata, principalmente, funções e problemas matemáticos. Essas funções são procedimentos que executam tarefas que, por meio de um conjunto de dados de uma ou mais entradas, produzirão um conjunto de dados de saída. Muitas das linguagens de programação existentes não são puramente funcionais. Algumas das linguagens que foram projetadas para essa abordagem de desenvolvimento funcional são multiparadigmas, que suportam diversos padrões de programação e que se adequam ao tipo de projeto desenvolvido dependendo da aplicação. Assim, esta ontologia limita-se à representação de conceitos de código-fonte funcional e não de outros paradigmas, como orientação a objetos com métodos e classes ou procedural com goto e jump.

Como apresentado no Capítulo 3, elicitamos os seguintes requisitos não funcionais para a FUNC-O: RNF01 – Ser fundamentada por uma ontologia de fundamentação para reutilizar o que já é consolidado na literatura, RNF02 – Ser definida a partir de uma fonte de conhecimento base e RNF03 – Estar em um padrão que facilite a interoperabilidade dos conceitos com outras ontologias existentes. Assim, para construção da ontologia utilizamos a ontologia de fundamentação UFO-A e a linguagem de modelagem OntoUML, definimos seus conceitos a partir de fontes de conhecimento da literatura e realizamos a sua integração com a SCO - Ontologia de Código Fonte que representa conceitos comuns e a diferentes tipos de linguagens, inclusive de função e variável que definimos como os subdomínios de FUNC-O. Para os requisitos funcionais definimos oito questões de competência: RF01 – Quais os principais conceitos de um código-fonte funcional?, RF02 – Quais os tipos de funções existentes em um código-fonte funcional?, RF03 – Quais os principais componentes de uma função?, RF04 – Quais variáveis compõem um código-fonte funcional?, RF05 – Quais variáveis compõem uma função?.

A construção da ontologia seguiu todas as fases apresentadas no Capítulo 3 de modo que o processo de construção fosse guiado e garantisse o entendimento do método SABiOS e do domínio de código-fonte funcional. A Figura 9 apresenta a ontologia FUNC-O, que foi integrada a SCO. Os conceitos definidos na ontologia FUNC-O são prefixados com **func::**, conceitos reutilizados da ontologia SCO-O são prefixados com **sco::** e conceitos adicionados à ontologia FUNC-O, mas sugeridos para ser incorporados na ontologia SCO-O não são

prefixados.

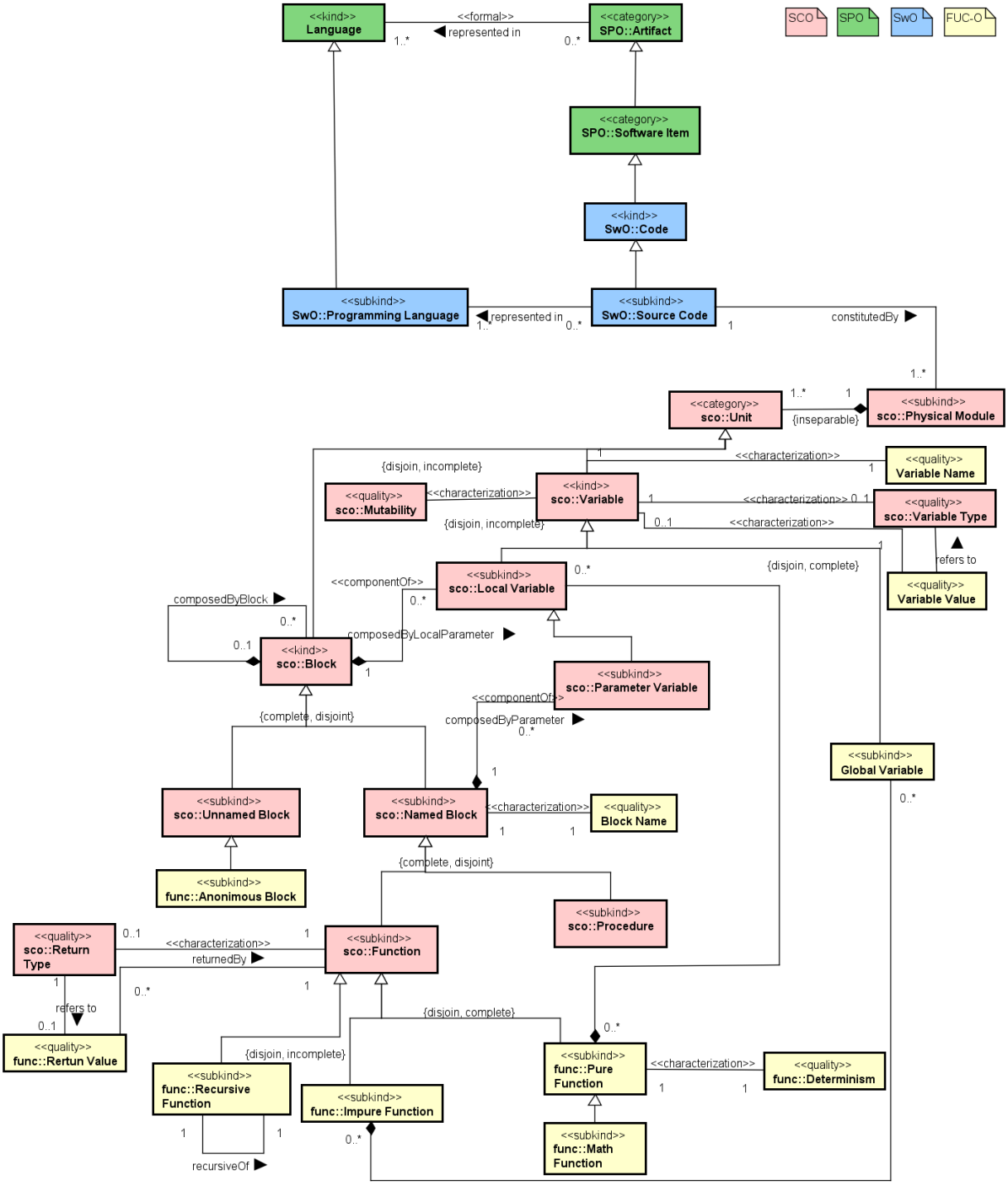

Figura 9 – FUNC-O - Ontologia de Código Funcional. Fonte:autor

.

A FUNC-O tem como objetivo representar os conceitos relevantes presentes nos códigos-fonte representados em linguagens de programação funcional em relação a função, suas definições e variáveis que a compõem. Por sua vez, FUNC-O está ancorada na *Source Code Ontology* (SCO), que estabelece uma conceituação comum no domínio de código-fonte

a partir da integração com as ontologias de *Software Process Ontology* (SPO) e *Software Ontology* (SwO).

Assim, SPO estabelece a conceituação comum sobre o domínio de processo de software, onde ***Artifact*** é um objeto consumido ou produzido durante o processo de software, que pode ser, entre outras coisas, um ***Software Item***, como um pedaço de software produzido durante o processo de software.

Em SwO, um *Software Item* é especializado em ***Code***, representando um conjunto de instruções de computador e definições de dados que são *represented in* ***Program Language*** como um ***Source Code***.

Em SCO, *Source Code* é *constituted of* ***Physical Modules***, ou seja, unidades físicas nas quais os arquivos físicos são armazenados. *Physical Modules* são *composed of* ***Units***, que podem ser ***Block***, uma sequência de instruções definidas por uma dada delimitação, ou ***Variable***, que mantém um item de informação localizado na memória. Enquanto um ***Unnamed Block*** é um *Block* que não tem um nome, um ***Named Block*** é nomeado por um nome identificador, podendo ser uma ***Fuction*** quando *returned by* ***Return Type*** ou ***Procedure*** quando não possui um tipo de retorno. *Variable* pode ser uma ***Parameter Variable*** declarada na assinatura de um *Named Block* ou ***Local Variable*** declarada dentro de um *Block*.

Em FUNC-O, os conceitos *Function* e *Variable* são especializados e uma análise ontológica é aplicada sobre SCO a partir das descobertas do domínio de código-fonte funcional. Assim, *Named Block* é *characterized by* ***Block Name*** e *composed by Parameter Variable*. Uma *Function*, por sua vez, é *characterized by* ***Return Value***, que atribui o valor retornado pela função que *refers to* ao *Return Type*. Uma *Function* pode ser uma ***Pure Function*** para manter a originalidade dos parâmetros e retornos esperados, sendo caracterizada pelo ***Determinism*** que define uma mesma saída para uma dada entrada, ou ***Impure Function*** para atualizar o estado de suas variáveis a qualquer momento, sendo uma ***Global Variable*** parte dela, uma vez que é uma variável acessível em todos os escopos da aplicação. Uma *Function* ainda pode ser definida para diferentes propósitos, tais como, ***Math Function*** para realização de cálculos e operações matemáticas ou ***Recursive Function*** para realizar chamadas dela mesma.

Function e *Variable* são caracterizados por um tipo e um valor correspondente a esse tipo, representados na ontologia como ***Variable Value*** e Return Value que *refers to* ***Variable Type*** e Return Type, respectivamente. Uma *Variable* também é *characterized* por ***Mutability*** uma vez que, quando instanciada com um valor, ela possuirá esse valor para sempre.

Named Block e *Variable* são caracterizados por um nome identificador representado na ontologia pelos conceitos *Block Name* e *Variable Name*. Por outro lado, *Unamed Block* é

especializado em ***Anonymous Block***, blocos que não precisam ser definidos previamente cuja sua criação geralmente tem o único propósito de passar o conteúdo do bloco para um bloco de ordem superior, sendo tratado pelo outro bloco como uma variável.

5 Avaliação

Para a avaliação da ontologia FUNC-O, adotamos a verificação e validação para avaliar se a ontologia atende às expectativas sobre os requisitos e propósito da ontologia.

5.1 Verificação de FUNC-O

Para a verificação do modelo conceitual, de acordo com o SABiOS, devemos identificar se os elementos que compõem a ontologia (conceitos e relações) são capazes de responder às questões de competência definidas na atividade [REQI-ELIC] Elicit Requirements, a fim de construir um raciocínio à medida que as perguntas vão sendo respondidas pelos conceitos. A Tabela 8 apresenta os resultados para as questões de competência levantadas para a ontologia FUNC-O.

Para os requisitos funcionais, conforme vimos anteriormente, foram elicitadas as seguintes questões de competência:
Subdomínio: Função
RF01 – Quais os principais conceitos de um código-fonte funcional?
RF02 – Quais os tipos de funções existentes em um código-fonte funcional?
RF03 – Quais os principais componentes de uma função?

Subdomínio: Variável
RF04 – Quais variáveis compõem um código-fonte funcional?
RF05 – Quais variáveis compõem uma função?

Tabela 8 – Resultado da verificação FUNC-O

ID	**Questão de Competência**
RF01	Artifact *specialized in* Software Item Software Item *specialized in* Code Code *specialized in* Source Code Source Code *constituted by* Physical Module Physical Module *composed by* Unit Unit *specialzied in* Block, Variable and Clause
RF02	Function *specialized in* Math Function, Pure Function, Recursive Function, Impure Function.
RF03	Function *subtype of* Named Block Named Block *subtype of* Block Named Block *composed by* Block Name Block *composed by* Local Variable
RF04	Variable *specialized in* Local Variable and Parameter Variable Function composed by Parameter Variable Function composed by Local Variable Parameter Variable *specialized in* Anonymous Variable
RF05	Function *subtype of* Named Block Named Block *subtype of* Block Named Block *composed by* Parameter Variable Block *composed by* Local variable

5.2 Validação de FUNC-O

Para a validação da ontologia é necessário analisar se todos os conceitos levantados atendem a situações da realidade, por meio da instanciação de seus conceitos. Para isso, seguindo o método SABiOS, representamos uma função fatorial no código-fonte de cada uma das linguagens de programação estudadas (Python, R, Lisp e Haskell). A Tabela 9 apresenta os resultados dessa instanciação na ontologia FUNC-O.

Tabela 9 – Resultado da Validação de FUNC-O

Código	**Instância FUNC-O:**

Python:
`1 def fatorial(x):`
`2  if x == 1:`
`3  return 1`
`4  else:`
`5  return x * fatorial(x - 1)`
	Recursive Function: bloco declarado da linha 1 a 5
	Block Name: fatorial
	Parameter Variable: x
	Return Value = 1
	Return Value = x * fatorial(x - 1)

Código	**Instância FUNC-O:**

R:
`1 fact = function( n ) (`
`2  if( n <= 1) (`
`3  return(1)`
`4  )`
`5  else (`
`6  return(n * fact(n-1)`
`7  )`
`8 )`
	Recursive Function = bloco declarado da linha 1 a 8
	Block Name = fact
	Parameter Variable = n
	Return Value = 1
	Return Value = (n * fact(n-1))

Haskell:
`1 fatorial :: Integer -> Integer fatorial n`
`2 | n == 0 = return 1`
`3 | n > 0 = fatorial (n-1) * n &`
	Recursiva Function = bloco declarado da linha 1 a 3
	Block Name = fatorial
	Parameter Variable = n
	Value Type = Integer
	Return Type = Integer
	Return Value = 1
	Return Value = fatorial (n1) n

Lisp:
`1 (defun fatorial (n)`
`2  (if ( = n 0)`
`3  1`
`4  (* n (fatorial (- n 1)))`
`5  )`
`6 )`
	Recursive Function = bloco declarado da linha 1 a 6
	Block Name = fatorial
	Parameter Variable = n

Resultado da Validação de FUNC-O

Em todas as linguagens selecionadas, identificamos a instanciação de *Function* a partir da representação semântica de *Recursive Function*, onde as funções realizam a chamada delas mesmas pela relação *recursive of* na ontologia. Por sua vez, *Function* é definida com nome identificador (*fatorial*, *fact* e *factorial*) em todas as linguagens, incorporando a semântica de *Block Name* em Python, R, Haskell e Lisp. *Function* também apresenta *Parameter Variable* em todas as linguagens e O *Return Value* é representado por todas as linguagens selecionadas, embora em algumas esse retorno não apareça explicitamente no código-fonte como em Lisp.

Embora os princípios de código funcional estejam bem estabelecidos, o tratamento nas linguagens de programação não é uniforme. Cada linguagem tem a sua particularidade

e adota sintaxe e semântica diferentes. Neste contexto, FUNC-O pode ser usada para apoiar a interoperabilidade entre eles.

Para validar que a ontologia FUNC-O aborda os conceitos além das linguagens estudadas, instanciamos a ontologia com código-fonte da linguagem de programação Scheme. A Tabela 10 demonstra que os conceitos da ontologia FUNC-O foram instanciados corretamente para o código-fonte em linguagem Scheme.

Tabela 10 – Resultado da validação FUNC-O Linguagem Scheme

Código da Linguagem:	**Instância FUNC-O:**
1 (define fatorial 2 (lambda (n) 3 (if (= n 0) 4 1 5 (* n (fatorial (- n 1))))))	*Recursive Function* = bloco declarado da linha 1 a 5 *Block Name* = fatorial *Anonimous Block* = lambda *Parameter Variable* = n

Como observado, *fatorial* é uma ***Function*** com ***Block Name*** *fatorial*. *Lambda* é um ***Anonymous Block*** utilizado para criar a função composta pela ***Parameter Variable*** "n", de modo que *fatorial* é o *Block Name* associado ao valor da expressão representada por lambda. Por fim, **Return Value** é representado de maneira implícita na linguagem.

5.3 Mapeamento de FUNC-O

Uma vez que FUNC-O representa os conceitos de código-fonte funcional de forma consensual, a Tabela 11 apresenta o mapeamento desses conceitos para as linguagens estudadas Python, R, Haskell e Lisp. Os conceitos da ontologia que não são representados pela respectiva linguagem são mapeamentos com "-".

Tabela 11 – Mapeamento de FUNC-O

FUNC-O	**Python**	**R**	**Haskell**	**Lisp**
Anonymous Block	Lambda Function	Anonymous Function	Lambda Express	Lambda Function
Function	Function	Function	Function	Function
Math Function	Math Function	Math Function	Math Function	Math Function
Pure Function	Pure Function	-	Pure Function	Pure Function
Impure Function	Impure Function	-	Impure Function	Impure Function

Recursive Function	Recursive Function	Recursive Function	Recursive Function	Recursive Function
Rerturn Value	Rerturn Value	Rerturn Value	Rerturn Value	Rerturn Value
Variable Name	Variable	Variable	Variable	Variable
Variable Value	Variable	Variable	Variable	Atom
Variable Type	Variable	Variable	Variable	Variable
Global Variable	Variable	Variable	Variable	Variable

Mapeamento de FUNC-O

6 Conclusão

Este capítulo apresenta na Seção 6.1 as considerações finais do trabalho, principais contribuições, dificuldades e limitações encontradas. Na Seção 6.2 são apresentadas perspectivas e possibilidades de trabalhos futuros.

6.1 Considerações Finais

Em virtude do avanço progressivo do desenvolvimento de linguagens de programação para áreas de aplicações distintas, houve também o crescimento de problemas relacionados à interoperabilidade semântica em código fonte. Esses problemas causam comprometimento da capacidade de interpretação correta de informações e embora já estejam sendo abordados na área de Engenharia de Ontologias, ainda não houve a sua representação de forma completa, uma vez que existem paradigmas de linguagens de programação, as separando em categorias de linguagens e tornando esse meio cada vez mais amplo.

Este trabalho investigou, através do estudo de linguagens de programação, o paradigma funcional e o método SABiOS, baseado no método SABiO (FALBO, 2014), de modo que fosse aplicado Engenharia de Ontologias para a construção da ontologia de programação funcional FUNC-O, a tornando bem fundamentada e que apresentasse uma conceituação consensual e compartilhada deste domínio conforme a motivação e objetivos definidos na Seção 1.1 e Seção 1.2.

Em sua aplicação e para validação do método, foi realizado atividades dos processos principais e de suporte de todo o ciclo de vida da ontologia, desde a fase de concepção até a fase de referencia, conforme 2.4 e 3. A abordagem foi desenvolvida utilizando a UFO *(Unified Foundational Ontology)*, uma ontologia de fundamentação, de maneira que foi melhorado e dado suporte a construção do modelo conceitual. O método SABiOS se mostrou um método fácil e de compreensão clara, guiando mais detalhadamente o processo de desenvolvimento da ontologia através de cada fase e atividade proposta.

Assim, a FUNC-O apresentou características identificadas a partir de estudos sobre o paradigma de programação funcional, reconhecendo suas entidades semanticamente tornando explícito o entendimento deste paradigma, foi integrada a SCO, ancorando o fragmento da ontologia estabelecen uma conceituação comum no domínio. Desse modo, a FUNC-O pôde auxiliar na disseminação do conhecimento sobre o domínio, para que assim o este conhecimento sirva de apoio a programadores, que virão a utiliza-la.

Uma das maiores dificuldades ao longo do desenvolvimento do trabalho, foi que nem todas as informações à respeito dos conceitos de cada linguagem estudada estavam disponi-

bilizados por meio de textos, sites ou livros, sendo este um desafio para a interpretação de conceitos, assim sendo necessário realizar um estudo mais aprofundado de cada linguagem, assumindo algumas premissas, pois abriam margens para diversas compreensões. Apesar dessas limitações, foi possível realizar a representação unificada dessas informações.

6.2 Trabalhos Futuros

Trabalhos futuros estão direcionados ao amadurecimento e expansão da ontologia até a fase operacional. Assim, será continuado o estudo aprofundado do método SABiOS, com a finalidade de aplicar os passos para a construção e validação da ontologia operacional. A ontologia elaborada foi verificada a partir de questões de competência e validada com código-fonte de programas reais representados em linguagem funcional. Futuramente, também pretende-se integrar a ontologia à rede de ontologias de código fonte (SCON) (AGUIAR, 2021) e contribuir com o amadurecimento do método SABiOS por meio de lições aprendidas e do desenvolvimento de uma ferramenta informacional para auxiliar sua aplicação.

Como experiência pessoal, finalizo este trabalho dando destaque ao grande aprendizado que me foi proporcionado, desde o início das atividades com todas as pesquisas e estudos realizados até a sua conclusão. De certo modo, pude desenvolver uma experiência em Projeto de Pesquisa no qual este trabalho está contextualizado, meus conhecimentos acerca do domínio de programação funcional foram substancialmente melhorados. Aprendi a pensar de forma mais ampla e obtive capacidade de solucionar questões por meio de um raciocínio mais eficiente.

Referências

AGUIAR, C. Z. d.; ZANETTI, F.; SOUZA, V. E. S. Source code interoperability based on ontology. In: *XVII Brazilian Symposium on Information Systems*. [S.l.: s.n.], 2021. p. 1–8. Citado na página 30.

AGUIAR, C. Zacche de. *Interoperabilidade Semântica entre Códigos Fonte baseada em Ontologia*. Tese (PhD Thesis) — Universidade Federal do Espírito Santo, Brasil, 2021. Em construção. Citado 8 vezes nas páginas 7, 11, 12, 24, 25, 26, 38 e 54.

ALURA. *Python e Orientação a Objetos*. 2020. Disponível em: <https://www.caelum.com.br/apostila/apostila-python-orientacao-a-objetos.pdf>. Acesso em: 2021. Citado na página 32.

BARCELLOS, M. Uma estratégia para medição de software e avaliação de bases de medidas para controle estatístico de processos em organizações de alta maturidade. In: . [S.l.: s.n.], 2009. Citado na página 21.

BERTOLINI, C. et al. Linguagem de programação i. Brasil, 2019. Citado 2 vezes nas páginas 15 e 16.

BOIS, A. R. D. Programação funcional com a linguagem haskell. *Disponivel em: http://www. inf. ufpr. br/andrey/ci062/ProgramacaoHaskell. pdf*, 2008. Citado na página 37.

CLOCKSIN, W. F.; MELLISH, C. S. *Programming in Prolog*. [S.l.]: 5e. Springer-Verlag, Nova York, Estados Unidos, 2003. Citado na página 16.

COUTINHO, T. O que é uma linguagem de programação, para que serve e como funciona? *voitto*, Voitto Group, 2020. Citado na página 15.

DOWNEY, A. B. *Pense em Python*. [S.l.]: Novatec Editora, 2016. Citado 2 vezes nas páginas 32 e 33.

FALBO, R. A. SABiO: Systematic Approach for Building Ontologies. In: GUIZZARDI, G. et al. (Ed.). *Proc. of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering*. Rio de Janeiro, RJ, Brasil: CEUR, 2014. Citado 6 vezes nas páginas 11, 12, 20, 23, 24 e 53.

FALBO, R. de A. et al. Organizing ontology design patterns as ontology pattern languages. In: SPRINGER. *Extended Semantic Web Conference*. [S.l.], 2013. p. 61–75. Citado na página 20.

FARINELLI, F.; MELO, S.; ALMEIDA, M. B. O Papel das Ontologias na Interoperabilidade de Sistemas de Informação: Reflexões na esfera governamental. In: *Encontro Nacional de Pesquisa em Ciência da Informação (ENANCIB)*. [S.l.: s.n.], 2013. ISBN 9788578110796. ISSN 1098-6596. Citado na página 12.

FENSEL, D. et al. On-to-knowledge: ontology-based tools for knowledge management. In: *Proceedings of the eBusiness and eWork2000 Conference*. [S.l.: s.n.], 2000. Citado na página 23.

FERNÁNDEZ-LÓPEZ, M.; GÓMEZ-PÉREZ, A.; JURISTO, N. Methontology: from ontological art towards ontological engineering. American Association for Artificial Intelligence, 1997. Citado na página 22.

FIGUEIRA, L. Estatística é com r. *Estatística*, 2016. Citado 2 vezes nas páginas 34 e 35.

FRANÇA, P. C. Conceitos, classes e/ou universais: com o que é que se constrói uma ontologia? *Linguamática*, v. 1, n. 1, p. 105–121, 2009. Citado na página 18.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, Elsevier, v. 43, n. 5-6, 1993. Citado na página 19.

GUARINO, N. *Formal Ontology in Information Systems. Formal Ontology in Information Systems, FOIS'98, Trento, Italy*. [S.l.]: IOS Press, 1998. Citado 4 vezes nas páginas 7, 19, 20 e 21.

GUIZZARDI, G. Uma abordagem metodológica de desenvolvimento para e com reuso, baseada em ontologias formais de domínio. *Programa de Mestrado em Informática–Universidade Federal do Espírito Santo, Vitória, http://wwwhome. cs. utwente. nl/˜ guizzard/MSc*, 2000. Citado na página 19.

GUIZZARDI, G. Ontological foundations for structural conceptual models. Telematica Instituut / CTIT, Enschede, 2005. Citado 3 vezes nas páginas 19, 20 e 21.

GUIZZARDI, G. et al. Ontologias de fundamentação, modelagem conceitual e interoperabilidade semântica. In: CITESEER. *Proceedings of the Iberoamerican Meeting of Ontological Research. Anais*. [S.l.], 2011. Citado na página 21.

GUIZZARDI, G. et al. Towards ontological foundations for conceptual modeling: The unified foundational ontology (ufo) story. *Applied ontology*, IOS Press, v. 10, n. 3-4, p. 259–271, 2015. Citado na página 21.

IME-USP, D. de Ciência da C. *Variáveis, expressões e comandos*. 2016. Disponível em: <https://panda.ime.usp.br/panda/static/aulasPython/aula02.html>. Acesso em: 2021. Citado na página 32.

ISOTANI, S.; BITTENCOURT, I. I. *Dados Abertos Conectados: Em busca da Web do Conhecimento*. [S.l.]: Novatec Editora, 2015. Citado 4 vezes nas páginas 7, 18, 22 e 23.

JUCA, A. Uma introdução a cálculo lambda. *medium*, Medium Group, 2018. Citado na página 17.

LEITAO, A. M.; CACHOPO, J. Introdução á linguagem lisp. *Disponível na em*, 1995. Citado 2 vezes nas páginas 35 e 36.

MACIEL, T. S. *Uso de Teorias Ontológicas para Modelagem de Transtornos Psicológicos*. [S.l.], 2019. Citado na página 18.

MALAQUIAS, J. R. Programação funcional em haskell. *decom BCC222*, 2016. Citado na página 37.

MARLOW, S. et al. Haskell 2010 language report. 2010. Citado na página 37.

MCCARTHY, J. et al. *LISP 1.5 Programmers Manual*. [S.l.]: 2e. MIT Press, Cambridge, Estados Unidos, 1965. Citado 2 vezes nas páginas 16 e 35.

MEESTER, B. D. The function ontology. 2021. Citado 2 vezes nas páginas 30 e 32.

NASCIMENTO, F. Programação funcional: O que é? *alura*, Alura Group, 2019. Citado na página 18.

NOY, N. F.; MCGUINNESS, D. L. et al. *Ontology development 101: A guide to creating your first ontology*. [S.l.]: Stanford knowledge systems laboratory technical report KSL-01-05, 2001. Citado na página 22.

PASINI, R. *Estudo e projeto de uma ontologia para a área da saúde*. [S.l.], 2009. Citado 2 vezes nas páginas 19 e 23.

PIFFER, M. M.; KROTH, R. A. A linguagem funcional haskell. 2002. Citado na página 37.

REFERENCE, M. P. Manual de referência da linguagem python. *Copyright*, v. 2, 2020. Citado 2 vezes nas páginas 32 e 33.

REFERENCE, M. R. Manual de referência da linguagem r. *Copyright*, v. 4.2.2, 2000. Citado na página 34.

REIS, F. dos. *Declaração e Atribuição de Variáveis em R*. Disponível em: <http://www.bosontreinamentos.com.br/programacao-em-r/declaracao-e-atribuicao-de-variaveis-em-r/>. Acesso em: 2022. Citado na página 34.

SAMUEL-ROSA, A. *Introdução ao R para análise de dados e construção de gráficos*. [S.l.]: bookdown, 2020. Citado na página 34.

SCHMITT, F. A.; SILVA, B. da et al. Utilização da programação funcional no mundo dos blocos geométricos. Florianópolis, SC, 2003. Citado 2 vezes nas páginas 17 e 18.

SEBESTA, R. W. *Conceitos de Linguagens de Programação-11*. [S.l.]: Bookman Editora, 2011. Citado 3 vezes nas páginas 15, 17 e 18.

SILVEIRA, S. R. et al. Paradigmas de programação: Uma introdução. 1e. SINAPSE, Belo Horizonte, p. 95, 2021. Citado na página 18.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge engineering: Principles and methods. *Data & knowledge engineering*, Elsevier, v. 25, n. 1-2, 1998. Citado na página 19.

SUCHANEK, M. *Especificação OntoUML*. 2018. Disponível em: <https://ontouml.readthedocs.io/en/latest/intro/index.html>. Acesso em: 2023. Citado na página 21.

SWARTOUT, W.; TATE, A. Ontologies. *IEEE Intelligent Systems and Their Applications*, v. 14, n. 1, p. 18–19, 1999. ISSN 16877438. Citado na página 11.

TERRA, M. G. et al. Modelagem de uma ontologia para construção de algoritmos. *Revista do CCEI*, v. 21, n. 36, p. 32–46, 2017. Citado na página 30.

VAREJÃO, F. Linguagens de programação java, c e c++ e outras: conceitos e técnicas, pp. 6–12. *Campus, Rio de Janeiro*, 2004. Citado 5 vezes nas páginas 7, 11, 15, 16 e 18.

VAZ, P. M. *Programação Python*. Disponível em: <https://www.codingame.com/playgrounds/52499/programacao-python-intermediario---prof--marco-vaz/funcao-lambda>. Acesso em: 2022. Citado na página 32.

ZAMBORLINI, V. C. Estudo de alternativas de mapeamento de ontologias da linguagem ontouml para owl: Abordagens para representação de informação temporal. *Federal University of Espírito Santo. Available only in Portuguese*, 2011. Citado 2 vezes nas páginas 18 e 21.