UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

OntoScratch: ontologias para representação e avaliação de habilidades do pensamento computacional em projetos Scratch

Nícolas Oreques de Araujo

Pelotas, 2020

Nícolas Oreques de Araujo

OntoScratch: ontologias para representação e avaliação de habilidades do pensamento computacional em projetos Scratch

Dissertação apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Tiago Thompsen Primo
Coorientadora: Prof Dra. Ana Marilza Pernas

Pelotas, 2020

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

A111o Araujo, Nícolas Oreques de

OntoScratch: : ontologias para representação e avaliação de habilidades do pensamento computacional em projetos Scratch / Nícolas Oreques de Araujo ; Tiago Tompsen Primo, orientador ; Ana Marilza Pernas, coorientadora. — Pelotas, 2020.

104 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2020.

1. Engenharia de ontologias. 2. Modelagem semântica. 3. Modelagem de dados. 4. Pensamento computacional. 5. Scratch. I. Primo, Tiago Tompsen, orient. II. Pernas, Ana Marilza, coorient. III. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Nícolas Oreques de Araujo

OntoScratch: ontologias para representação e avaliação de habilidades do pensamento computacional em projetos Scratch

Dissertação aprovada, como requisito parcial, para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 20 de julho de 2020

Banca Examinadora:

Prof. Dr. Tiago Tompsen Primo (orientador)
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Marilton Sanchotene de Aguiar
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dra. Larissa Astrogildo de Freitas
Doutora em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul.

Aos meus avós, pelos ensinamentos.
À minha namorada, pelo companheirismo.
Aos meus amigos, pelos momentos compartilhados.
Aos meus pais e minha família, pelo suporte e apoio.
Aos meus orientadores, por terem embarcado e auxiliado nesta aventura de 6 meses.

"You can’t change the wind, but you can set your sails."
— BILLIE JOE ARMSTRONG

RESUMO

DE ARAUJO, Nícolas Oreques. **OntoScratch: ontologias para representação e avaliação de habilidades do pensamento computacional em projetos Scratch**. Orientador: Tiago Thompsen Primo. 2020. 105 f. Dissertação (Mestrado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2020.

Diversas iniciativas visando introduzir conceitos relacionados a computação no currículo escolar tem iniciado em países da Europa, Estados Unidos e, mais recentemente, no Brasil. Um exemplo disto é a aplicação do projeto Clubes de Computação Criativa, realizado desde 2015 em escolas públicas na cidade de Pelotas/RS. Dentre a lista de habilidades e conhecimentos abordados, destaca-se o estímulo ao desenvolvimento do pensamento computacional através de ferramentas de programação visual, como o Scratch, devido a sua abordagem prática e de baixo custo. Porém, neste cenário, surgem novos desafios, como o armazenamento dos dados coletados nestas atividades, para que possam ser trabalhados e para que gerem evidências do impacto do ensino do pensamento computacional. Nesta questão, destaca-se a ausência de uma representação de conhecimento capaz de caracterizar os dados de atividades e avaliações que utilizem o Scratch de forma a possibilitar a análise da evolução do aluno ou a realização de outras inferências sobre estes dados. Assim, este trabalho tem como objetivo a criação e o desenvolvimento de uma representação de conhecimento que ataque estes problemas, caracterizando um projeto Scratch e as avaliações de habilidades de pensamento computacional realizadas sobre ele, de forma a prover suporte para a realização de análises e inferências sobre estes dados. Espera-se que esta representação seja capaz de dar suporte a utilização de mecanismos inteligentes, possibilitando a realização de análises e inferências que auxiliem na compreensão do conhecimento adquirido pelo aluno, sua evolução e outros aspectos relacionados a seu aprendizado. Estes objetivos puderam ser atingidos através da concepção do OntoScrath, um conjunto de ontologias para representação do conhecimento envolvendo o ensino de pensamento computacional através do Scratch. Através desta representação, é possível armazenar de maneira estruturada o percurso dos alunos neste processo, permitindo a análise de seu processo de aprendizagem. Esta representação demonstrou seu potencial de inferência na avaliação realizada, sendo capaz de reproduzir fielmente, através de consultas SPARQL, a avaliação de habilidades do pensamento computacional da ferramenta Dr. Scratch.

Palavras-chave: Engenharia de Ontologias. Modelagem Semântica. Modelagem de Dados. Scratch. Pensamento Computacional. Computational Thinking.

ABSTRACT

DE ARAUJO, Nícolas Oreques. **OntoScratch: ontologies to represent and evaluate computational thinking abilities in Scratch Projects**. Advisor: Tiago Thompsen Primo. 2020. 105 f. Dissertation (Masters in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2020.

A lot of initiatives seeking to introduce concepts and habilities related to computational thinking are underway in countries across Europe, in the United States, and, more recently, in Brazil. An example of that is the application of the project "Creative Computing Clubs", carried out since 2015 in public schools in the city of Pelotas/RS. Among the list of skills and knowledge covered, the stimulus to the development of computational thinking through visual programming tools, such as Scratch, stands out due to its practical and low-cost approach. However, in this scenario, new challenges arise, such as the storage of the data collected in these activities, so that they can be worked on and so that they can generate evidence of the impact of teaching computational thinking. In this matter, the absence of a knowledge representation that is capable of characterizing the data of activities and evaluations that use Scratch in a way that allows the analysis of the student's evolution or the realization of other inferences about these data is clear. Therefore, this work aims to create and develop a knowledge representation that attacks these problems, describing a Scratch project and the computational thinking skills assessments performed on it, in a structure that provides support for the use of this data for further work. It is expected that this representation will be able to support the use of intelligent mechanisms, enabling the performance of analyzes and inferences that assist in the understanding of the knowledge acquired by the student, its evolution, and other aspects related to his learning. These objectives could be achieved through the design and development of OntoScrath, a set of ontologies for representing knowledge involving the teaching of computational thinking through Scratch. Through this representation, it is possible to store, in a structured way, students' paths in this process, allowing the analysis of their learning process. This representation exhibited its potential for inference in the evaluation performed, being able to faithfully reproduce, through SPARQL queries, the computational thinking skills' evaluation of the tool Dr. Scratch.

Keywords: Ontology Engineering. Semantic Modeling. Data Modeling. Scratch.

LISTA DE FIGURAS

Figura 1 Número de usuários ativos mensalmente na plataforma Scratch . . 14

Figura 2 Taxonomia para características do estudante 24
Figura 3 Um modelo de estudante utilizando a abordagem *overlay* 25
Figura 4 Um modelo de estudante utilizando a abordagem *perturbation* . . . 27
Figura 5 Tipos de ontologia de acordo com seu nível de especificação 32
Figura 6 Avaliação de um projeto no Dr. Scratch 35
Figura 7 Modelo de dados proposto para rede de conhecimento Scratch . . . 37

Figura 8 Diagrama do Ecossistema 39
Figura 9 Conjunto de Ontologias do OntoScratch 40
Figura 10 Representação ontológica para um projeto Scratch 42
Figura 11 Hierarquia de classes para atores 43
Figura 12 Exemplos de *Frames* 44
Figura 13 Hierarquia de classes para frames 44
Figura 14 Hierarquia de classes para bloco 45
Figura 15 Exemplo de classe para um bloco 46
Figura 16 Formatos de um bloco Scratch 46
Figura 17 Hierarquia de classes para formatos de bloco 47
Figura 18 Categorias de um bloco Scratch 48
Figura 19 Hierarquia de classes para categorias de bloco 49
Figura 20 Representação de *Inputs* 51
Figura 21 Hierarquia de classes para *Inputs* 52
Figura 22 Procedimentos em Scratch 54
Figura 23 Representação de Procedimentos - categoria *MyBlock* 55
Figura 24 Representação de Conceitos de Pensamento Computacional 56
Figura 25 Representação de avaliação de Pensamento Computacional de um projeto 57
Figura 26 Hierarquia de Classes Simplificada do OntoScratch 59

Figura 27 Projetos de Teste 62
Figura 28 Representação do Projeto 1 65
Figura 29 Avaliação do Projeto 1 na plataforma Dr. Scratch 66
Figura 30 Avaliação do Projeto 1 na OntoScratch 67
Figura 31 Representação do Projeto 2 68
Figura 32 Avaliação do Projeto 2 na plataforma Dr. Scratch 69
Figura 33 Avaliação do Projeto 2 na OntoScratch 70

LISTA DE TABELAS

Tabela 1 Conceitos base do pensamento computacional 21
Tabela 2 Regras de avaliação da ferramenta Dr. Scratch 23

LISTA DE ABREVIATURAS E SIGLAS

ACM	*Association for Computing Machinery*
BNCC	Base Nacional Comum Currícular
CSTA	*Computer Science Teachers Association*
ISTE	*International Society for Technology in Education*
FOAF	*Friend-of-a-Friend*
OWL	*Web Ontology Language*
PC	Pensamento Computacional
RDF	*Resource Description Framework*
RDFS	*RDF Schema*
SM	*Student Model*
SMED	Secretaria Municipal de Educação
SPARQL	SPARQL Protocol and RDF Query Language
UFPel	Universidade Federal de Pelotas
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Objetivos 16
1.2 Organização do Trabalho 17

2 FUNDAMENTAÇÃO TEÓRICA 18
2.1 Pensamento Computacional 18
2.1.1 Educação *Maker* 19
2.1.2 Scratch 20
2.1.3 Avaliação de Pensamento Computacional 21
2.2 Student Model 23
2.2.1 Overlay 26
2.2.2 Perturbation 27
2.2.3 Stereotype 28
2.2.4 Fuzzy 29
2.2.5 Ontologias 29
2.2.6 Redes Bayesianas 30
2.3 Ontologias 30
2.3.1 Linguagens de Criação e Manipulação de Ontologias 33
2.4 Trabalhos Relacionados 35
2.5 Considerações do Capítulo 39

3 ONTOSCRATCH: CONCEPÇÃO E TECNOLOGIAS 40
3.1 Escopo 41
3.2 Representação do Universo Scratch 42
3.2.1 Projeto 43
3.2.2 Atores 43
3.2.3 Frames 44
3.2.4 Blocos 46
3.2.5 Inputs 52
3.2.6 Definição de Procedimentos 54
3.3 Representação da Avaliação de Pensamento Computacional 56
3.3.1 Conceitos 57
3.3.2 Avaliação 58
3.4 Considerações do Capítulo 59

4 AVALIAÇÃO E RESULTADOS 62
4.1 Metodologia 62
4.2 Avaliação dos Projetos 63
4.3 Resultados 65
4.3.1 Projeto de Testes 1 65
4.3.2 Projeto de Testes 2 68
4.4 Considerações do Capítulo 71

5 CONSIDERAÇÕES FINAIS 72
5.1 Conclusão 72
5.2 Trabalhos Futuros 73

REFERÊNCIAS 75

APÊNDICE A CONSULTAS SPARQL 84
A.1 Abstração 84
A.2 Controle de Fluxo 86
A.3 Interatividade com Usuário 88
A.4 Paralelismo 92
A.5 Pensamento Lógico 95
A.6 Representação de Dados 97
A.7 Sincronização 102
A.8 Pontuação Total 104

1 INTRODUÇÃO

A computação está presente de forma praticamente intrínseca na vida humana, não sendo possível mais conceber uma sociedade sem o uso do computador (BLINKSTEIN, 2019). Assim, diversas iniciativas visando introduzir conceitos relacionados a computação na educação básica tem iniciado em países da Europa, Estados Unidos e, mais recentemente, no Brasil (BLIKSTEIN, 2018).

Dentre a lista de habilidades e conhecimentos introduzidos nestas iniciativas, o Pensamento Computacional (PC) talvez seja o mais importante e o menos compreendido (BLIKSTEIN, 2008). Ele baseia-se em conceitos de Computação, envolvendo a solução de problemas, compreensão do comportamento humano e capacidade de projetar sistemas, representando uma forma de pensamento analítico (DE FRANÇA; AMARAL, 2013). Esta habilidade busca ser incentivada pela educação *maker*, uma metodologia que combina o Construcionismo e o Construtivismo Social (DOUSAY, 2017). Assim, ela combina duas epistemologias distintas: o aprendizado através de interações em um grupo e o aprendizado através do ato de criação.

Nestas iniciativas, o uso de ferramentas de programação visual, como o Scratch, vem sendo a principal abordagem, destacando-se como uma ferramenta prática e de baixo custo. Por sua vez, o Scratch é um ambiente de programação visual desenvolvido pelo MIT Media Lab, baseando-se nas ideias construcionistas do Logo (SEYMOUR, 1980) e do Etoys (MALONEY et al., 2010) e permitindo a criação de histórias interativas e jogos. Seu objetivo é introduzir conceitos relacionados a programação de forma visual e interativa, aliado ao suporte de uma comunidade que possibilite a colaboração e cooperação. Criado em 2007, o Scratch vem apresentando crescente popularidade, consolidando sua ferramenta e comunidade, apresentando mais de meio milhão de usuários ativos mensalmente, como pode ser visualizado na Figura 1 (SCRATCH, 2020).

Dessa forma, diversos projetos utilizam-se da ferramenta Scratch para explorar conceitos e práticas computacionais em salas de aula, buscando estimular o desenvolvimento do PC (DE FRANÇA; AMARAL, 2013). Na cidade de Pelotas/RS, o projeto Clubes de Computação Criativa é aplicado desde 2015, através de uma parceria en-

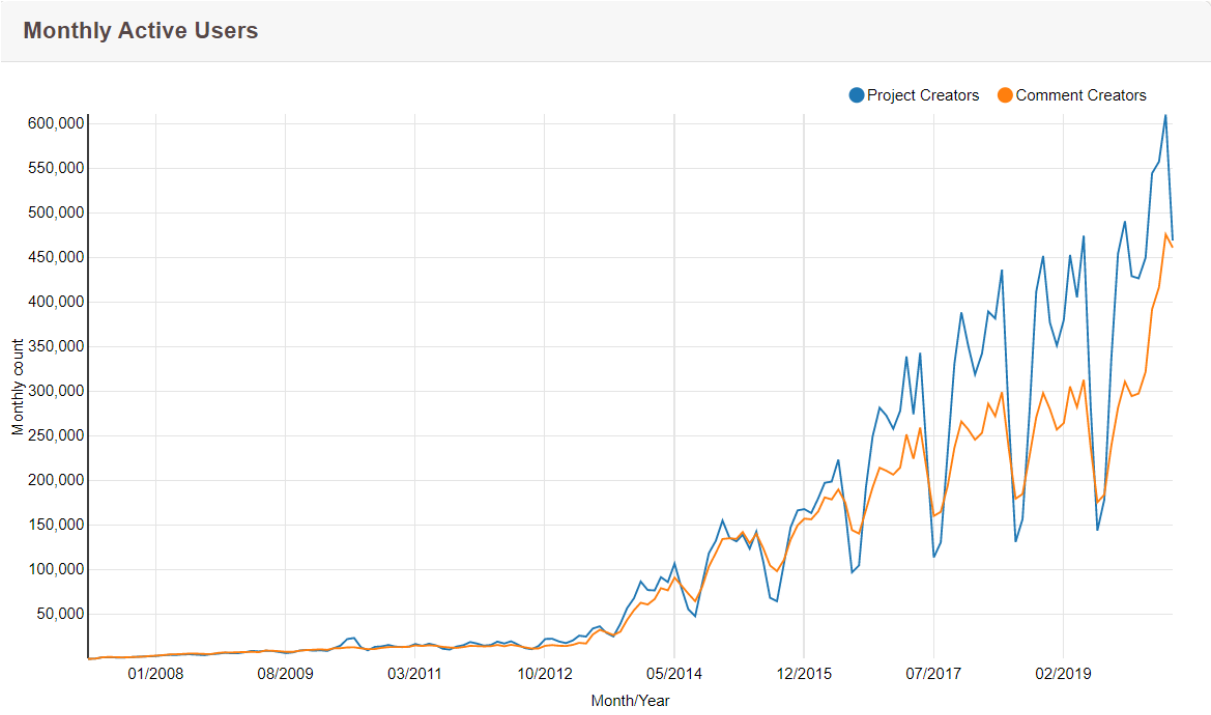


Figura 1 – Número de usuários ativos mensalmente na plataforma Scratch
Fonte: SCRATCH, 2020

tre a Universidade Federal de Pelotas (UFPel), a Secretaria Municipal de Educação (SMED) e o Núcleo Pelotas da Rede Brasileira de Aprendizagem Criativa. Através de atividades práticas alinhadas ao currículo da escola e à Base Nacional Comum Curricular (BNCC), seu principal objetivo é o estímulo ao desenvolvimento do PC em escolas públicas da cidade.

Neste cenário, é necessária a geração de evidências para demonstrar o impacto da realização destas atividades. Assim, surge o desafio de armazenar os dados de atividades, além de como utilizá-los para avaliar o progresso de um aluno, ou seja, como visualizar sua evolução. Visando atacar este problema, diversas metodologias de avaliação foram criadas, porém, em sua grande maioria, aplicadas de forma manual e pouco prática (BRENNAN; RESNICK, 2012; WILSON; HAINEY; CONNOLLY, 2012; SEITER; FOREMAN, 2013).

Neste sentido, a aplicação Dr. Scratch (MORENO-LEÓN; ROBLES; ROMÁN-GONZÁLEZ, 2015) foi proposta como forma de auxiliar a avaliação do projeto resultante de uma atividade. Esta ferramenta realiza uma análise automatizada, pontuando o projeto com base da decomposição da nota em 7 conceitos relacionados, como “Pensamento Lógico” e “Abstração”. Desta forma, a ferramenta permite, através do *upload* do projeto, a realização de uma avaliação de um projeto rapidamente, provando-se uma valiosa ferramenta de apoio (OLUK; KORKMAZ, 2016).

Entretanto, esta avaliação refere-se a apenas um ponto no tempo, ou seja, ao

resultado final do projeto. Ela não permite a avaliação do progresso e da evolução do aluno em cada um de seus conceitos. Além disso, surge como obstáculo a ausência de uma representação de conhecimento que permita armazenar estes dados de forma a permitir que estes sejam trabalhados e gerem evidências sobre o trabalho realizado.

Esta representação de conhecimento possibilitaria a realização de análises, raciocínios e inferências sobre os dados coletados, permitindo a detecção de padrões e a tomada de ação, como a adaptação de atividades com base nas dificuldades dos alunos (CAMPOS; SOSTER; BLIKSTEIN, 2019). Esta demanda fica evidente através de trabalhos como (TROIANO et al., 2019) e (SEITER; FOREMAN, 2013), onde são realizadas análises sobre o conhecimento construído no ensino do pensamento computacional, porém as mesmas tem seu escopo limitado por utilizarem o arquivo padrão de projeto, sendo limitadas devido a ausência de um modelo de dados mais abrangente, com capacidade de representação temporal.

Além disso, existem iniciativas que visam a coleta destes dados, como (JUNIOR et al., 2019) e (TROIANO et al., 2019), porém sem a formalização destes dados coletados em um formato padronizado e de fácil inferência para trabalhos futuros. Assim, se faz necessária a organização dos dados coletados, criando sua devida representação de conhecimento de modo a possibilitar a execução de análises temporais e a utilização de modelos de inferência.

1.1 Objetivos

O principal objetivo desta dissertação é prover uma representação de conhecimento capaz de caracterizar os dados de um projeto Scratch e de avaliações de habilidades do pensamento computacional demonstradas nele, permitindo o acompanhamento do aluno e dando suporte para a utilização de mecanismos inteligentes, como ferramentas de inferência.

Como objetivos específicos, destacam-se: (1) a criação de uma representação de conhecimento para projetos Scratch, capaz de dar suporte à realização de inferências, coletando e descrevendo as interações realizadas nos projetos; (2) a criação de uma representação de conhecimento para as avaliações de pensamento computacional de um projeto Scratch, permitindo a formalização e registro destas avaliações; (3) propor um modelo que seja aberto à comunidade e que facilite a realização de novos trabalhos sobre estes dados.

Os objetivos apresentados foram atingidos com a concepção do OntoScratch, uma representação de conhecimento baseada em ontologias para a representação de projetos Scratch e avaliações de habilidades do pensamento computacional ao longo do tempo. Baseada em duas ontologias modulares, esta representação de conhecimento é facilmente reutilizável e expansível, além de permitir a realização de inferências so-

bre estes dados.

1.2 Organização do Trabalho

No Capítulo 2 é apresentado o embasamento teórico utilizado para o presente trabalho, incluindo os trabalhos relacionados encontrados por meio da revisão bibliográfica. O Capítulo 3 contempla o escopo e o desenvolvimento da proposta concebida por esta dissertação. No Capítulo 4, são apresentados os resultados de testes de inferência realizados com as ontologias desenvolvidas. Por fim, no Capítulo 5, é abordada uma discussão sobre o trabalho e os resultados obtidos com a representação de conhecimento, apresentando também os possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo busca apresentar a fundamentação teórica do presente trabalho. Na Seção 2.1 deste capítulo será apresentada a definição de pensamento computacional, junto com outros conceitos correlacionados importantes para compreensão do problema abordado e motivação, como a educação *maker* e métodos para a avaliação de pensamento computacional. Já na Seção 2.2, será abordado o embasamento teórico de modelos de estudante, destacando as principais abordagens e cenários de uso. Na Seção 2.3, será apresentada a fundamentação teórica de ontologias, apresentando suas principais características, vantagens e desafios. Por fim, na Seção 2.5, será realizada uma conclusão sobre os pontos apresentados nestes capítulo, buscando unir a fundamentação teórica com o problema apresentado.

2.1 Pensamento Computacional

O pensamento computacional (PC) consiste em uma metodologia para resolução de problemas, baseando-se nos conceitos fundamentais da ciência da computação. Apesar da utilização da computação na educação datar da década de 60, com o uso da linguagem Logo para o ensino de matemática, a evolução da área e formalização do termo pensamento computacional ocorreu somente décadas mais tarde (FEURZEIG; PAPERT; LAWLER, 2011). Em 2006 Jeannette Wing realizou uma análise do conjunto de habilidades desenvolvidas por profissionais da computação para resolução de problemas e, com o resultado, propôs e formalizou o conceito da forma que é atualmente utilizado (WING, 2006).

Como complementado por (BLIKSTEIN, 2008), o PC não se trata de saber utilizar o computador, navegar na internet e enviar e-mails, mas sim utilizar o computador como uma ferramenta para aumento do poder cognitivo, da produtividade, inventividade e criatividade, aplicando-o em nossas capacidades intelectuais. Isto se dá pois a resolução de problemas, base da Ciência da Computação, também pode ser aplicada em qualquer área como a Matemática, a Química ou a Sociologia (NUNES, 2010).

A partir do trabalho de (WING, 2006), a exploração da utilização do conjunto de

habilidades contempladas pelo pensamento computacional para a resolução de problemas não apenas na área de computação ganhou força. Em 2009, a *Computer Science Teachers Association* (CSTA), propôs um conjunto de materiais de apoio e referência para a exploração do desenvolvimento do PC na educação básica (ISTE, 2011). O crescimento da área e o sucesso das iniciativas resultantes é notório, gerando diversos trabalhos que visam levantar discussões para a implementação de uma matéria curricular obrigatória primariamente focada em computação em diversos países, como os Estados Unidos, Holanda e Inglaterra (YADAV et al., 2017).

Junto a estas iniciativas, ocorre também o crescimento das linguagens de programação visuais como Scratch, Toontalk, Stagecast Creator e Alice (LYE; KOH, 2014). Diversas revisões sistemáticas recentes da literatura acerca do PC apontam a dominância deste tipo de ferramentas para o desenvolvimento do pensamento computacional, porém com a emergência de alternativas como *game design* e os jogos digitais (BOMBASAR et al., 2015; ZANETTI; BORGES; RICARTE, 2016; BORDINI et al., 2017; AVILA et al., 2017). Estas ferramentas tornam-se a abordagem mais comum para o estímulo ao desenvolvimento do pensamento computacional, através de atividades práticas, fazendo uso da teoria de ensino Educação *Maker*.

2.1.1 Educação *Maker*

A educação *maker* é uma teoria de ensino originária da conceituação do Construcionismo (PAPERT, 1991). Esta teoria defende que um projeto gerará mais impacto na vida de um aluno caso este aluno esteja engajado e se identifique com o que está fazendo e é muito utilizada em atividades focadas no ensino do PC.

Em (MAYER, 1989) são citadas três características fundamentais para o processo de ensino criativo:

- Permitir a compreensão de conceitos abrangentes através de uma fragmentação dos mesmos em habilidades pontuais que contribuam para adquirir conhecimentos importantes para o objetivo final dos alunos.
- O processo de construção do conhecimento é tão importante quanto o resultado final e, por isso, deve ser focado.
- O ensino da aprendizagem criativa deve ser realizado com atividades que contribuam para áreas específicas, como por exemplo um projeto que envolva o desenvolvimento artístico do aluno, ao invés de um projeto com tarefas genéricas.

Dentre estas características, é importante ressaltar o enfoque da educação *maker* no processo de aprendizado. Este é citado como parte até mesmo mais importante do que o resultado final de um projeto. Diretamente ligado a estas características e

ideias, (RESNICK, 2014) propõe quatro fundamentos buscando incentivar o trabalho colaborativo em projetos:

- *Projects*: pessoas trabalham melhor quando estão inseridas em projetos dos quais entendam seu objetivo e sua importância.
- *Peers*: o compartilhamento de experiências e ideias entre pessoas é fundamental para a criação e o aprendizado.
- *Passion*: projetos com os quais os participantes possuam interesse proporcionam melhores resultados.
- *Play*: ambientes de aprendizado devem estimular e permitir que as pessoas tentem e experimentem novas possibilidades.

Em paralelo, diversas áreas utilizam os conceitos da educação *maker* com diversos recursos para realizar o ensino e o desenvolvimento da criatividade dos alunos. Dentre estas, destaca-se a computação criativa. Ela utiliza o computador como meio de incentivo a criatividade, buscando estreitar os laços entre pessoas e computador através do estimulo a imaginação, criatividade e dos interesses próprios de cada indivíduo (BRENNAN; CHUNG; HAWSON, 2011).

Esta área busca utilizar a computação para, através da criatividade dos indivíduos, solucionar problemas. Assim, a computação criativa não busca formar profissionais da área de computação, como engenheiros e cientistas, mas sim proporcionar aos estudantes a oportunidade de se integrarem ao mundo tecnológico, tornando-se desenvolvedores de suas ideias (BRENNAN; CHUNG; HAWSON, 2011).

Para o ensino e estímulo da computação criativa, existem diversas ferramentas utilizadas, dentre elas destaca-se o Scratch (RESNICK et al., 2009). Esta ferramenta de programação visual permite a alunos desenvolverem projetos diversos, buscando proporcionar um ambiente onde possam experimentar e compartilhar experiências. Esta ferramenta é discutida em maiores detalhes na Subseção 2.1.2.

2.1.2 Scratch

O Scratch[1] é uma linguagem de programação visual (COX, 2007) gratuita criada pelo MIT Media Lab em 2007. A ferramenta pode ser utilizada através do navegador ou em sua versão *offline* para *desktop*. Através de uma comunidade, a ferramenta busca incentivar a interação entre seus usuários, estimulando o compartilhamento de projetos e a troca de experiências.

Apesar de ser focado em crianças entre 8 (oito) e 16 (dezesseis) anos, o Scratch possui uma comunidade com usuários de todas as idades (RESNICK et al., 2009).

[1] http://scratch.mit.edu/

Por tratar-se de uma linguagem de programação visual, os usuários interagem com blocos para construir seus projetos, sem necessitar realmente escrever código. Desta forma, o Scratch demonstra ser uma excelente ferramenta de ensino para usuários inexperientes de programação, pois sua interface simples e intuitiva facilita a entrada neste universo.

Assim, por seu ambiente criativo e sua proposta construtiva e colaborativa, o Scratch apresenta alta popularidade entre propostas de utilização da educação *maker* e do ensino de pensamento computacional (YADAV et al., 2017).

2.1.3 Avaliação de Pensamento Computacional

Com a aplicação de atividades de estímulo ao pensamento computacional, é necessária a realização de avaliações em relação ao progresso dos alunos e do impacto destas atividades. Para isto, primeiramente é fundamental compreender quais habilidades e conceitos o compõem e o que estes representam. A definição dos conceitos formados desta composição, além de como estes devem ser abordados no ensino, especialmente para crianças, é uma questão que não possui uma resposta clara e exata (BARR; STEPHENSON, 2011). Esta questão é extensamente debatida, com diversas diferentes propostas de divisão.

Para (DENNING; MARTELL, 2015), a ciência da computação é a ciência composta por mecânicas – como comunicação, coordenação e automação –, princípios de design – como performance, confiabilidade e segurança – e práticas – como programação, modelagem e validação. Baseando-se em uma definição similar, a *Association for Computing Machinery* (ACM) propôs e desenvolveu uma definição específica de computação para o ensino voltado a crianças, onde a área é definida como “o estudo de computadores e processos algoritmos, incluindo seus princípios, seu hardware e software, suas aplicações e seu impacto na sociedade” (TUCKER, 2003). Posteriormente, trabalhos mais recentes questionam esta definição, argumentando que os conceitos base do PC também estão presentes em diversas outras áreas e, portanto, as definições dos mesmos não devem ser tão focadas na ciência da computação (FELLEISEN; KRISHNAMURTHI, 2009; HEMMENDINGER, 2010).

Assim, com estes pontos em mente, o *Computer Science Teachers Association* (CSTA) e o *International Society for Technology in Education* (ISTE) reuniram 26 especialistas no ensino de PC com o objetivo de criar uma definição operacional de pensamento computacional e dos conceitos que o compõem, focando principalmente no seu ensino para crianças (CSTA, 2012). Como resultado, chegou-se em uma definição de PC que busca tornar-se mais interdisciplinar e mais independente da Ciência da Computação. Desta forma, foram definidos sete habilidades considerados como base para o PC e sua representação para diferentes áreas, como Matemática, Ciências Sociais e Ciências Linguísticas. O resultado deste mapeamento pode ser observado na

Tabela 1 – Conceitos base do pensamento computacional

Conceito	Ciência da Computação	Matemática	Ciência	Ciências Sociais	Ciências Linguísticas
Abstração	Utilizar procedimentos para encapsular um conjunto de comandos repetidos. Utilizar condicionais, loops e recursão.	Utilizar variáveis e funções em álgebra	Construir um modelo de uma entidade física	Sintetizar fatos e deduzir conclusões a partir deles	Utilizar uma metáfora ou analogia. Escrever uma história com diferentes fluxos.
Algoritmos e Procedimentos	Estudar algoritmos clássicos e implementá-los para um problema	Realizar uma longa divisão, fatorar um número	Realizar um procedimento experimental		Escrever instruções
Análise de Dados	Desenvolver um programa para realizar uma análise estatística básica sobre um conjunto de dados	Contar as ocorrências de eventos, como número de rolagens de dados e analisar os resultados	Analisar dados de um experimento	Identifica tendências em dados estatísticos	Identificar padrões para sentenças de diferentes tipos
Automação		Utilizar ferramentas ou trechos de código para automatizar o trabalho	Utilizar a ferramenta probeware	Utilizar o excel	Utilizar o corretor gramatical
Coleta de Dados	Buscar uma fonte de dados para uma problema da área	Buscar uma fonte de dados para um problema da área, como por exemplo, atirar moedas ou dados	Coletar dados de um experimento	Estudar estatísticas de uma população ou batalha	Realizar a análise linguística de sentenças
Decomposição de Problemas	Definir objetos e métodos	Aplicar operações ordenadas em uma expressão	Classificar uma espécie		Escrever um esboço
Paralelismo	Utilização de threads ou processos para dividir o processamento de dados ou de uma tarefa	Resolução de sistemas lineares ou multiplicação de matrizes	Simultaneamente executar diversos experimentos com diferentes parâmetros		
Representação de Dados	Utilizar estruturas de dados como listas, filas, pilhas, grafos e etc.	Utilizar histogramas, gráficos de barras, gráficos de pizza e etc. para representar dados. Utilizar conjuntos, listas e grafos para representar dados.	Sintetizar/resumir dados de um experimento	Sintetizar e representar tendências	Representar diferentes padrões de diferentes tipos de sentença
Simulação	Utilizar algoritmos para animação	Representar graficamente uma função em um plano cartesiano e modificar os valores de suas variáveis	Simular a movimentação do sistema solar	Jogar Age of Empires ou Oregon Trail	Encenar uma história

Adaptado de BARR; STEPHENSON, 2011

Tabela 1.

Algumas iniciativas buscaram propor metodologias para realização da avaliação e acompanhamento da evolução do aprendizado de conceitos do PC junto a ferramentas de programação visual como o Scratch. Trabalhos como (BRENNAN; RESNICK, 2012) e (WILSON; HAINEY; CONNOLLY, 2012), propuseram sólidos *frameworks* para execução desta avaliação, decompondo-a em uma série de conceitos similares. No entanto, em ambos os trabalhos, o enfoque era em fornecer informações que dessem suporte ao professor para a realização desta avaliação, sem contemplar uma ferramenta que a executasse. A dificuldade de convergência em relação aos conceitos que compõem o pensamento computacional foi um fator complicador para o desenvolvimento de ferramentas que realizassem a avaliação do desenvolvimento de um estudante na área (BRENNAN; RESNICK, 2012; GROVER; PEA, 2013).

Em (BOE et al., 2013), é proposto o *Hairball* uma ferramenta que realiza a análise estática de códigos de projeto Scratch, procurando detectar potenciais problemas e padrões errados de código, como trechos duplicados e blocos inalcançáveis. Apesar de realizar a análise de forma automatizada, a ferramenta não trazia para a avaliação os conceitos de PC.

Assim, em (MORENO-LEÓN; ROBLES; ROMÁN-GONZÁLEZ, 2015), foi proposta uma ferramenta web focada em realizar a avaliação de conceitos de PC demonstrados em um projeto Scratch, de forma a fornecer métricas e quantificadores sobre o trabalho realizado. Esta ferramenta é disponibilizada como gratuita e de código aberto, tendo sido amplamente aceita e evoluída (OLUK; KORKMAZ, 2016; TROIANO et al., 2019; JUNIOR et al., 2019).

Para a execução da avaliação, a ferramenta avalia sete conceitos do pensamento computacional: abstração e decomposição de problemas; paralelismo; pensamento lógico; sincronização; controle de fluxo; interatividade com usuário; e representação de dados. Para cada um destes, é atribuída uma nota de 0 a 3, representando, respectivamente, a não demonstração do conceito, baixa proficiência, proficiência em desenvolvimento e domínio do conceito. Estas notas são então somadas e seu valor atribuído como nota geral do nível de PC demonstrado no projeto.

Para execução desta avaliação, a ferramenta utiliza-se de regras para identificação de padrões de código no projeto, seguindo uma adaptação da metodologia apresentada em (SEITER; FOREMAN, 2013). Estes padrões podem ser visualizados na Tabela 2.

2.2 Student Model

Um *Student Model* (SM) representa uma estrutura contendo informações diretas e indiretas sobre um estudante, como seus interesses, preferências, dados pessoais e suas habilidades (HOLT et al., 1994). Comumente, a construção do perfil é focada em um contexto, direcionando as informações armazenadas. Por exemplo, para um contexto de *e-learning*, o foco será dado para informações relativas a interação do aluno com o sistema – como navegação e tempo de conexão –, além do progresso de seu aprendizado (CHRYSAFIADI; VIRVOU, 2013).

A criação de um SM deve buscar capturar as principais características que permitam formar o perfil do aluno, buscando uma representação completa, coesa e coerente com o contexto estudado. Esta dependência do contexto faz com que as informações específicas armazenadas possuam ampla variação, porém sua semântica é comum mesmo em diferentes contextos. Assim, em (HAMIM; BENABBOU; SAEL, 2019a) é realizada uma revisão sistemática dos últimos 5 (cinco) anos de literatura sobre modelos de estudante e, com base nestas informações, é definida e apresentada uma

Tabela 2 – Regras de avaliação da ferramenta Dr. Scratch

Conceito	Nível de Competência			
	Não Demonstrado (0)	Básico (1 ponto)	Em Desenvolvimento (2 pontos)	Proeficiente (3 pontos)
Abstração e Decomposição de Problemas	-	Duas ou mais *sprites* com dois ou mais fluxos de execução cada	Definição de procedimentos	Utilização do recurso de clones
Controle de Fluxo	-	Sequenciamento de blocos	Uso da estrutura "Repeat" ou "Forever"	Uso de estrutura "RepeatUntil"
Interatividade com Usuário	-	Uso do bloco de captura de evento "GreenFlag"	Captura de eventos de pressionamento de tecla ou clique em *sprite* ou blocos de *input* de teclado ou mouse	Bloco de evento para comparação de variáveis ou uso de blocos de áudio e vídeo
Paralelismo	-	Duas sequências de fluxo com bloco de evento "GreenFlag"	Dois blocos de evento de tecla pressiona ou dois blocos de evento para clique em *sprite*	Dois blocos de recebimento de mensagens ou dois blocos de comparação de variáveis ou dois blocos de evento de troca de pano de fundo ou bloco de criação de clone
Pensamento Lógico	-	Uso de bloco "If"	Uso de bloco "IfElse"	Uso de operadores lógicos
Representação de Dados	-	Utilização de algum bloco modificador de propriedades da *sprite*	Utilização de blocos de operações em variáveis	Uso de blocos de operações em listas
Sincronização	-	Uso do bloco "Wait"	Uso do bloco "Broadcast" ou "WhenIReceiveMessage" ou "StopAll" ou "StopProgram" ou "Stop ProgramSprite"	Uso do bloco "WaitUntil" ou "WhenBackdropChangeTo" ou "BroadcastAndWait"

Adaptado de MORENO-LEÓN; ROBLES; ROMÁN-GONZÁLEZ, 2015

taxonomia com as características mais relevantes e comumente utilizadas, buscando unificar a nomenclatura e criar uma base para novos SMs (Figura 2). No total, a taxonomia define 10 (dez) diferentes categorias de características:

- Identidade Pessoal: corresponde a informações que permitam definir o estudante como único em relação a outros, como gênero, sexo, nacionalidade, nome, ocupação e e-mail.
- Identidade Social: contempla informações derivadas to pertencimento a um grupo social como estado civil, nível educacional dos pais, religião e cultura.
- Condição Física: refere-se ao condicionamento físico e as habilidades de um estudante, como a habilidade auditiva, visual e verbal.
- Acadêmica: composta pelas informações realizadas ao desempenho e histórico acadêmico do aluno em diversos níveis como suas notas, diplomas e desempenho.

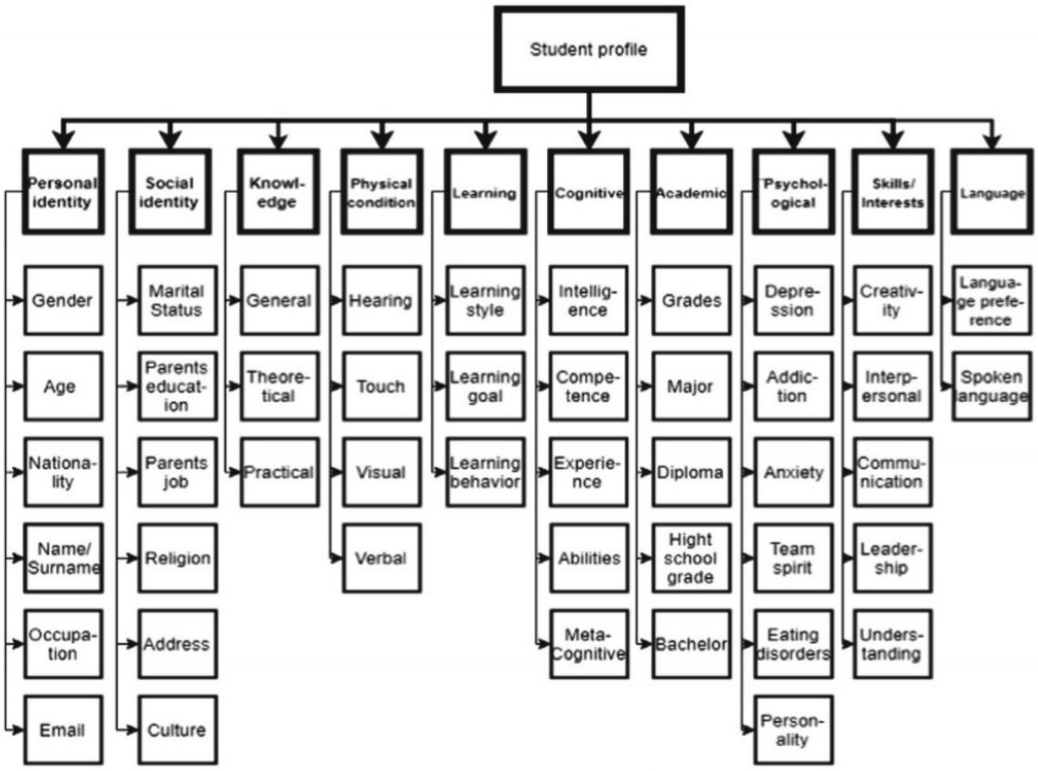


Figura 2 – Taxonomia para características do estudante
Fonte: HAMIM; BENABBOU; SAEL, 2019a

- Aprendizado: refere-se as informações relacionadas ao aprendizado de um aluno. Características comuns dessa categoria são o estilo de aprendizado, objetivo do aluno e seu perfil comportamental.
- Cognitiva: corresponde ao escopo de processos referentes a manipulação de informação. Comumente contempla aspectos como inteligência, experiência e competência.
- Conhecimento: refere-se ao conhecimento adquirido pelo aluno, podendo este ser através da atividade ou conhecimento prévio. Características comuns dessa categoria são os conhecimentos sobre os conteúdos, podendo estes serem gerais ou mais especializados dependendo do contexto de uso do modelo.
- Psicológica: refere-se a personalidade e saúde mental do estudante, armazenando informações como ansiedade, personalidade, espírito de equipe e possíveis condições de saúde, como depressão.
- Habilidades/Interesses: contempla os talentos e interesses que estimulam o aluno, como a criatividade, habilidade interpessoal, comunicação e liderança.

- Linguagem: refere-se a linguagem preferencial do aluno, além de linguagens adicionais e seu nível de competências nestas.

Para modelagem destas informações, diversas abordagens são utilizadas. Dentre estas, destacam-se na literatura as seguintes: *Overlay*, *Stereotype*, *Perturbation*, *Fuzzy*, Redes Bayesianas e Ontologias. Estas diferentes abordagens serão aprofundadas nas subseções seguintes, buscando exemplificar os pontos positivos e negativos de cada uma.

2.2.1 Overlay

Uma das abordagens mais clássicas e populares, o modelo *overlay* foi inventado por (STANSFIELD; CARR; GOLDSTEIN, 1976) e permaneceu como a técnica mais utilizada até a ascensão da inteligência artificial após 2008 (CHRYSAFIADI; VIRVOU, 2013). Este modelo assume que o estudante possui um conhecimento incompleto do domínio em questão, considerando sempre que este conhecimento é correto. Ou seja, o conhecimento do estudante é um subconjunto do conhecimento total deste domínio (HOLT et al., 1994).

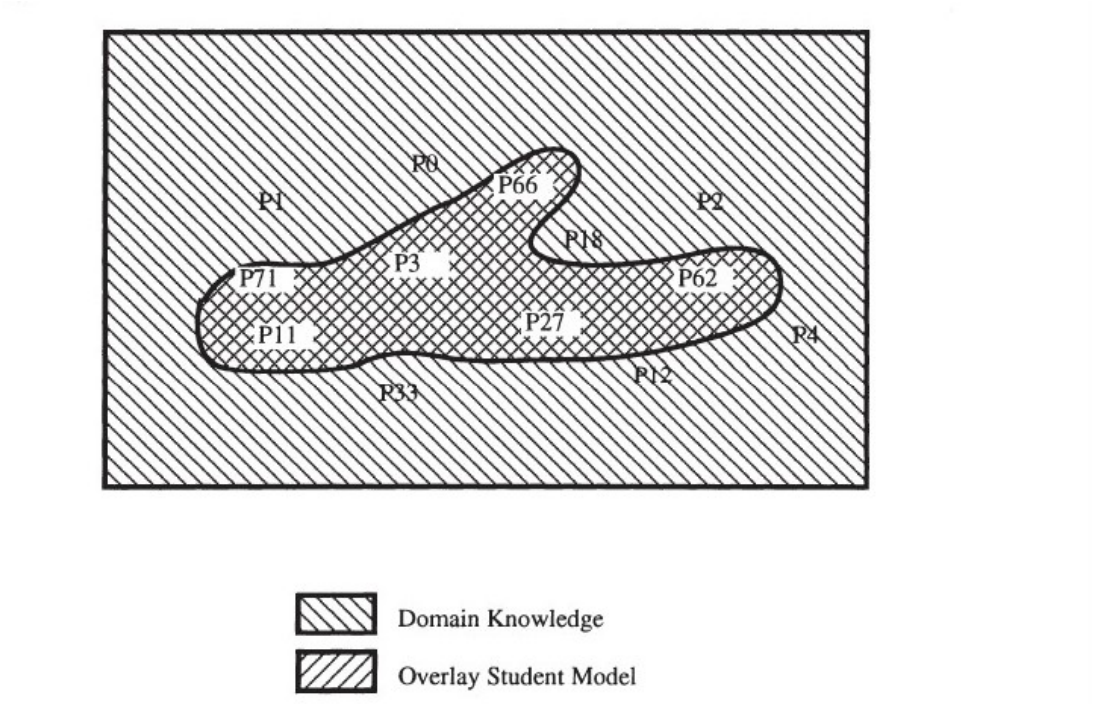


Figura 3 – Um modelo de estudante utilizando a abordagem *overlay*
Fonte: HOLT et al., 1994

O conhecimento total do domínio é considerado como o conhecimento de um especialista e a diferença entre ele e o conhecimento do estudante se dá, segundo o modelo, pela falta de ensino do aluno. Assim, o modelo busca diminuir a diferença entre esses dois conjuntos o máximo possível, decompondo o conhecimento de domínio

em conjuntos menores para identificar os pontos que devem ser apresentados ao estudante. Em virtude desta decomposição, a qualidade e especificidade deste modelo depende do nível de granularidade utilizado para decomposição e da qualidade do algoritmo de estimativa de conhecimento utilizado para o estudante (MARTINS et al., 2008).

O modelo de *overlay* puro utiliza-se de valores booleanos para sinalizar se o estudante possui ou não o conhecimento em questão, porém existem derivações do mesmo que especificam medidas qualitativas (como bom, médio e ruim) ou um valor numérico, utilizando-se de conceitos de lógica fuzzy (CHRYSAFIADI; VIRVOU, 2013).

Devido a estrutura utilizada para representação do conhecimento, este modelo não permite a representação de conhecimentos errados adquiridos pelo estudante ou a diferenciação entre as preferências do aluno, sua personalidade e suas características cognitivas específicas. Assim, este modelo não é adequado para a representação de modelos sofisticados, pois não leva em consideração a forma que um estudante adquiri novos conhecimentos e como esse conhecimento se integra e relaciona com seus conhecimentos prévios (RIVERS, 1989). Por estas razões, diversos trabalhos optam por uma abordagem híbrida ao adotarem o modelo de *overlay*, combinado-o com outras estratégias como *stereotypes*, *perturbation* e *fuzzy*.

2.2.2 Perturbation

A abordagem chamada de *perturbation* ou *buggy* é uma extensão do modelo de *overlay*, adicionando a ele a possibilidade de representação de conhecimentos errados de um estudante. Nestes modelos, o conhecimento do estudante é representado por um subconjunto do conhecimento total do domínio com a adição de concepções erradas sobre o conteúdo que possua (NGUYEN; DO, 2008). Esta representação permite uma maior capacidade de reação e correção por parte dos sistemas que o utilizem, pois estes podem buscar esclarecer e reapresentar conteúdos com base nas concepções erradas de um aluno (CHRYSAFIADI; VIRVOU, 2013).

O conjunto de erros e concepções errôneas mapeadas e representadas em um modelo *perturbation* é comumente chamado de *bug library*. Esta coleção pode ser construída pela técnica enumerativa, gerando-os de forma empírica través da análise de erros, ou pela técnica generativa, construindo os erros a partir de um conjunto de regras e erros comuns.

A técnica enumerativa, também conhecida como "*buggy*" por ter sido proposta no sistema homônimo (BROWN; BURTON, 1975), analisa o modelo para determinar quais erros possíveis e quais são comuns de ocorrerem. Esta técnica, apesar de popular, apresenta problemas de desempenho e não representa todos os erros possíveis, podendo ocorrer de alguns estudantes cometerem erros não classificados (CLANCEY, 1988). Já a técnica generativa utiliza um modelo cognitivo para detectar

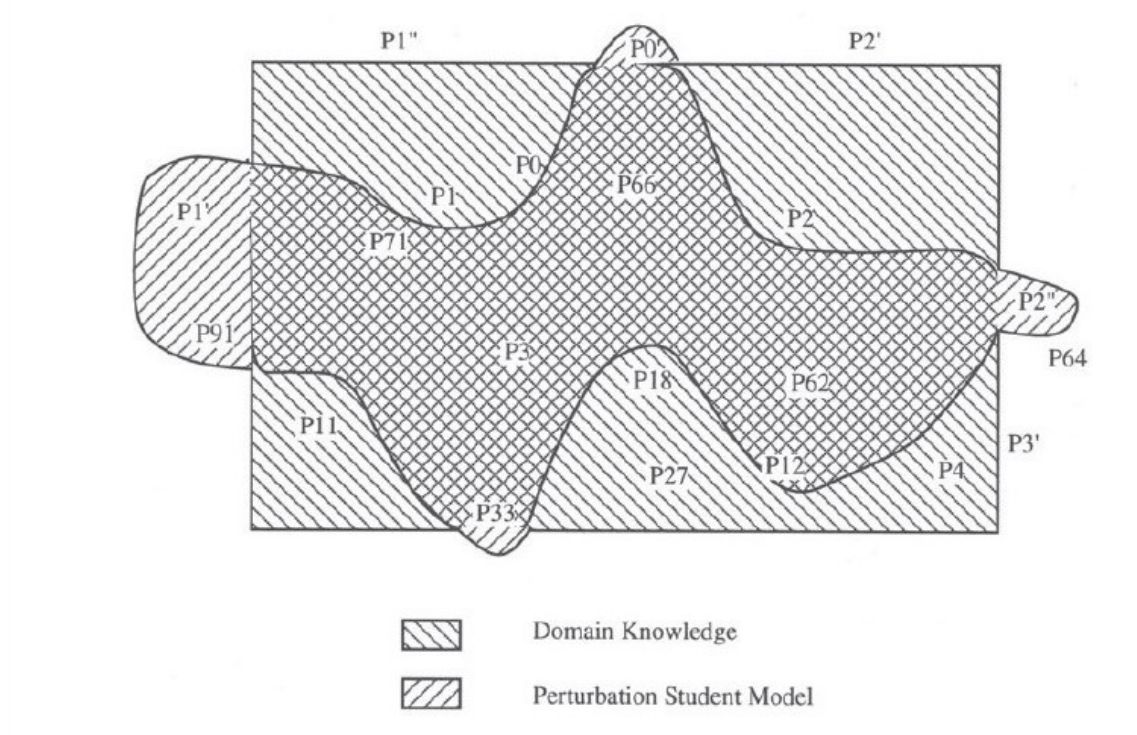


Figura 4 – Um modelo de estudante utilizando a abordagem *perturbation*
Fonte: HOLT et al., 1994

os erros realizados por um estudante e classifica-os como extrapolações erradas do conhecimento apresentado (MAYO, 2001).

Este modelo é muito utilizado em hibridizações para auxiliar na racionalização e análise do comportamento dos estudantes. Através do mapeamento de seus erros e conceitos errados, o mesmo produz um conhecimento adicional para o entendimento destes comportamentos, sendo de grande auxílio para ferramentas adaptativas.

2.2.3 Stereotype

Outra abordagem muito utilizada é a chamada de *stereotypes*, introduzida por (RICH, 1979) no sistema chamado de GRUNDY. Esta abordagem define grupos chamados de estereótipos, onde a diferenciação entre eles é realizada por características bem definidas que determinam a classificação do usuário para cada grupo.

Para isso, cada grupo possui uma condição de gatilho, que é utilizada para determinar a classificação positiva, e uma condição de retração, utilizada para determinar a remoção da classificação (KAY, 2000). Assim, caso um indivíduo cumpra a condição de gatilho, o mesmo será classificado com este estereótipo, porém, ao cumprir a condição de retração, este é removido do grupo.

Este tipo de modelagem é limitada por sua falta de flexibilidade, pois depende dos estereótipos e classificações pré-definidas. Porém, possui amplo uso em combinação com outras técnicas, especialmente para combater o problema de inicialização de

outros algoritmos, conhecido como *cold start* (TSIRIGA; VIRVOU, 2002). Assim, este modelo pode ser utilizado para realizar a inicialização dos estudantes em grupos pré-definidos, utilizando esta classificação enquanto um segundo modelo coleta dados para realizar novas classificações.

2.2.4 Fuzzy

O aprendizado de um aluno é um processo complexo, difícil de ser representado de forma concreta. Determinar o conhecimento adquirido por um aluno neste processo não é algo simples, visto que isto é diretamente ligado a outros fatores que nem sempre são observados ou modelados (JEREMIĆ; JOVANOVIĆ; GAŠEVIĆ, 2012).

Uma possível abordagem para lidar com esta incerteza é a utilização de lógica fuzzy. Esta metodologia é capaz de lidar com a incerteza causada por dados imprecisos ou incompletos, além da subjetividade humana (DRIGAS; ARGYRI; VRETTAROS, 2009).

Estudos demonstram que a utilização de técnicas de lógica fuzzy para a recomendação e personalização de ambientes virtuais de aprendizado resulta em uma melhora em sua performance, resultando em uma melhoria de adaptatividade e maior satisfação dos usuários (CHRYSAFIADI; VIRVOU, 2012). Como demonstrado em (STATHACOPOULOU et al., 2005), o uso destas técnicas permite que o sistema reproduza, de forma similar, a avaliação realizada por professores em relação às características do aluno, auxiliando na compreensão de seu perfil pedagógico.

Por estas razões, diversos trabalhos utilizam-se destas para a construção de um *student model* híbrido, combinado-as com outras técnicas para facilitar a modelagem de incertezas, como o nível de conhecimento de um aluno.

2.2.5 Ontologias

Ontologias nos permitem definir um vocabulário comum para compartilhar informações sobre um domínio (NOY; MCGUINNESS et al., 2001). Elas são formadas por conceitos que formam este domínio e suas relações, expressas de forma que estas sejam compreensíveis por máquinas.

Esta abordagem permite a representação de conceitos e propriedades de uma forma que estas possam ser reutilizadas e, se necessário, expandidas. Além disso, sua base formal, além da sustentação teórica, também permite a realização de raciocínios sobre a informação representada (CLEMENTE; RAMÍREZ; DE ANTONIO, 2011).

(WINTER; BROOKS; GREER, 2005) analisou as principais vantagens do uso deste tipo de abordagem para a criação de um *student model*, destacando sua semântica formal, facilidade de expansão e reuso, além de grande suporte para raciocínios e ferramentas de inferências. Por estes fatores, ontologias estão se tornando uma das

abordagens mais comuns para a criação de um SM, comumente utilizadas em uma abordagem híbrida, ficando responsáveis pela estrutura do modelo (HAMIM; BENABBOU; SAEL, 2019b).

Esta abordagem será discutida em maiores detalhes na Seção 2.3.

2.2.6 Redes Bayesianas

Uma rede bayesiana é um grafo direcionado acíclico onde os nodos representam variáveis, enquanto as arestas que os ligam representam a dependência probabilística ou relacionamento causal entre eles (PEARL, 2014). Para a criação de um *student model* com redes bayesianas, os nodos podem representar diferentes dimensões de um estudante, como seu conhecimento, suas emoções e sua motivação.

Através desta estrutura, redes bayesianas permitem a representação de incertezas, além de proverem uma sólida base teórica (LIU, 2006). Sua alta representatividade unida ao fato de serem facilmente expressas de forma gráfica e da existência de ferramentas de suporte estáveis facilita sua adoção, tornando-as uma das abordagens dominantes para a criação de um SM (CHRYSAFIADI; VIRVOU, 2013).

Para a criação de um *student model* baseado em redes bayesianas, é de fundamental importância a definição de uma boa arquitetura para a rede e de bons valores para dependência entre eles. (MAYO, 2001) classifica-as em três categorias em relação à metodologia utilizada para construção:

- *Expert-centric*: estas redes utilizam o conhecimento de especializas para definir a estrutura da rede, suas relações e seus valores de dependência.
- *Efficiency-centric*: nesta categoria encaixam-se os modelos que buscam restringir a estrutura das redes para maximizar a sua eficiência computacional.
- *Data-centric*: estas redes utilizam-se de dados de experimentações ou testes anteriores para definir a estrutura e os valores de dependência.

Assim, para criação de um SM utilizando redes de estudantes, um dos principais desafios é a necessidade de consulta a um especialista ou da realização de experimentos, buscando definir a melhor estrutura da rede e de valores para as dependências.

2.3 Ontologias

Ontologias são formas de modelar, formalmente, a estrutura de um sistema, contemplando as entidades relevantes e as relações entre elas. Em termos mais técnicos, uma ontologia é um artefato de engenharia, constituído por um vocabulário específico para descrever um determinado contexto, aliado a um conjunto de premissas em

relação ao significado pretendido para as palavras que compõem este vocabulário (GUARINO, 1998).

Sua criação se dá através da observação das entidades relevantes e consequente organização destas em conceitos e relacionamentos, representados, respectivamente, através de predicados de lógica de primeira ordem unários e binários (FENSEL, 2001). O cerne da estrutura de uma ontologia, consiste em uma hierarquia de conceitos representada através de generalizações e especializações, ou seja, uma taxonomia (GUARINO; OBERLE; STAAB, 2009). Desta forma, estes conceitos podem ser conectados através de relacionamentos, aos quais podem somar-se axiomas para expressar relações mais complexas ou adicionar restrições em relação a seu significado.

Assim, para auxiliar na construção destes vocabulários, diversas metodologias foram desenvolvidas (IQBAL et al., 2013). Porém, em sua base, muitas utilizam-se dos mesmos conceitos, pois o processo de construção – ou engenharia – de uma ontologia deve ser genérico e independente de domínio, de forma a dar suporte para a construção da representação de conhecimento independente do contexto ou da especificidade desejada (GUARINO; WELTY, 2004).

Com esta proposta, (GUARINO; WELTY, 2002) propôs a *OntoClean*, uma metodologia para engenharia de ontologias independente de domínio, que busca auxiliar na criação do vocabulário através de orientações baseadas em fundamentos da área filosófica ontologia. Através destes conceitos, a metodologia define meta propriedades que são utilizadas para caracterizar o significado pretendido com cada propriedade, classe, conceito e relacionamento que compõe a ontologia.

Ela baseia-se em quatro meta propriedades que, ao serem atribuídas à entidades de uma ontologia, permitem descrever melhor o seu comportamento esperado, além de permitir analisar a estrutura e a natureza dos conceitos apresentados. São elas:

- **Essência** diz respeito a se uma propriedade é obrigatória verdadeira para que uma entidade seja classificada como ela é. Por exemplo, a propriedade de *ser duro* é essencial para um martelo, pois todo martelo precisa ser duro para ser considerado um martelo.

- Já a meta propriedade de **rigidez** classifica se uma propriedade é essencial para todas as suas instâncias. Utilizando do mesmo exemplo, a propriedade de *ser duro* não é rígida, pois a mesma é essencial para instâncias de martelo, porém não é para esponjas. Um exemplo de rigidez, é a propriedade *ser uma pessoa*, pois todas as entidades que podem ser uma pessoa, devem sê-la, ou seja, ela é verdadeira para todas as instância que podem possuí-la. No entanto, para a classe que representa esponjas, esta propriedade não é essencial, pois apesar de algumas esponjas serem duras, nem todas elas possuem esta propriedade, tornando-a não essencial.

- A meta propriedade **identidade** refere-se a questão de como classificar entidades como iguais ou diferentes. Assim, a identidade de uma entidade é definida através de um critério, utilizado na comparação para responder a esta pergunta de equivalência. Por exemplo, supondo duas classes chamadas de *duração de tempo*, com instâncias como “uma hora” e “duas horas”, e *intervalo de tempo*, referindo-se a intervalos específicos como “13:30-15:10 Quarta-Feira”, é possível utilizar o critério de identidade para avaliar se a classe *intervalo de tempo* é uma subclasse de *duração de tempo*, visto que um intervalo de tempo representa também uma duração de tempo. Neste caso, a identidade para *duração de tempo* se dá através de seu tamanho, onde duas durações de mesmo tamanho podem ser consideradas idênticas. No entanto, para *intervalo de tempo*, não é possível afirmar o mesmo. Para dois intervalos serem considerados equivalentes, é necessário que ambos possuam a mesma duração e se refiram ao mesmo momento de tempo. Assim, modelar a classe de intervalo como uma subclasse de duração não seria uma boa decisão de representação, visto que esta modelagem resultaria em permitir que uma instância da superclasse possuísse duas instâncias da sub-classe de diferente semântica, como dois intervalos de mesma duração, mas em dias diferentes.

- **Unidade** é a meta propriedade que refere-se a possibilidade de identificar as partes que compõem uma entidade, permitindo discernir entre o que representa uma parte de um objeto e em que condições é possível considerá-lo completo. Por exemplo, para a classe *água*, suas instâncias não representam objetos completos, pois as mesmas representam uma certa quantidade de água, não sendo possível distingui-las como uma entidade isolada. Em contrapartida, uma instância da classe *oceano* representa um objeto completo, pois é possível identifica-la como uma entidade isolada, como “o Oceano Atlântico”.

Para (VAN HEIJST; SCHREIBER; WIELINGA, 1997; STUDER; BENJAMINS; FENSEL, 1998; GUARINO, 1998; BORST, 1999), ontologias podem ser classificadas em quatro categorias de acordo com seu nível de especificidade:

- Ontologias de Alto Nível: descrevem conceitos gerais que não dependem de um problema ou domínio específico, como espaço físico ou tempo, os quais são independentes de um domínio ou problema particular.

- Ontologias de Domínio: descrevem o vocabulário relacionado a um domínio genérico, como medicina ou automobilismo, especializando os termos introduzidos pela ontologia de alto nível.

- Ontologias de Atividade: descrevem um vocabulário relacionado a uma tarefa

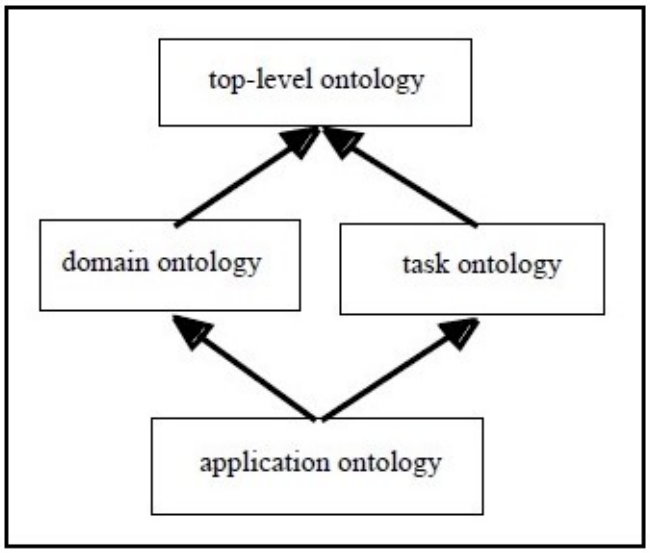


Figura 5 – Tipos de ontologia de acordo com seu nível de especificação
Fonte: GUARINO, 1998

ou atividade genérica, como vender ou comprar, a partir da especialização dos termos da ontologia de alto nível.

- Ontologias de Aplicação: descrevem conceitos que dependem de uma ontologia de domínio e de uma ontologia de atividade, ou seja, descrevem conceitos que correspondem a função que uma entidade de domínio exerce enquanto realiza uma determinada atividade, como, por exemplo o conceito de “peça de reposição”.

2.3.1 Linguagens de Criação e Manipulação de Ontologias

O *Resource Description Framework* (RDF) é a linguagem recomendada pelo World Wide Web Consortium (W3C) como padrão para a definição e uso de metadados de recursos na Web. Ela define um modelo de dados para a descrição de recursos de forma que estes sejam compreensíveis e processáveis por máquinas (HORROCKS; MCGUINNESS; WELTY, 2003).

Segundo (BROEKSTRA et al., 2001), o RDF baseia-se em três tipos de objetos:

- Recursos: um recurso pode representar uma página web completa, uma coleção de páginas ou um recurso não acessível via Web, como um livro impresso.
- Propriedades: uma propriedade é uma característica ou uma relação utilizada parra descrever um recurso.
- Declarações: uma declaração em RDF consiste em um recurso específico junto com uma propriedade nomeada e o valor desta propriedade para o recurso em questão.

Estas três partes são chamadas, individualmente, de sujeito, predicado e objeto. Comumente esta tripla é descrita por $P(S, O)$, representando uma relação chamada P entre os nodos S e O. O RDF também permite uma forma de reificação, onde uma declaração RDF pode ser o sujeito ou o objeto de uma tripla, permitindo o encadeamento e aninhamento de grafos (BROEKSTRA; KAMPMAN; VAN HARMELEN, 2002).

Por sua vez, o *RDF Schema* (RDFS), é um mecanismo que permite a definição de vocabulários específicos para dados RDF, como novos predicados (BROEKSTRA et al., 2001). Isto é permitido através da pré-definição de alguns termos como *Class*, *subClassOf* e *Property*, que podem ser utilizados para definir o vocabulário específico da aplicação. Expressões em RDFS são expressões que também são válidas em RDF, porém com a diferenciação de que no RDFS é realizado um acordo em relação a semântica de certos termos e em relação a consequente interpretação de certas declarações (BROEKSTRA; KAMPMAN; VAN HARMELEN, 2002).

Entretanto, o RDFS não prevê semântica para a definição de primitivas e a expressividades destas é limitada e insuficiente para a realização de modelagens ontológicas completas e para a utilização de mecanismos de inferência e raciocínio (BROEKSTRA et al., 2001). Para esta finalidade, a W3C propôs a linguagem *Web Ontology Language* (OWL), focada em ser utilizada para descrição de conteúdos de forma compreensível para máquina e com maior capacidade de expressividade e interpretação do que o XML, o RDF e o RDFS (MCGUINNESS; VAN HARMELEN et al., 2004).

Recomendada pela W3C desde 2004, a linguagem OWL foi amplamente aceita e aplicada em diversos problemas da ciência da computação (GRAU et al., 2008). Ela se diferencia por permitir a descrição da estrutura de um domínio em notação similar a orientação a objetos, com seu domínio sendo descrito por classes e propriedades, permitindo a representação de relacionamentos entre classes, cardinalidade e características de propriedades, dentre outras (PERNAS; DANTAS, 2004).

A linguagem OWL é dividida em 3 sub-linguagens, distintas pelo nível de formalismo exigido e pela liberdade dada ao usuário na definição de ontologias (MCGUINNESS; VAN HARMELEN et al., 2004):

- *OWL Lite*: a versão mais enxuta, oferece suporte para usuários que estão em busca de uma hierarquia de classificação e a definição de restrições simples.
- *OWL DL*: oferece suporte para usuários que desejam maior poder de expressividade, mantendo a características de ser computacionalmente completa – ou seja, de que todas as suas conclusão são garantidamente computáveis.
- *OWL Full*: focada para usuários que buscam a maior expressividade possível, essa versão permite a alteração do significado de propriedades pré-definidas do RDF e do OWL, porém ao custo de não garantir ser computacionalmente completa – ou seja, sem garantias de que todas suas conclusões são computáveis.

Com a recomendação da recomendação do uso do RDF pela W3C, o problema natural de consultar dados em RDF tornou-se pauta para discussão, com diversas soluções sendo propostas (HAASE et al., 2004). Dentre estas, destaca-se a linguagem SPARQL Protocol And RDF Query Language (SPARQL).

Essencialmente, o SPARQL é uma linguagem de consulta para busca por padrões em grafos, com sua consulta consistindo de três partes (PÉREZ; ARENAS; GUTIERREZ, 2009). A primeira, representa o padrão de grafo buscando, possuindo operadores como o aninhamento de padrões, especificação de padrões opcionais ou até mesmo a filtragem de valores. Já a segunda parte, realiza a modificação da solução, permitindo a modificação dos valores retornados após o filtro da primeira parte, com operadores operadores para diversas operações como a projeção, ordenação e limitação de valores. Por fim, a terceira parte representa a consequência desta consulta, permitindo a seleção de valores, a resposta de perguntas booleanas ou até mesmo a construção de novas triplas.

O SPARQL tornou-se a linguagem recomendada com padrão pela W3C para consultas a triplas e ontologias, além de ser amplamente utilizada tanto da comunidade aberta, quanto de empresas (BIZER; SCHULTZ, 2009).

2.4 Trabalhos Relacionados

Nesta seção serão apresentados os trabalhos relacionados relevantes, buscando contextualizar sua abordagem e seus objetivos para auxiliar na comparação com a proposta do presente trabalho.

Diversos trabalhos exploram a questão de avaliar o desenvolvimento do pensamento computacional através de projetos Scratch. Dentre as ferramentas e metodologias encontradas, a mais popular apresenta ser o Dr. Scratch (MORENO-LEÓN; ROBLES; ROMÁN-GONZÁLEZ, 2015). Esta ferramenta gratuita e de código aberto realiza a avaliação do PC de um projeto automaticamente, pontuando-o em sete diferentes aspectos: abstração e decomposição de problemas; paralelismo; pensamento lógico; sincronização; controle de fluxo; interatividade com usuário; e representação de dados.

Para cada um destes conceitos é atribuída uma nota de 0 a 3 representando, respectivamente: conceito não demonstrado, básico, conceito em desenvolvimento e conceito proficiente. Estas notas são somadas e o valor total atribuído como nota geral do projeto.

Esta ferramenta tem como público alvo não apenas os professores, mas também os próprios alunos. Assim, ela permite que os alunos realizem o envio do arquivo de seu projeto no formato *sb2*, padrão da ferramenta Scratch, em uma interface web amigável (Figura 6). Um grande enfoque é dado no formato para apresentação dos

resultados, buscando apresentá-los de forma a não desestimular o usuário em casos negativos.

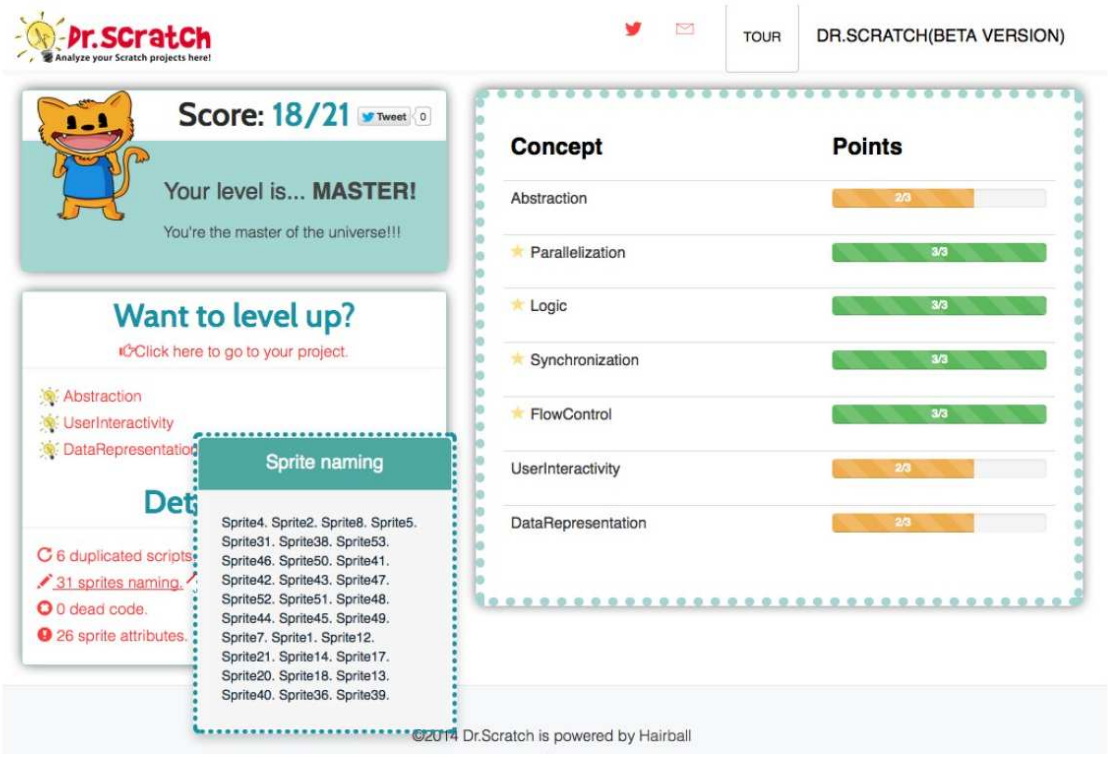


Figura 6 – Avaliação de um projeto no Dr. Scratch
Fonte: MORENO-LEÓN; ROBLES; ROMÁN-GONZÁLEZ, 2015

A execução desta avaliação se dá através de regras que buscam identificar padrões de programação nos projetos. Apesar de sua simplicidade, este método apresenta bons resultados, extensivamente explorados pela literatura. Em (OLUK; KORKMAZ, 2016), é realizado o acompanhamento de 31 (trinta e um) estudantes ao longo de 6 (seis) semanas de aulas, mantendo registro de suas avaliações pelo Dr. Scratch e de avaliações manuais do nível de pensamento computacional. Estes dados demonstraram uma forte correlação entre as pontuações, onde um aumento na pontuação do Dr. Scratch representava um aumento também no nível de pensamento computacional dos alunos.

Já em (CHANG; TSAI; CHIN, 2017; DHARIWAL, 2019), são apresentadas ferramentas com enfoque no desenvolvimento de ferramentas visuais que permitam ao usuário analisar e descobrir padrões de programação em projetos Scratch. A proposta de ambos trabalhos contrasta com a avaliação realizada pelo Dr. Scratch, buscando apresentar padrões visuais e de programação que auxiliem na avaliação ao invés de quantificar as habilidades do aluno.

Porém, a grande maioria dos trabalhos considera apenas o ponto final para realizar a avaliação do projeto. Ou seja, estas avaliações são realizadas sobre o resultado da

atividade, porém não levam em consideração a evolução do aluno até aquele ponto.

Em (TROIANO et al., 2019), a análise é direcionada para a evolução do nível de PC do aluno ao longo do tempo, não adotando mais a visão de apenas um ponto estático final. Para a análise, 317 estudantes foram instruídos a desenvolver jogos com temáticas educativas ao longo de 35 aulas. Durante as atividades, eram realizados *snapshots* dos projetos a cada minuto, salvando o estado atual do projeto, representado pelo arquivo *sb2* naquele momento. Destes projetos, 100 tiveram de ser descartados devido a possuírem arquivos corrompidos ou terem sido abandonados pelos estudantes durante as aulas. Assim, o *dataset* final era composto por 217 projetos Scratch. Para cada projeto, foi utilizada a ferramenta Dr. Scratch para avaliar o nível de PC demonstrado em cada um dos estados. Ou seja, a avaliação foi realizada para cada estado do projeto com intervalos de 1 minuto, criando assim a evolução dos alunos nos níveis de PC ao longo do desenvolvimento do projeto.

A adição do fator de temporalidade à avaliação permitiu ampliar a gama de análises realizadas. Através destes dados, foi possível identificar dificuldades e facilidades de aprendizado dos alunos. Por exemplo, estes demonstravam maior dificuldade para desenvolver o conceito de abstração, enquanto apresentavam maior facilidade para desenvolver pensamento lógico. Além disso, foi realizada a clusterização dos alunos com base em suas avaliações de cada uma das 7 habilidades ao longo do tempo, permitindo identificar alunos com perfis de aprendizado e níveis de conhecimento similares.

Entretanto, este trabalho, assim como a grande maioria dos trabalhos analisados, continua a operar sobre o arquivo *sb2* da ferramenta Scratch, representando um ponto fixo no tempo. Para contornar este problema, o trabalho (TROIANO et al., 2019) utilizou-se de fotografias durante o desenvolvimento, porém sem fugir do formato padrão do arquivo, sem considerar uma representação de conhecimento para estes dados que servisse de base para novos trabalhos.

Para atacar este problema, em (JUNIOR et al., 2019), é proposto um coletor de ações para a ferramenta Scratch, permitindo a transmissão em tempo real das ações do aluno. Através deste coletor, objetiva-se permitir a análise completa da trajetória de aprendizado do aluno, pois, através da sequência de ações desempenhadas, é possível reconstruir o estado do projeto em qualquer momento de tempo.

Porém, apesar de realizar o importante passo de coleta de ações, o trabalho de (JUNIOR et al., 2019), assim como a grande maioria analisada, não realiza a proposta de um formato para representação do conhecimento de um projeto Scratch reconstruído ou mesmo para armazenamento das avaliações realizadas ao longo do tempo com a ferramenta Dr. Scratch.

Em (QI et al., 2020), é proposta a criação de um grafo de conhecimento para o domínio Scratch. Neste rede, busca-se representar a plataforma *online* da ferramenta,

representando os perfis dos usuários, suas interações – como comentários e pedidos de amizade – além de resumos gerais de seus projetos.

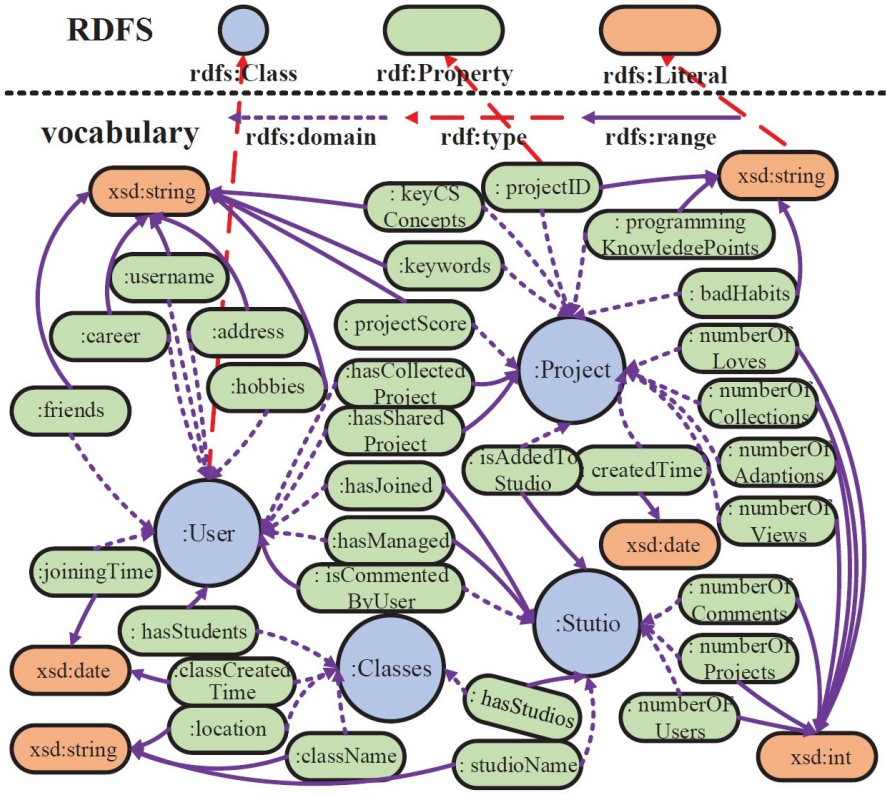


Figura 7 – Modelo de dados proposto para rede de conhecimento Scratch
Fonte: QI et al., 2020

Devido ao maior enfoque na plataforma e em suas interações, a representação dos projetos em si é simplificada, utilizando apenas dados sociais e métricas relacionadas a padrões de programação. Utilizando o Dr. Scratch, o trabalho representada a pontuação final de PC de um projeto e seus padrões de código detectados, como a presença de algumas más práticas de programação Figura 7. Desta forma, a representação do código em si é novamente ignorada, assim como o detalhamento da avaliação e a evolução do aprendizado de PC pelo aluno.

Já em (ARAÚJO; LIMA; HENRIQUES, 2019) é proposta uma ontologia para a representação do processo de aprendizado de pensamento computacional. Neste trabalho são propostas representações de conhecimento para os conceitos abordados pelo PC, assim como também a representação das abordagens utilizadas em seu ensino. Como resultado, este trabalho apresenta uma plataforma contendo recursos que auxiliam no ensino de PC, recomendando a atividade ideal a ser utilizada por um professor dependendo de qual conceito ele deseja abordar.

2.5 Considerações do Capítulo

Este capítulo apresentou a fundamentação teórica necessária para a realização da presente dissertação. O conceito de Pensamento Computacional foi abordado, junto a seus tópicos específicos relevantes, como a Educação *Maker*, a ferramenta *Scratch* e métodos para avaliação do PC. Ainda, o capítulo apresentou a definição de Student Model, apresentando diversas estratégias comumente utilizadas para descrever os mesmos, como *overlay*, *perturbation*, *stereotype*, *fuzzy*, Redes bayesianas e ontologias.

Em razão da escolha pelo uso de uma abordagem baseada em ontologias para o presente trabalho devido a seu potencial de reuso e de inferência, esta foi fortemente focada no capítulo.

Por fim, foram apresentados trabalhos relacionados a presente dissertação. Através destes trabalhos, é possível notar a ausência de trabalhos com enfoque na criação de um modelo para representação do conhecimento de um projeto Scratch e da evolução de um aluno no seu aprendizado de PC. É neste cenário que o presente trabalho insere sua principal contribuição, realizando a criação de uma representação de conhecimento capaz de reproduzir os trabalhos de um aluno e suas avaliações ao longo do tempo, servindo como base para análises e trabalhos futuros na área.

3 ONTOSCRATCH: CONCEPÇÃO E TECNOLOGIAS

Este capítulo descreve o desenvolvimento da representação de conhecimento para suporte ao ensino de pensamento computacional através de atividades Scratch. Esta modelagem foi concebida com enfoque em representar os projetos desenvolvidos e a avaliação de habilidades do pensamento computacional, de modo a permitir a realização do acompanhamento do progresso de alunos, provendo suporte a realização de inferências e a criação de ferramentas de suporte para o professor.

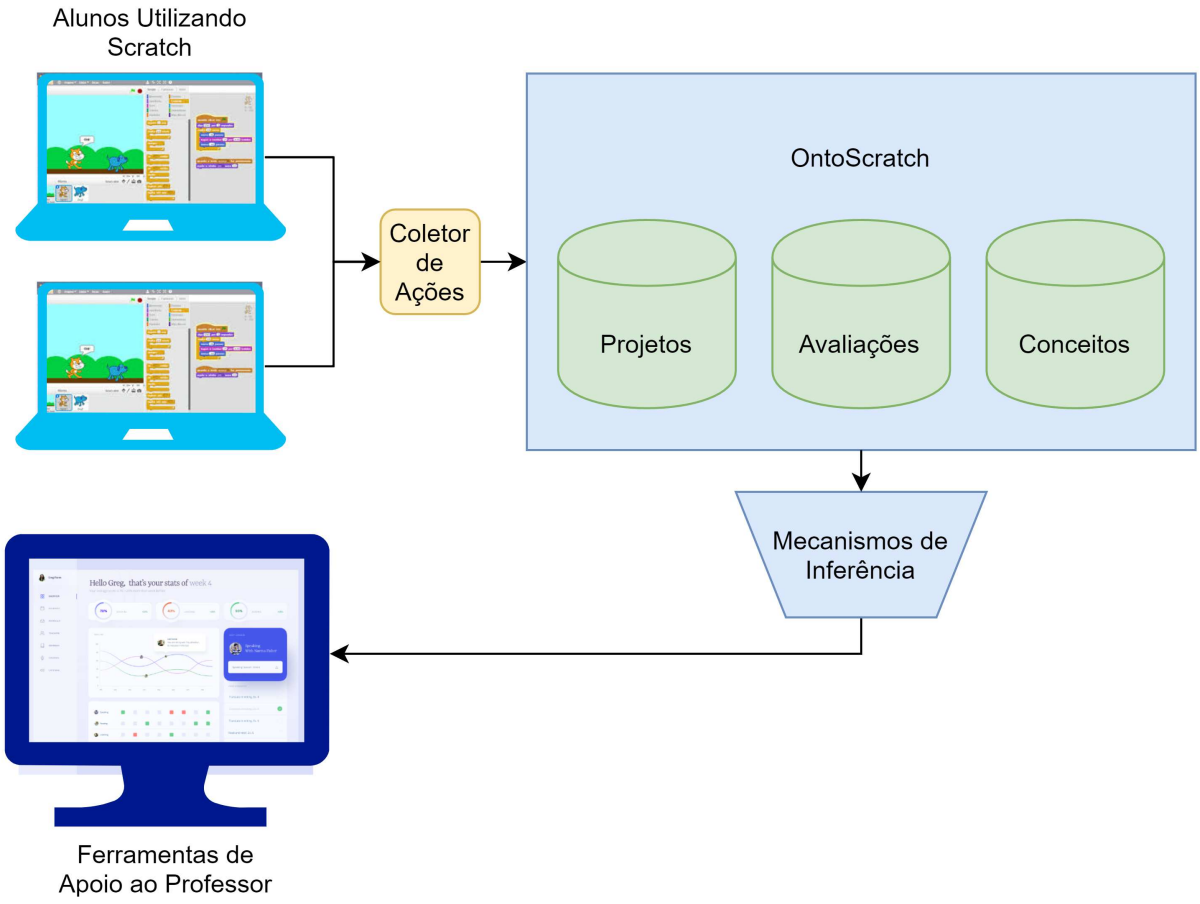


Figura 8 – Diagrama do Ecossistema

A concepção deste modelo representa a possibilidade de conexão do mesmo a

coletores de ações em tempo real, como apresentado em (JUNIOR et al., 2019). Estes coletores capturam as ações realizadas por alunos, como adição e remoção de blocos, em tempo real, permitindo a reconstrução do estado do projeto em qualquer momento de tempo.

Na Seção 3.1, é definido o escopo da presente dissertação, contextualizando o problema atacado e a estrutura da representação de conhecimento desenvolvida. Já nas seções 3.2 e 3.3, é apresentado o desenvolvimento desta representação, apresentando e justificando as escolhas realizadas.

3.1 Escopo

A presente pesquisa colabora com o projeto Clubes de Computação Criativa que vem sendo executado desde 2015 na cidade de Pelotas/RS por meio de uma parceria entre a Universidade Federal de Pelotas (UFPel), a Secretaria Municipal de Educação (SMED) e o Núcleo Pelotas da Rede Brasileira de Aprendizagem Criativa. Neste projeto, são priorizadas atividades mão na massa alinhadas ao currículo das escolas e à Base Nacional Comum Currícular (BNCC), priorizando a utilização de atividades e tecnologias de baixo custo.

Com base pedagógica na aplicação da Computação Criativa, é necessária a geração de evidências a partir dos dados coletados que demonstrem o impacto destas atividades nas escolas públicas de Pelotas. Neste contexto, o presente trabalho propõe-se em realizar a construção do modelo para armazenamento destes dados, de modo a fornecer o suporte necessário para acompanhamento dos alunos, além de possibilitar a construção de novas ferramentas de auxílio para o professor.

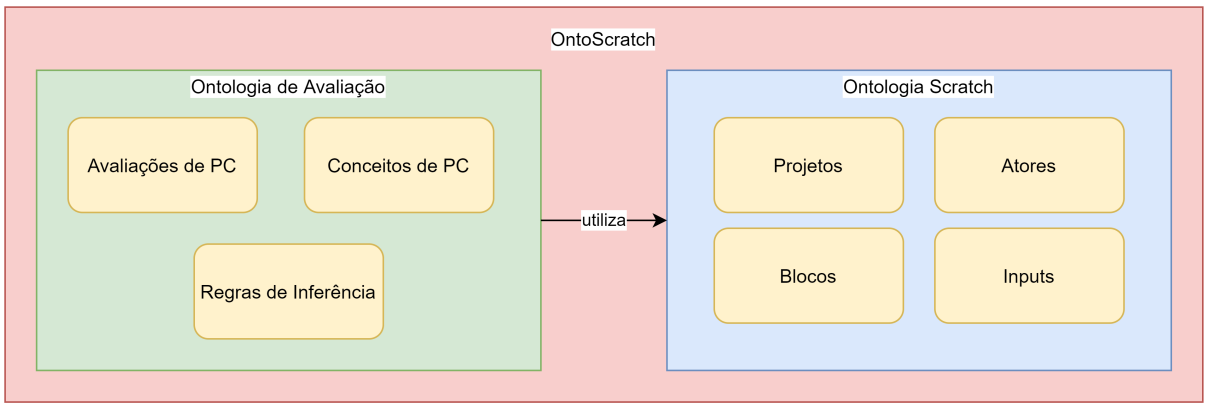


Figura 9 – Conjunto de Ontologias do OntoScratch

Assim, foi dado especial enfoque na representação de um projeto em sua totalidade, minimizando perdas semânticas e de representatividade, de modo a não limitar o escopo de análises e inferências futuras. Além disso, a respeito da representação

da avaliação sobre cada projeto, foi dedicada especial atenção para a possibilidade de acompanhamento da evolução do aluno ao longo da construção de um projeto, de modo a permitir o acompanhamento do caminho de aprendizado para cada conceito específico avaliado. Este fator permite que sejam realizadas análises sobre quais conceitos são mais facilmente desenvolvidos por cada aluno e em quais estes apresentam maior dificuldade.

Devido a importância do suporte a inferências e raciocínios por este modelo de dados, optou-se pela representação do mesmo através de um conjunto de ontologias (Figura 9). A divisão desta representação em ontologias distintas tem como principal motivação facilitar o reuso do vocabulário e sua futura expansão. Assim, a primeira ontologia é responsável por representar o universo de um projeto Scratch, com todos seus objetos e relações, sendo abordada em detalhes na Seção 3.2. Já a segunda ontologia, é responsável pela representação da avaliação de pensamento computacional sobre um projeto, suas regras e conceitos avaliados, sendo detalhada na Seção 3.3.

3.2 Representação do Universo Scratch

Apesar de sua simplicidade de uso, a representação do universo de um projeto Scratch é extensa e complexa. A ferramenta apresenta 119 (cento e dezenove) blocos únicos em sua terceira versão (SCRATCH WIKI, 2008), sem considerar as diversas extensões disponíveis que adicionam novos blocos. Estes podem ser combinados em diversos *scripts*, com diferentes sequências e parametrizações, aumentando ainda mais esta gama de possibilidades.

A documentação do Scratch está disponível em uma wiki[1]. Nela, estão disponíveis exemplos de uso, além da documentação dos elementos que compõem a ferramenta e suas extensões. Esta documentação serviu como dicionário de dados e base para a criação desta ontologia, sendo fonte de boa parte da nomenclatura utilizada. Esta opção se deu com o intuito de evitar conflitos entre classificações, buscando tornar mais simples a transição entre vocabulário da ferramenta e da ontologia.

Assim, criou-se uma classe base, chamada de *ScratchObject*, para representar todo e qualquer objeto pertencente ao universo virtual do Scratch. Subclasse de *Thing*, ela serve como base para todas as outras classes a serem desenvolvidas nesta ontologia.

Nas subseções a seguir serão abordadas as representações de conhecimento desenvolvidas para compor a OntoScratch [2], buscando apresentar e discutir as escolhas de modelagem realizadas

[1] https://en.scratch-wiki.info/
[2] https://github.com/Naraujo13/OntoScratch

3.2.1 Projeto

Base para o universo de criação, o projeto contempla toda a atividade desenvolvida pelo usuário. Nele, são armazenados os personagens e cenários, além dos comportamentos desenvolvidos e esperados para cada um deles. Assim, um projeto Scratch é modelado como uma classe específica, uma subclasse de *ScratchObject* e de *Project*, do popular vocabulário *Friend-of-a-Friend*[3] (FOAF), como pode ser visualizada na Figura 10.

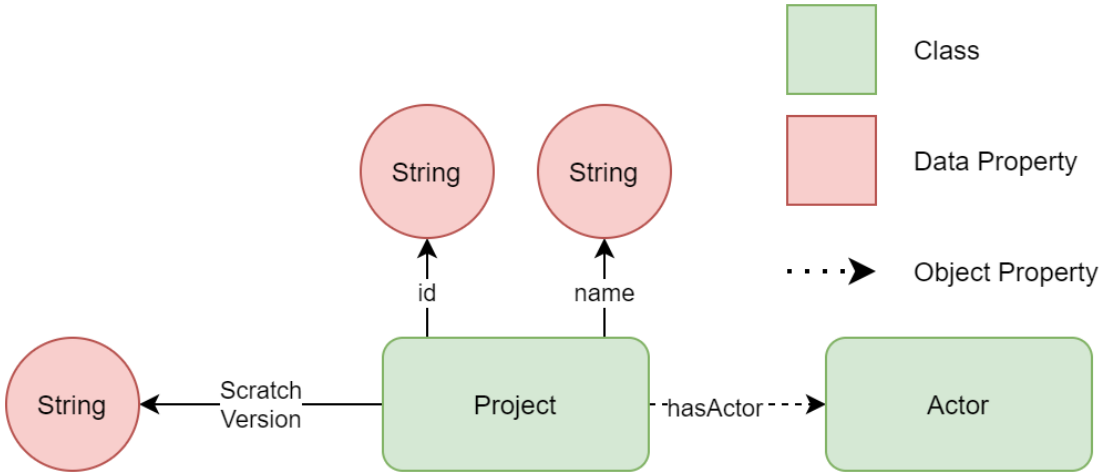


Figura 10 – Representação ontológica para um projeto Scratch

O nome do projeto e identificador do usuário criador são armazenados como atributos de dados do tipo *string*, além do identificador único internamente utilizado para o projeto e de um identificador da versão do Scratch utilizada. O armazenamento e diferenciação da versão da ferramenta utilizada é de fundamental importância, pois com a atualização da ferramenta entre diferentes versões, novas funcionalidades são adicionadas ou até mesmo modificadas, alterando o universo de possibilidades possibilitado pela ferramenta.

Assim, cada indivíduo desta classe representa uma atividade Scratch, possuindo, por sua vez, atores, que serão os responsáveis pelo comportamento esperado.

3.2.2 Atores

No universo Scratch, atores são aqueles que atuam em um projeto, ou seja, todos aqueles que são capazes de ter *scripts* de blocos para alterar o seu comportamento. Assim, os mesmos podem ser divididos em dois grupos: personagens (*Sprites*) e cenários (*Stages*).

Personagens representam atores que são capazes de movimentar-se, ou seja, de possuir *scripts* de comportamento que alterem sua posição. Cenários, por sua vez,

[3] http://xmlns.com/foaf/spec/

representam os palcos por onde os personagens se movimentam, podendo também possuir *scripts* de comportamento, porém são incapazes de se movimentar.

Assim, para representação na ontologia, optou-se pela criação de uma classe base *Actor*, subclasse de *ScratchObject* e disjunta de *Project*, para representar o grupo de atores em geral de um projeto. Além disso, a diferenciação entre personagens e palcos foi realizada através da criação de duas classes especializadas, *Sprite* e *Stage*, onde ambas são sub-classes de *Actor*.

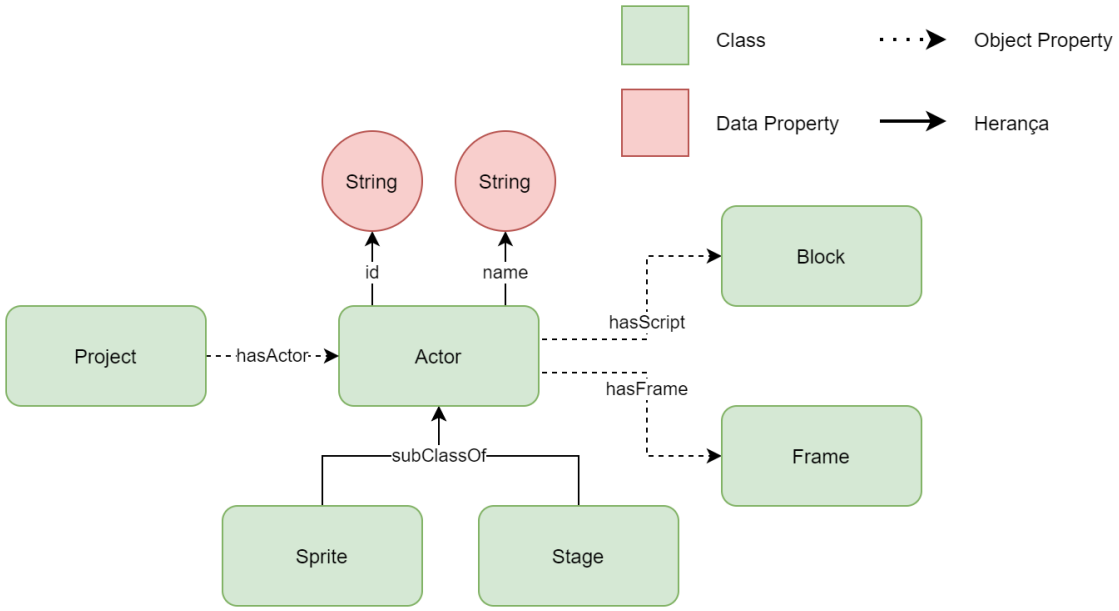


Figura 11 – Hierarquia de classes para atores

Todo ator, seja ele um personagem ou palco, possui alguns fatores em comum: (1) possui um nome atribuído, representado por um atributo de dados no formato de String, associado diretamente à classe base *Actor*; (2) todo ator possui diferentes representações visuais chamadas de *frames* e discutidas mais a fundo na Subseção 3.2.3; e, (3) todo ator possui *scripts*, sendo estes representados por blocos e aprofundados na seção Subseção 3.2.4. E, por fim, a conexão dos atores com seu projeto na ontologia se dá através da propriedade de objeto *hasActor*, cujo domínio é classe *Project* e imagem *Actor*, além de sua relação inversa chamada *actsIn*.

Esta hierarquia de classes, junto com suas respectivas propriedades de objeto e de dados, pode ser visualizada na Figura 11.

3.2.3 Frames

Em um projeto Scratch, cada ator pode possuir diversas formas de representação visual, sendo cada uma destas chamadas de *frame*, como exemplificado na Figura 12. Estes, assim como atores, podem ser divididos em dois grupos: (1) fantasias (*costumes*), quando estes representam diferentes representações visuais de um mesmo

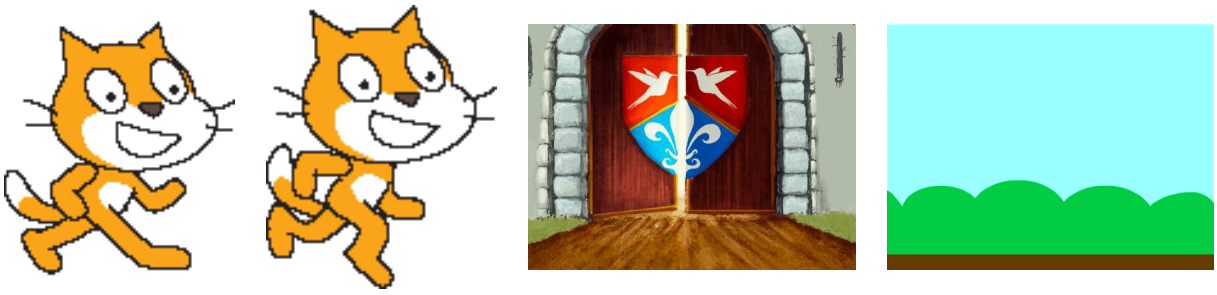


Figura 12 – Exemplos de *Frames* de personagens (*costumes*) e de *Frames* de palcos (*backdrops*)

personagem; (2) panos de fundo (*backdrops*), quando estes representam diferentes visualizações de um palco.

Deste modo, a classe *Frame* foi representada como uma subclasse de *ScratchObject* e disjunta de *Project* e *Actor*, representando uma forma visual de um ator. Ademais, foram criadas duas subclasses de *Frame*, *Costume* e *Backdrop*, sendo estas representações visuais específicas para *Sprites* e *Stages*, respectivamente.

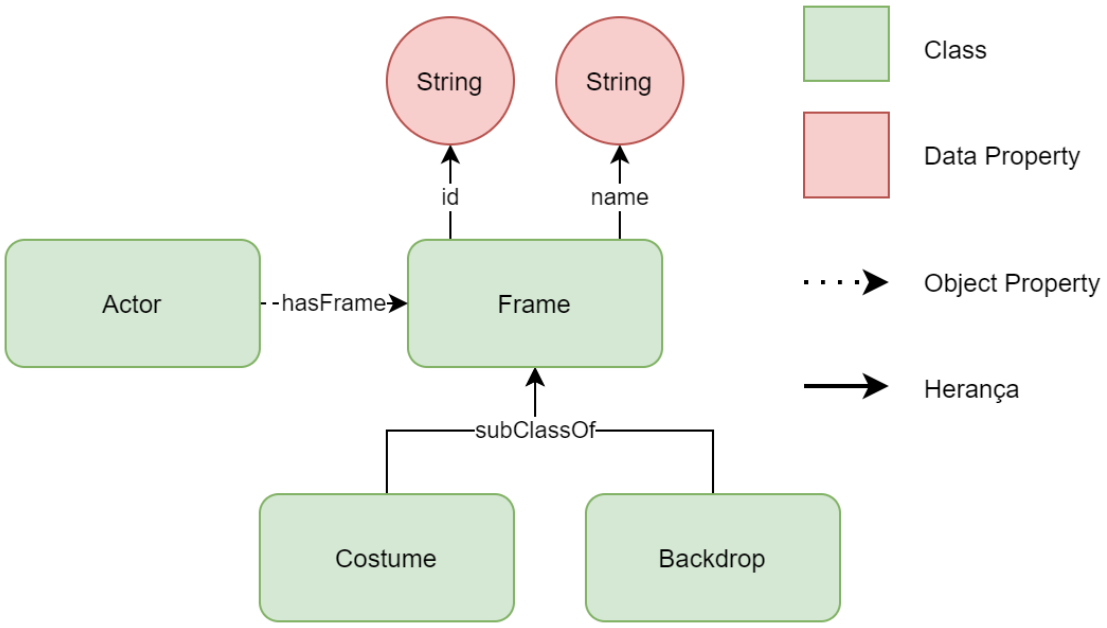


Figura 13 – Hierarquia de classes para frames

Assim como os atores, todo *frame*, seja ele uma fantasia ou um pano de fundo, possui um nome e um identificador associado, sendo estes representados por propriedades de dados. Já a relação entre o ator e seu *frame* foi modelada através de uma propriedade de objeto chamada *hasFrame*, com a propriedade inversa *frameIsFrom*.

Esta hierarquia de classes, junto com suas respectivas propriedades de objeto e de dados, pode ser visualizada na Figura 13.

3.2.4 Blocos

Blocos são estruturas visuais que são encaixadas no estilo de peças de quebra-cabeça para criar códigos e *scripts* de comportamento no Scratch. Os blocos são conectados verticalmente, onde este encaixe é determinado por cada um dos formatos do bloco. Este formato visual de programação facilita o aprendizado, visto que evita erros sintáticos e a necessidade de memorização de comandos, porém ao custo de flexibilidade, visto que blocos não podem ser facilmente editáveis.

É possível distinguir os blocos em duas hierarquias diferentes: em relação ao seu formato e em relação a sua categoria. A primeira é responsável por definir as relações de um bloco com outros, definidos se o mesmo pode ter sucessores, antecessores ou blocos internos. Já em relação a suas categorias, estas são definidas pela funcionalidade esperada.

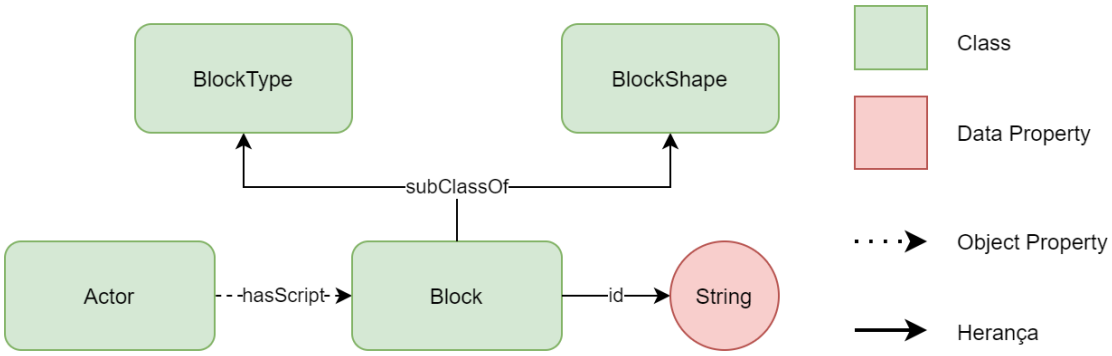
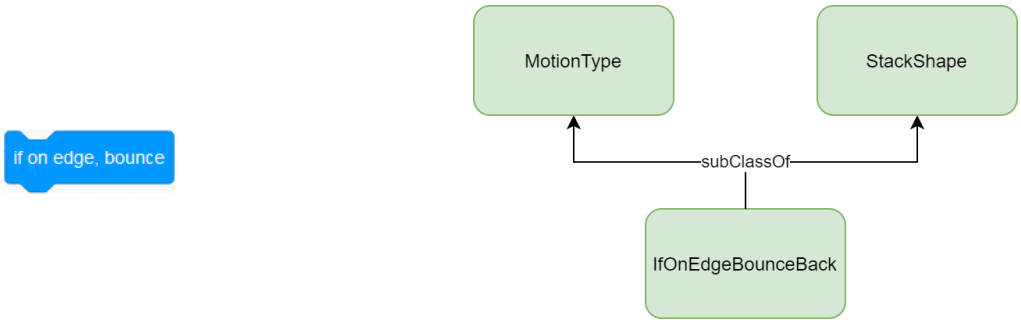


Figura 14 – Hierarquia de classes para bloco

Assim, a classe que representa um bloco Scratch sempre será uma subclasse de um formato e de uma categoria (Figura 14), de modo a definir sua função e seu relacionamento com outros blocos. Cada uma destas hierarquias erá abordada melhor nas subseções 3.2.4.1 e 3.2.4.2, respectivamente.

Um exemplo disto é para o bloco *IfOnEdgeBounceBack* (Figura 15), utilizado para fazer *Sprites* rebaterem ao chegarem nas bordas da tela, onde este é uma subclasse de uma categoria de bloco, neste caso *MotionType*, e de um formato de bloco, neste caso *StackShape*. Assim, esta classe representa este bloco, porém, no Scratch, o mesmo bloco pode ser utilizado como diversas instâncias diferentes. Assim, cada instância de um bloco é representada como um indivíduo do mesmo, diferenciada pelo seu identificador único e com diferentes relações de objeto. Este formato se estende para cada um dos 119 (cento e dezenove) blocos existentes no Scratch.

No universo Scratch, os atores, sejam eles *Stages* ou *Sprites*, possuem *scripts* de comportamento. Estes *scripts* consistem de uma pilha de blocos encaixados sequencialmente que altera o comportamento daquele ator específico. Assim, um ator pode possuir diversos fluxos de execução, representados por diferentes pilhas de blocos.



(a) Bloco

(b) Classe

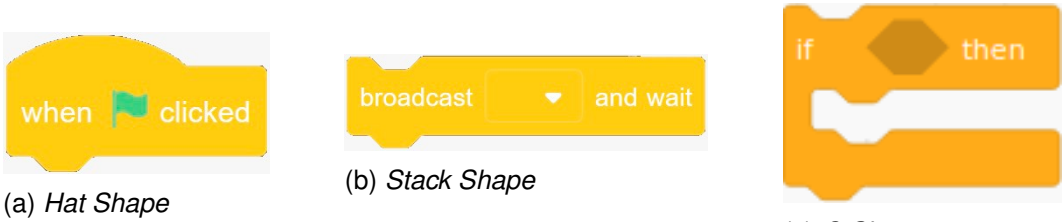
Figura 15 – Exemplo de classe para um bloco

Este padrão é representado na ontologia através da propriedade de objeto *hasScript*, entre o ator e o primeiro bloco que inicia o *script*.

Esta hierarquia de classes, junto com suas respectivas propriedades de objeto e de dados, pode ser visualizada na Figura 14.

3.2.4.1 Formatos de Blocos

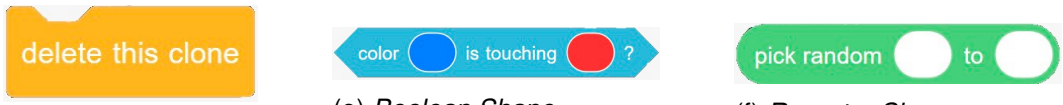
O formato dos blocos Scratch determina o encaixe que os mesmos podem ter com outros blocos. Ou seja, a forma do bloco determina as relações que este bloco pode ter com outros, como blocos sucessores e antecessores ou até mesmo a parametrização. A determinação do encaixe de blocos através do formato facilita o aprendizado, associando o fator visual intuitivo à tarefa.



(a) *Hat Shape*

(b) *Stack Shape*

(c) *C Shape*



(d) *Cap Shape*

(e) *Boolean Shape*

(f) *Reporter Shape*

Figura 16 – Formatos de um bloco Scratch

Ao todo, existem seis formatos distintos de blocos (Figura 16):

- *Hat Blocks*: blocos de chapéu são aqueles que iniciam todos os *scripts*. Eles possuem a parte superior arredonda e um encaixe na parte inferior, de modo que só é possível encaixar blocos sucessores, nunca antecessores. No total, existem 11 (onze) blocos com este formato.
- *Stack Blocks*: blocos de pilha são responsáveis pela grande maioria dos coman-

dos. Eles possuem um formato que permite encaixes acima e abaixo, sendo possível encaixar sucessores e antecessores. Este é o formato de bloco mais comum, existindo 77 (setenta e sete) blocos diferentes com este formato.

- *Boolean Blocks*: blocos booleanos são responsáveis por condicionais, eles retornam sempre um valor verdadeiro ou falso. Seu formato é hexagonal e são utilizados como *inputs* de outros blocos, como estruturas de repetição e condicionais. Existem 13 (treze) blocos diferentes deste formato.
- *Reporter Blocks*: blocos relatores são blocos responsáveis por retornar números e *strings*. Estes podem retornar valores decorrentes de ações, como operações matemáticas, ou valores armazenados em variáveis. De formato arredondado, estes blocos são utilizados como *inputs* de outros blocos, existindo 37 (trinta e sete) variações de blocos únicos deste formato.
- *C Blocks*: blocos no formato de C são responsáveis por representar estruturas de repetição e condicionais. Eles possuem formatos similar ao de blocos de pilha, porém também encapsulam outros blocos, ou seja, são capazes de possuir blocos encaixados na sua parte interna, além de um antecessor e sucessor. No total, existem 5 (cinco) blocos com este formato.
- *Cap Blocks*: blocos finalizadores são blocos que finalizam *scripts*. Eles possuem um encaixe na parte superior e são retos na parte inferior, permitindo apenas o encaixe de antecessores. Existem apenas 2 (dois) blocos deste formato.

Deste modo, a classe *BlockShape* foi representada como uma subclasse de *ScratchObject* e disjunta de *Project*, *Actor* e *Frame*, representando o formato de um bloco. Ademais, foram criadas subclasses para cada um dos formatos específicos, com cada subclasse representando blocos que possuem aquele formato e, consequentemente, as restrições de relacionamento com outros blocos pertinentes para o mesmo.

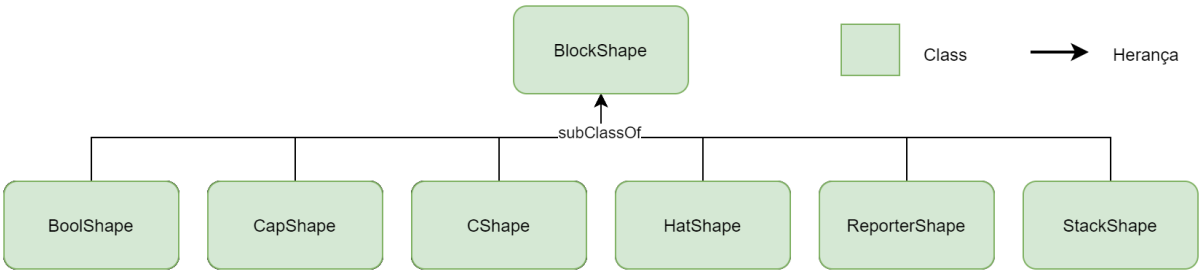
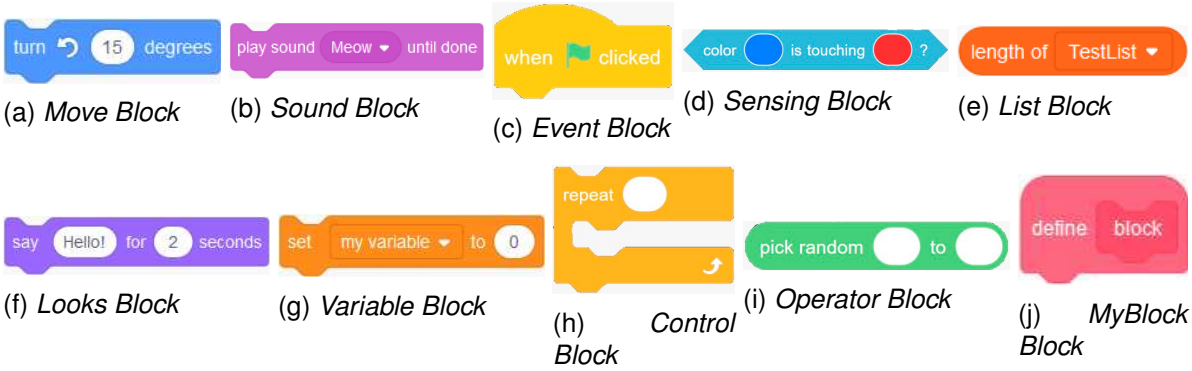


Figura 17 – Hierarquia de classes para formatos de bloco

Esta hierarquia de classes, junto com suas respectivas propriedades de objeto e de dados, pode ser visualizada na Figura 17.

3.2.4.2 *Categoria de Blocos*

A categoria de um bloco determina sua funcionalidade, ou seja, o comportamento esperado. Assim, blocos com efeitos similares são agrupados na mesma categoria e isto é visível pela sua tonalidade.



(a) *Move Block* (b) *Sound Block* (c) *Event Block* (d) *Sensing Block* (e) *List Block*

(f) *Looks Block* (g) *Variable Block* (h) *Control Block* (i) *Operator Block* (j) *MyBlock Block*

Figura 18 – Categorias de um bloco Scratch

No total, existem 10 (dez) categorias distintas de blocos (Figura 18):

- *Motion Blocks*: blocos de movimentação são aqueles que controlam a movimentação de uma *Sprite* e, consequentemente, não estão disponíveis para *Stages*. De cor azul, existem, o total, existem 17 (dezessete) blocos únicos de movimentação.
- *Looks Blocks*: blocos visuais são blocos roxos que controlam o visual de um ator, seja ele *Sprite* ou *Stage*. Existem 23 (vinte e três) blocos desta categoria no Scratch.
- *Sound Blocks*: blocos sonoros são aqueles que controlam o som da aplicação em Scratch. De cor magenta, existem 16 (dezesseis) blocos desta categoria.
- *Event Blocks*: blocos de evento são responsáveis pela captura de eventos e iniciar *scripts*. No total, existem oito blocos desta categoria, os quais apresentam a coloração marrom.
- *Control Blocks*: blocos de controle são utilizados para controlar o fluxo da aplicação, contendo comandos como condicionais, repetições e paradas. De cor dourada, existem 11 (onze) blocos desta categoria.
- *Sensing Blocks*: blocos sensores são responsáveis pela detecção de diferentes fatores em um projeto, como posição do mouse e colisões entre objetos. De cor azul clara, o Scratch possui 18 (dezoito) variações deste bloco.

- *Operators Blocks*: blocos operadores são responsáveis por realizar operações matemáticas, booleanas e manipular *strings*. De cor verde, existem 18 (dezoito) blocos desta categoria.
- *Variables Blocks*: blocos de variáveis são responsáveis por armazenar e manipular variáveis numéricas e de *strings*. Esta categoria possui cor laranja escura e 5 (cinco) variações de bloco.
- *List Blocks*: blocos de lista possuem cor vermelha e são utilizados para armazenar e manipular listas. No total, existem 12 (doze) blocos desta categoria.
- *My Blocks*: blocos desta categoria representam procedimentos em Scratch. Tratam-se de blocos customizáveis criados pelo usuário, onde podem definir novos blocos e o comportamento a ser realizado quando os mesmos forem executados. De cor rosa, existem apenas dois blocos desta categoria.

Assim, a classe *BlockType* foi criada como uma subclasse de *ScratchObject* e disjunta de *Project*, *Actor* e *Frame*. Também foram criadas subclasses de *BlockType* para cada uma das categorias categorias, cada uma representando uma categoria de blocos que compartilha determinadas características como explicitado acima.

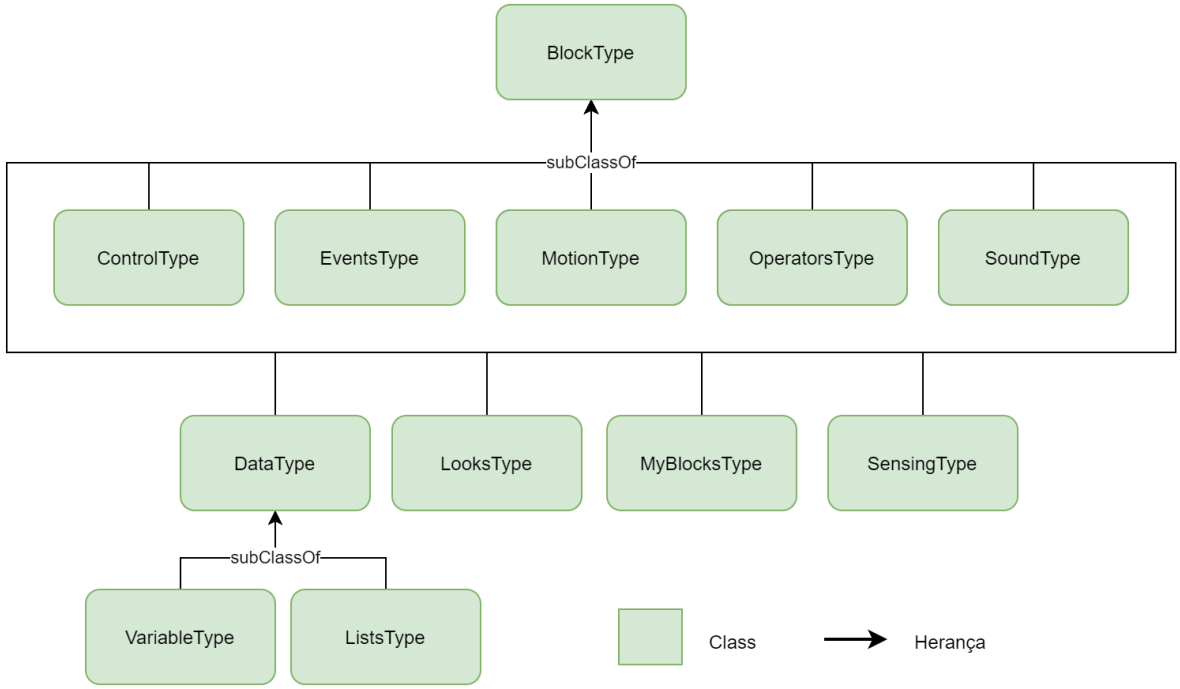


Figura 19 – Hierarquia de classes para categorias de bloco

Optou-se pela criação de uma categoria chamada de *DataType*, subclasse de *BlockType*, que agrupe as categorias de *Variable Blocks* e *List Blocks*, visto que ambas são categorias que manipulam variáveis e dados, apenas com tipos diferentes.

Assim, as classes *VariableType* e *ListType*, foram definidas como subclasses de *DataType*. Esta nomenclatura vai de encontro com a nova taxonomia utilizada na wiki do Scratch a partir da atualização para a terceira versão da aplicação.

Esta hierarquia de classes, junto com suas respectivas propriedades de objeto e de dados, pode ser visualizada na Figura 19.

3.2.4.3 Relações entre Blocos

Como exemplificado na Subsubseção 3.2.4.1, o formato de um bloco afeta como este pode relacionar-se com outros. Esta relação entre blocos pode ser de dois tipos:

- Relação de Sequência: referente ao encaixe de um bloco abaixo ou acima de outro. Esta relação é utilizada para o sequenciamento de comandos, formando os *scripts* de comportamento.
- Relação de Aninhamento: referente ao encaixe de um bloco na parte interna de outro. Esta relação ocorre com blocos de formato *CShape*, pois estes encapsulam uma sequência de blocos, comportamento comum para estruturas de repetição e condicionais.

Para representar a relação entre dois blocos, primeiramente foi definida uma relação base, representando um bloco estar conectado a outro. Esta relação foi chamada de *connects* e sua inversa de *isConnectedTo*. A primeira, representa um bloco que é dependente do outro, como um bloco anterior na sequência que conecta seu sucessor a esta pilha de blocos. A segunda, por sua vez, representa o oposto, um bloco que é dependente de outro, como um bloco conectado a seu antecessor.

A partir destas propriedades bases, foi definida uma subpropriedade para especificar a relação de anteceder e suceder chamada de *predecessor* e sua inversa de *sucessor*. Assim, uma relação de predecessor, uma especialização de *connects*, entre dois blocos A e B, define que o bloco A está conectado no topo do bloco B. O mesmo pode ser afirmado para a propriedade inversa *sucessor*, uma subpropriedade de *isConnectedTo*, a mesma define que o encaixe de um bloco embaixo de outro.

Para definição dos domínios e imagens destas relações, foram utilizados os formatos de blocos, visto que estes definem as relações entre eles. Desta forma a relação de sucessão possui como domínio a união entre as classes *CShape*, *CapShape* e *StackShape*, pois estes são os formatos de bloco que permitem o encaixe de blocos acima. Já a imagem desta propriedade se constitui pela união das classes *CShape*, *HatShape* e *StackShape*, pois estes são os formatos que permitem o encaixe de blocos abaixo.

A relação de aninhamento é definida de forma similar a de sequência, ou seja, através de duas propriedades de objeto que constituem subpropriedades de *connects*

e *isConnnectedTo*. Neste caso, as relações são de pai, *parentOf*, e filho, *childOf*. A primeira, uma especialização de connect, possui como domínio apenas a classe *CShape*, visto que este é o único formato de bloco que permite encapsular outros. A imagem, por sua vez, trata-se da união entre os formatos de blocos que permitem a conexão de outros acima, ou seja, todos aqueles que podem ser sucessores. Com isto, temos a união entre *CShape*, *CapShape* e *StackShape*.

3.2.5 Inputs

No Scratch diversos blocos podem possuir um ou mais *inputs* de dados para parametrizar o seu comportamento. Estes inputs podem ser de diversos formatos e preenchidos de diversas formas, como pode ser observado na Figura 20.

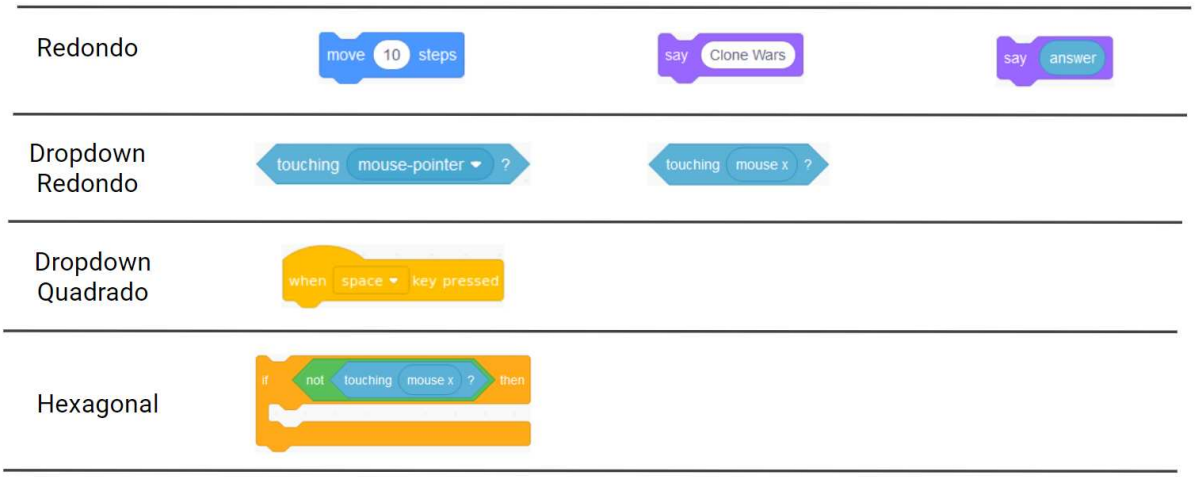


Figura 20 – Representação de *Inputs*

Existem duas características importantes pelas quais podemos diferenciar as entradas de dados: seu formato e seu tipo. No que refere-se ao formato, uma entrada de dados de um bloco pode ser hexagonal, redonda ou quadrada. De forma similar ao formato de bloco, o formato do *input* define quais blocos podem ser utilizados. Já no que diz respeito ao tipo, este pode ser um campo livre, permitindo que o usuário digite valores, ou já possuir uma lista de valores predefinidos.

Estes dois fatores combinados geram quatro diferentes *inputs*:

- *Inputs* Redondos de Valor: *inputs* redondos de valor podem ser valores inteiros ou strings informadas diretamente pelo usuário ou ter blocos de formato arredondado (*ReporterShape*) encaixados como entrada de dados. Neste último caso, o valor retornado pelo bloco arredondado será passado como *input* para o bloco.

- *Inputs* Redondos de Lista: *inputs* redondos de lista são similares a redondos de valor, pois também podem ter blocos com formato arredondado (*ReporterShape*)

encaixados como entrada de dados. Porém, o que os difere é que blocos de *input* de lista não podem ter valores entrados diretamente pelo usuário, neste caso, o usuário necessita escolher um dos valores da lista pré-definida.

- *Inputs* Quadrados de Lista: *inputs* quadrados de lista obrigam o usuário a escolher um dos valores pré-definidos, ou seja, não permitem que o mesmo digite valores livremente ou que encaixe outros blocos no lugar. Comumente, estes blocos possuem listas com valores padrão como teclas do teclado para captura de eventos.

- *Inputs* Hexagonais: *inputs* hexagonais são sempre referentes a valores booleanos. Estas entradas de dados obrigam o usuário a encaixar um bloco de formato hexagonal (*BooleanShape*), não permitindo a entrada de valores livres ou seleção por lista. O valor retornado pelo bloco (ou sequencia de blocos) encaixado é passado como entrada de dados.

Assim, para representação dos *inputs* criou-se inicialmente uma classe *Input*, subclasse de *ScratchObject* e disjunta das classes *Project*, *Actor* e *Frame*. A opção por não representá-la como disjunta de *BlockShape* e *BlockType* se dá pelo fato de um bloco poder ser utilizado como *input* de uma classe, como é o caso de blocos booleanos ou blocos de variáveis.

Ademais, criou-se uma hierarquia onde, primeiramente, os *inputs* são agrupados em virtude de seu formato. Assim, as classes *RoundInput*, *HexInput* e *SquareInput*, todas subclasses de *Input*, representam as entradas de dados.

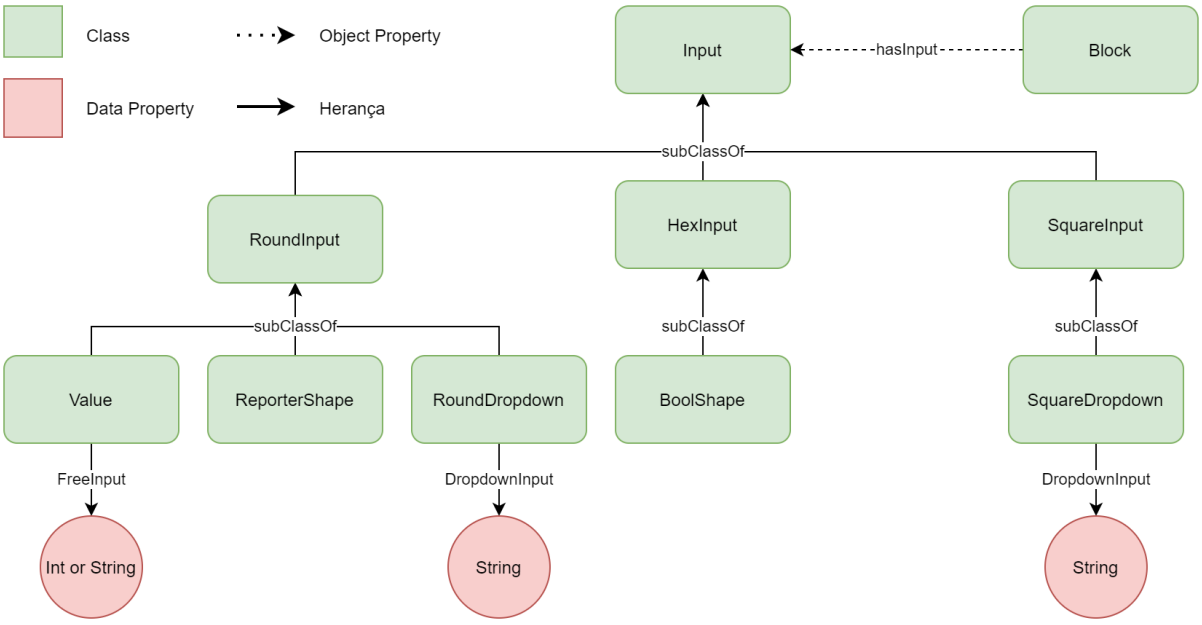


Figura 21 – Hierarquia de classes para *Inputs*

Como subclasses de *RoundInput*, foram definidas duas novas classes, *Value* e *RoundDropdown* para representar, respectivamente, entradas de dados livres e seleções em lista. Além disso, foi adicionada a propriedade de subclasse de *RoundInput* para a classe *ReporterShape*, de modo a permitir que estes blocos, compatíveis em formato sejam encaixados.

Já para *inputs* de formatos hexagonais, a mesma foi adicionada como superclasse de *BoolShape*, de modo a permitir a utilização destes blocos como entrada de dados nestes casos. Para o caso de *inputs* quadrados, foi criada a classe *SquareDropdown* para representação do caso de seleção dentre uma lista de valores.

Esta hierarquia de dois níveis (Figura 21), com o primeiro em relação ao formato, permite uma fácil expansão futura do vocabulário caso novas opções de entradas de dados sejam adicionadas à ferramenta Scratch em novas versão. Para isto, basta adicionar a nova opção como uma subclasse do formato correspondente.

Para representar a conexão entre um bloco e seu *input*, foi definida uma propriedade de objeto base para representar a conexão entre a classe *Block*, o domínio, e a classe *Input*, a imagem. Esta relação foi chamada de *hasInput* e sua inversa de *inputOf*.

A seguir, cada relação é especificada para representação da relação de possuir um input de um formato específico. Assim, a relação *hasInput* possui três subpropriedades, *hasRoundInput*, *hasHexInput* e *hasSquareInput*, com suas respectivas relações inversas, *roundInputOf*, *hexInputOf* e *squareInputOf*. Para estas subpropriedades, optou-se pela definição de domínios e imagens mais restritos. Por exemplo, para a propriedade *hasRoundInput*, o domínio é composto pela união entre todos os blocos que possuem espaços de encaixe arredondados, enquanto sua imagem é a união entre um valor inteiro, uma string e a classe para blocos arredondados *ReporterShape*.

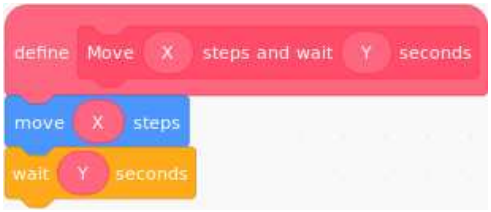
3.2.6 Definição de Procedimentos

Procedimentos são a principal forma de abstração, junto a variáveis, utilizada na programação procedural. Eles permitem que o usuário defina sequências de passos parametrizáveis a serem reutilizados ao longo da aplicação. No Scratch, podemos conceber que a grande maioria dos blocos trata-se de uma chamada de procedimentos. Por exemplo, o bloco *Move()Steps* nada mais é do que uma chamada para uma função que executa a adição do número de passos a variável que representa a posição X da *Sprite*.

Assim, a definição de novos blocos no Scratch e, consequentemente, de procedimentos, se dá através da categoria de blocos *MyBlocks*. Esta categoria de blocos possui duas variações: um bloco de definição de função e um bloco de chamada de função, demonstrados na Figura 22.

O bloco de definição de procedimentos possui o formato de *HatBlock*, de forma que

só é possível encaixar blocos abaixo do mesmo. Na criação deste bloco é possível definir um número arbitrário de argumentos a serem passados, podendo optar entre parâmetros booleanos, de formato hexagonal, ou inteiros e strings, de formato arredondado. A cada argumento declarado também é possível atribuir um nome, como exemplificados na Figura 22, onde são definidos dois argumentos inteiros ou strings chamados de X e Y. Além disso, é possível atribuir nomes a esta função, de modo que estes se intercalem com os argumentos e formem o nome de exibição deste novo bloco. No exemplo anterior, foram intercalados nomes com os argumentos de forma a forma o novo bloco chamado *Move X steps and wait Y seconds*.



(a) Definição de Procedimento

(b) Chamada de Procedimento

Figura 22 – Procedimentos em Scratch

Os blocos encaixados logo abaixo do bloco de definição do procedimento representam o comportamento a ser executado quando este procedimento for chamado. Para estes blocos sucessores da definição do procedimento, é possível utilizar qualquer bloco do Scratch, incluindo os parâmetros do procedimento, que são considerados como blocos de variáveis neste escopo. Esta definição de procedimento pode ser observada na Figura 22, onde são encaixados os blocos *Move()Steps* e *Wait()Seconds*, com os parâmetros X e Y sendo passados como inputs para definir a quantidade de passos e a quantidade de segundos.

Assim, os blocos de procedimentos foram definidos como subclasses de *MyBlockType* devido a sua funcionalidade. Além disso, o bloco de definição de procedimento também é uma subclasse de *HatShape*, devido a seu formato que afeta sua relação com outros blocos, enquanto o bloco de chamada de procedimento é uma subclasse de *StackShape*.

A relação entre a chamada de procedimento e o bloco ao qual ela referencia foi modelada como uma propriedade de objeto chamada "*calls*", com a respectiva relação inversa "*isCalledBy*". Além disso, para armazenar o nome do procedimento definido, foi modelada uma propriedade de dados do tipo string chamada *ProcedureName* e relacionada ao bloco de definição de função, onde é armazenado o nome completo de exibição do bloco exibido.

Os argumentos definidos em um procedimento, por sua vez, se dão pela propriedade de objeto *defines*, entre o bloco de definição de funções e blocos do tipo *DataType*. Assim, é permitida a definição de um argumento de diferentes tipos, podendo

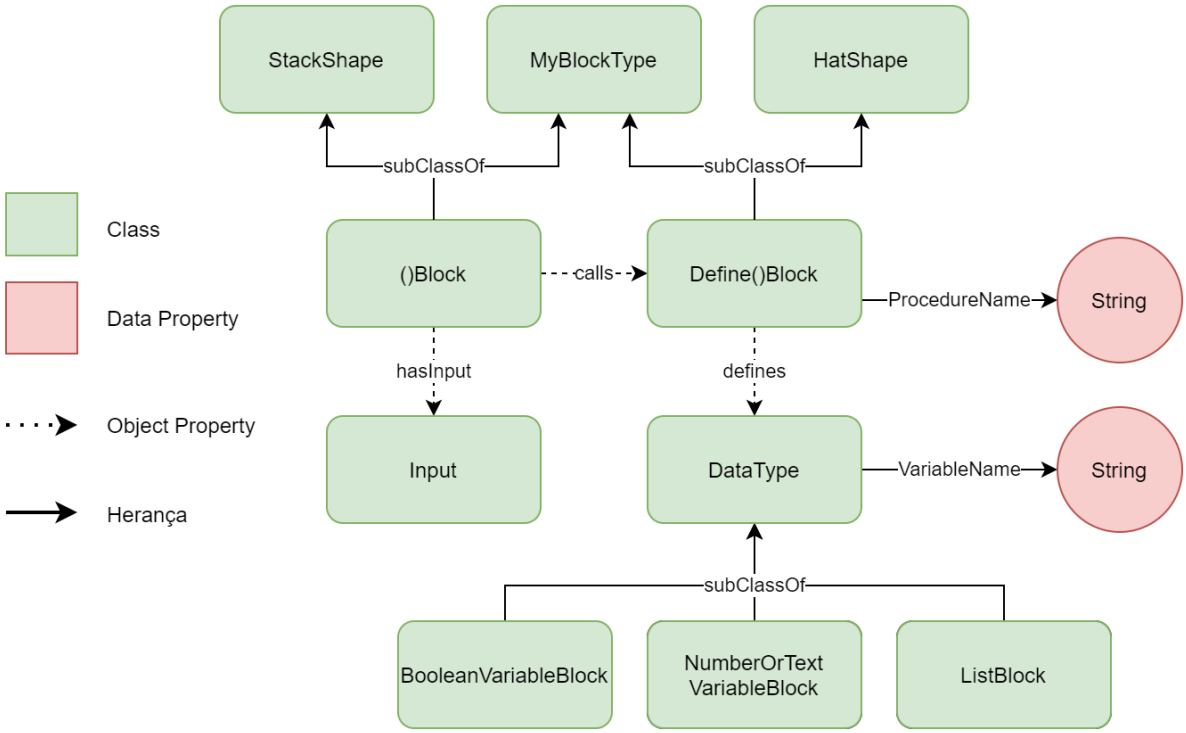


Figura 23 – Representação de Procedimentos - categoria *MyBlock*

ser booleano, numérico, texto ou lista. Já a chamada de função, por sua vez, possuir uma relação com inputs, permitindo que estes sejam outros blocos ou até mesmo valores diretamente entrados pelo usuário.

Esta hierarquia de classes, junto com suas respectivas propriedades de objeto e de dados, pode ser visualizada na Figura 23.

3.3 Representação da Avaliação de Pensamento Computacional

Para realização do acompanhamento da evolução de um aluno em relação ao seu conhecimento de PC, é necessária a representação das avaliações realizadas sobre seus projetos ao longo do tempo. Desta forma, será possível verificar sua evolução ao longo de uma ou mais atividades.

Como discutido na Subseção 2.1.3, devido a grande divergência entre os conceitos e habilidades relevantes para mensuração do PC, além da grande variedade de metodologias para execução desta avaliação, foi dado especial atenção na modelagem da ontologia de avaliação para permitir uma fácil expansão, além de fácil inferência.

Com estes fatores em mente, optou-se por uma adaptação da abordagem utilizada em (ARAÚJO; LIMA; HENRIQUES, 2019), realizando a separação da representação dos conceitos de PC da execução da avaliação em si. Este formato permite uma fácil adição de novos termos, como novos conceitos ou habilidades a serem mensurados,

permitindo a expansão e reuso do vocabulário. Esta representação será discutida em maiores detalhes nas subseções seguintes.

3.3.1 Conceitos

Devido a grande divergência na definição de quais conceitos e habilidades devem ser avaliados, somado a existência de variadas metodologias para execução da avaliação, optou-se por uma representação que fornecesse uma boa representação do que está sendo avaliado e uma fácil expansão do vocabulário.

Para isto, optou-se pela representação dos conceitos de pensamento computacional como uma classe específica ao invés de apenas propriedades de dados da avaliação. Este formato permite que sejam atreladas anotações que busquem auxiliar na compreensão do conceito representado. Isto facilita para o usuário o entendimento de quais fatores são considerados em cada conceito, permitindo compreender melhor sua representação.

Assim, criou-se uma classe *CTConcept*, representando genericamente um conceito do pensamento computacional. Esta classe possui uma propriedade de objeto chamada *isEvaluatedBy*, com sua relação inversa *evaluates*. Esta propriedade representa a relação de uma avaliação de pensamento computacional de um projeto com os conceitos avaliados especificamente. A classe *CTConcept* também possui um atributo de dados chamado de *ConceptScore*, representando a pontuação atribuída a este conceito durante a avaliação associada.

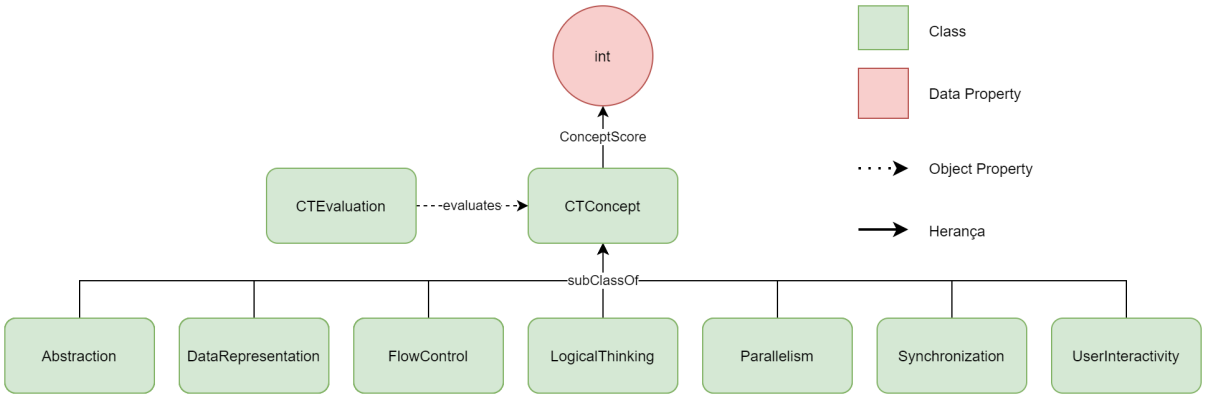


Figura 24 – Representação de Conceitos de Pensamento Computacional

Expandindo a hierarquia de classes para conceitos, esta classe genérica é especializada criando-se uma subclasse para cada conceito específico. Esta estrutura facilita o reuso e expansão do vocabulário, permitindo a adição de novos conceitos como subclasses de *CTConcept*, pois estes herdarão as propriedades e comportamentos padrões.

Apesar desta fácil expansão, era necessária a definição de conceitos de PC para

compor a ontologia. Assim, optou-se por seguir a definição utilizada na ferramenta Dr. Scratch devido à sua alta aceitação e utilização. Assim, foram definidos sete classes para conceitos, especializando-se a partir de *CTConcept*: *Abstraction*, *DataRepresentation*, *FlowControl*, *LogicalThinking*, *Parallelism*, *Syncrhonization* e *UserInteractivity*. Cada conceito foi definido com meta-propriedades de anotação buscando auxiliar na compreensão de sua representação.

3.3.2 Avaliação

Para verificação da evolução do aluno, é necessário o registro e o acompanhamento das avaliações sobre o nível de PC demonstrado em seus projetos Scratch. Este acompanhamento pode ser dar em dois diferentes níveis de granularidade: ao longo de uma mesma atividade, buscando identificar padrões de comportamento para o aluno e conceitos que possua maior dificuldade em desenvolver; e ao longo de diversas atividades, olhando principalmente para o resultado final, buscando identificar informações macro, como o nível de conhecimento geral do aluno nos fundamentos de PC.

Assim, optou-se pela criação de uma classe específica na ontologia, chamada de *CTEvaluation*, para representação de uma avaliação do nível de pensamento computacional demonstrado em um projeto Scratch. Esta avaliação pode contemplar ou não diversos conceitos e habilidades do PC, podendo incluir um variado número de diferentes combinações. Para representar quais conceitos foram considerados na execução da avaliação, foi criada uma propriedade de objeto chamada *evaluates*, com as classes *CTEvaluation* e *CTConcept* como imagem e domínio, respectivamente. Esta modelagem permite que diferentes instâncias de avaliação considerem diferentes combinações de conceito, permitindo uma maior flexibilidade de representação.

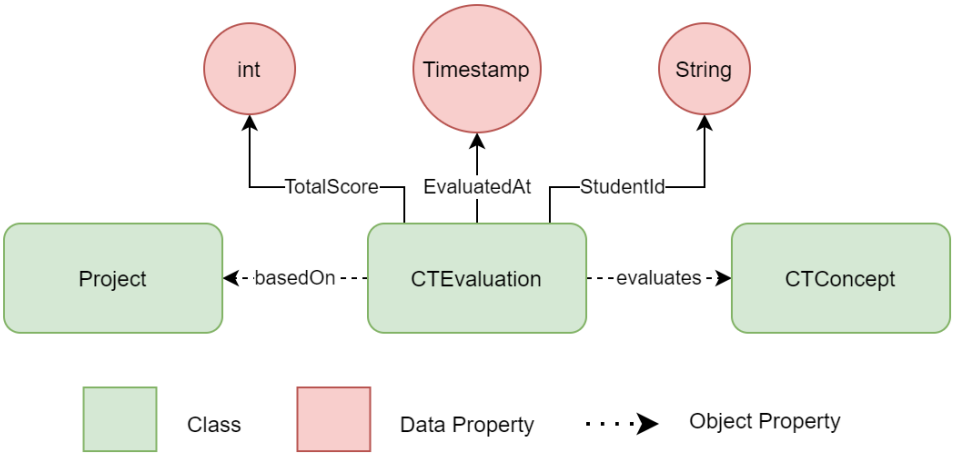


Figura 25 – Representação de avaliação de Pensamento Computacional de um projeto

Além da associação a conceitos, uma avaliação também necessita de um projeto

Scratch, sobre o qual será baseada sua execução. Assim, foi definida uma propriedade de objeto chamada *basedOn* para representá-la, com domínio da classe *CTEvaluation* e imagem a classe *Project*. Soma-se a ela, a definição de uma propriedade de dados chamada de *TotalScore*, criada como representação da pontuação final da avaliação conduzida.

Através destas definições, somadas àquelas apresentadas nas seções anteriores, é possível realizar a representação de uma avaliação do nível de pensamento computacional de um projeto Scratch de forma completa. Para isso, é necessária a criação de uma instância de *CTEvaluation* para representá-la, associando a ela os conceitos considerados em sua execução. A seguir, é possível atribuir uma nota individual a cada conceito e uma nota final a avaliação, através de suas respectivas propriedades de dados.

Entretanto, este modelo ainda não permite a representação da evolução de um aluno ao longo do tempo. Considerando este fator, foi definida uma propriedade de dados chamada *evaluatedAt* para a classe *CTEvaluation*, armazenando um *timestamp* com uma representação de tempo a nível de milissegundos. Esta propriedade representa o momento de execução da avaliação em questão. Assim, é possível associar a cada instância de avaliação o momento em questão considerado, permitindo a representação de ambos os níveis de granularidade de acompanhamento anteriormente mencionados.

O acompanhamento de um único projeto é possível através da criação de diversas instâncias da classe de avaliação, cada uma com o *timestamp* do momento desejado e o estado do projeto naquele momento de tempo. Já o acompanhamento da evolução do aluno entre diferentes atividades poderá ser realizado através da comparação entre duas ou mais instâncias de avaliação com diferentes projetos associados. É importante ressaltar que para identificar se duas avaliações referem-se a mesma atividade ou não, basta a verificação do identificador único do projeto Scratch associado a cada uma delas.

3.4 Considerações do Capítulo

Através da ontologia definida nas seções anteriores, é possível realizar a representação de um projeto Scratch em sua totalidade, além de também representar a avaliação do nível de pensamento computacional demonstrado no mesmo. Esta representação utiliza-se de um vocabulário acessível para usuários da plataforma Scratch, pois utiliza como dicionário de dados a própria documentação da ferramenta.

Devido a extensão da representação de conhecimento desenvolvida – esta possui 190 classes, 45 propriedades de objeto e 9 propriedades de dados – é inviável a apresentação de sua hierarquia de classes completa. Assim, uma versão resumida pode

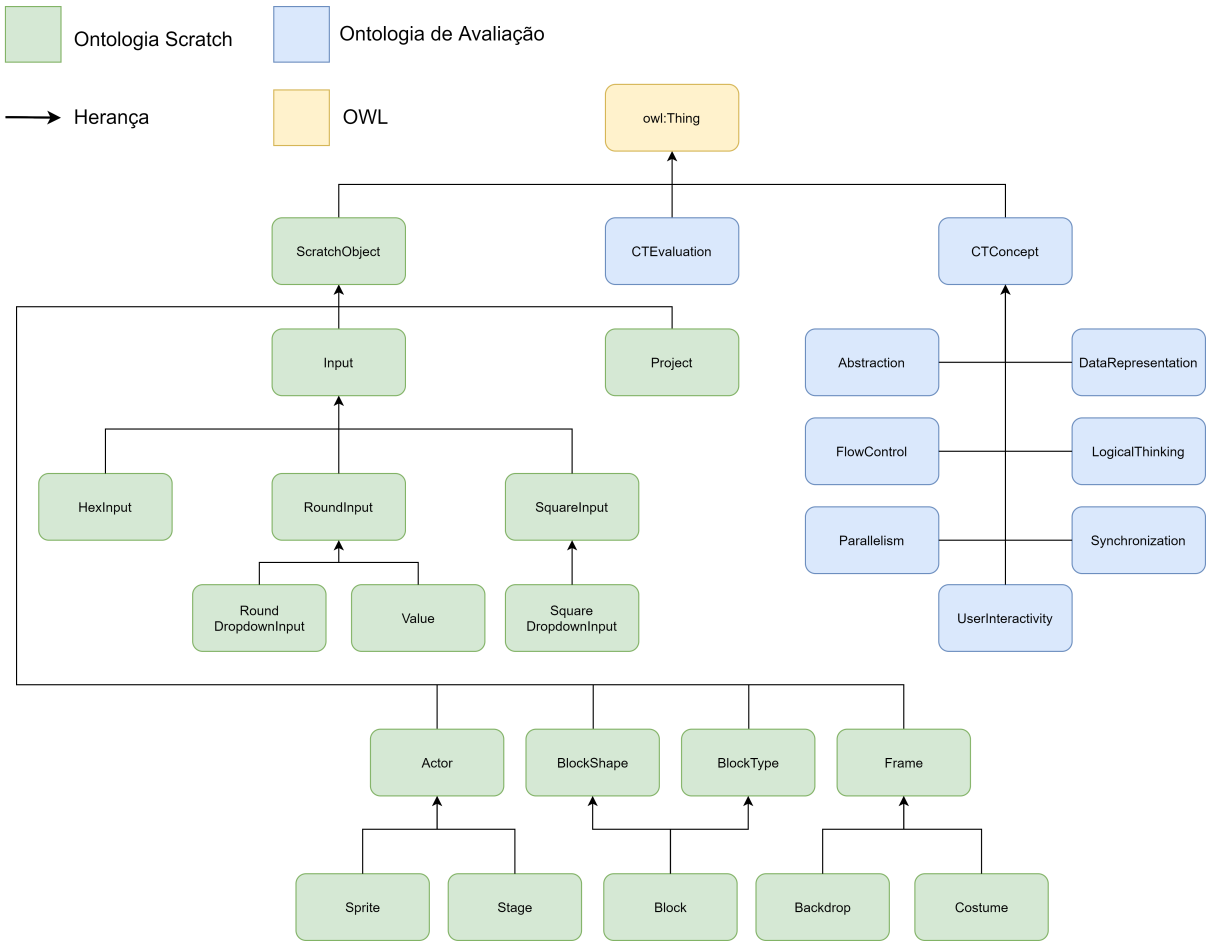


Figura 26 – Hierarquia de Classes Simplificada do OntoScratch

ser visualizada na Figura 26, onde a representação de blocos foi fortemente simplificada, não apresentando as classes para categorias, formatos e blocos específicos. As classes em verde apresentadas na figura pertencem a ontologia desenvolvida para descrição do universo Scratch, enquanto as em azul pertencem a ontologia de avaliação.

Com este modelo, é possível registrar o resultado efetivo do trabalho de um aluno – seu projeto – e também seu progresso – suas avaliações de PC –, em um formato formalmente definido e de fácil extensão. Estes fatores abrem caminho para a realização de novos trabalhos sobre estes dados, como análises e inferências, que agregarão valor direto para o professor na sala de aula.

4 AVALIAÇÃO E RESULTADOS

Neste capítulo serão apresentadas as avaliações realizadas sobre as ontologias propostas como representações de conhecimento, bem como o resultado obtido através destas avaliações. Para tanto, a Seção 4.1 falará sobre a metodologia de testes utilizada e o cenário de caso de uso aplicado. Já na Seção 4.2, será apresentado o desenvolvimento desta avaliação, enquanto a Seção 4.3 apresentará os resultados obtidos.

4.1 Metodologia

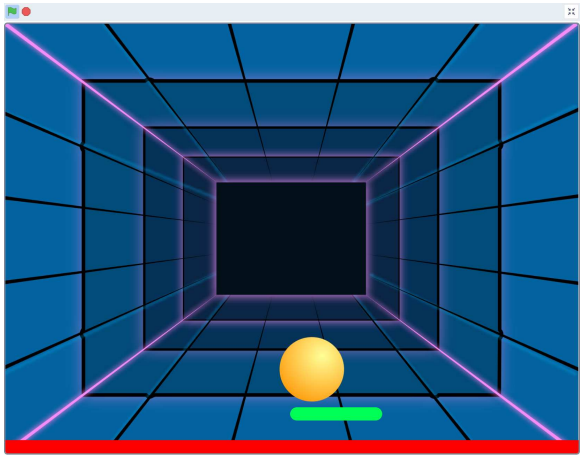
O cenário de avaliação proposto busca verificar o cumprimento dos dois principais objetivos no desenvolvimento desta representação de conhecimento: ser capaz de representar com a menor perda de representatividade possível um projeto Scratch; e permitir a realização de análises e inferências sobre estes dados.

Desta forma, propõe-se como cenário de avaliação a necessidade de um professor de avaliar o nível de pensamento computacional demonstrado em uma atividade por dois de seus alunos, utilizando a metodologia de avaliação do Dr. Scratch. Para atingir este objetivo, é necessário representar os projetos de ambos na ontologia e, através da utilização de mecanismos de inferência, realizar a avaliação desejada.

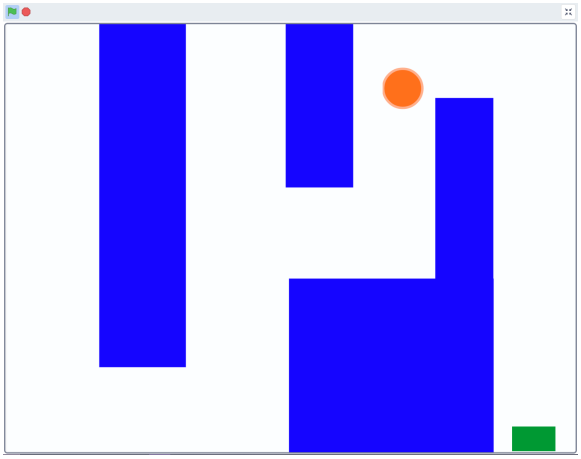
Para construção deste cenário de teste, optou-se pela escolha de dois projetos dentre os exemplos disponibilizados na documentação oficial do Scratch: *Pong Starter*[1] e textitMaze Starter[2]. Ambos foram escolhidos por apresentar uma boa variação de blocos e diferentes padrões de programação, porém não serem grandes a ponto de tornar difícil sua visualização e compreensão, como pode ser observado na Figura 27. Para facilitar sua referência, estes serão chamados nas seções seguintes deste capítulo como Projeto 1 e Projeto 2, respectivamente.

Por fim, o resultado da avaliação realizada através dos mecanismos de inferência será comparada à avaliação conduzida sobre os mesmos projetos diretamente na

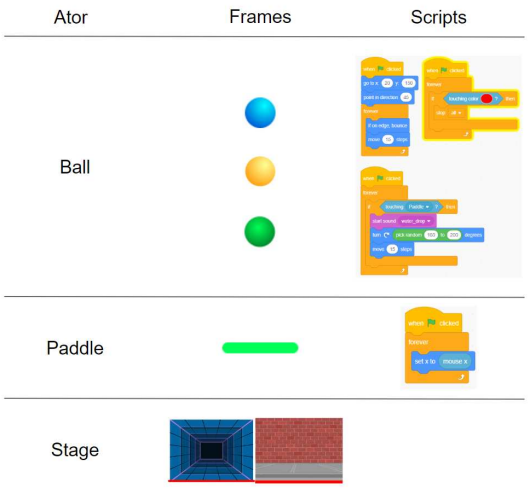
[1] https://scratch.mit.edu/projects/10128515/
[2] https://scratch.mit.edu/projects/10128431/



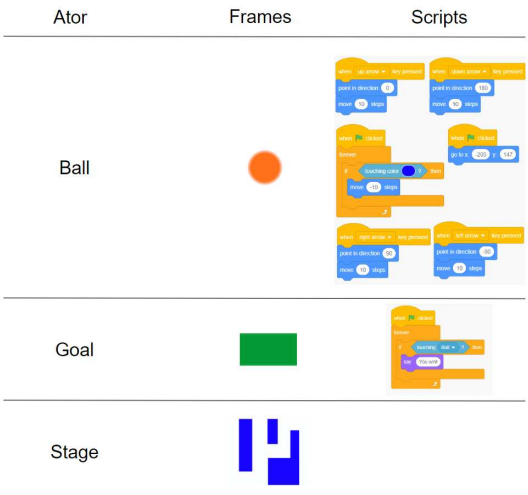
(a) Projeto 1 - *Pong Starter*



(b) Projeto 2 - *Maze Starter*



(c) Estrutura do Projeto 1 - *Pong Starter*



(d) Estrutura do Projeto 2 - *Maze Starter*

Figura 27 – Projetos de Teste

versão beta da ferramenta web Dr. Scratch[3].

4.2 Avaliação dos Projetos

Para execução do cenário de teste, foi necessária a escolha de um mecanismo de inferência para atuar sobre a ontologia. Com base nas opções atuais, optou-se pela escolha da linguagem SPARQL[4] devido a sua definição como recomendação de padrão pela W3C. Para execução destas consultas, foi utilizado o *framework* Snap-SPARQL através de um *plugin*[5] da ferramenta *Protégé*[6].

Este *framework* diferencia-se dos demais por operar em conjunto com um *reaso-*

[3]http://www.drscratch.org/
[4]https://www.w3.org/TR/sparql11-overview/
[5]https://github.com/protegeproject/snap-sparql-query
[6]https://protege.stanford.edu/

ner, permitindo a consulta à dados inferidos ao invés de apenas à dados explícitos (HORRIDGE; MUSEN, 2015). Soma-se a isto a sua praticidade de uso devido a sua integração com a ferramenta de modelagem *Protégé*, facilitando o desenvolvimento e sua utilização junto aos diversos motores de raciocínio disponibilizados. Assim, para a realização desta avaliação, optou-se pela utilização do *reasoner* Pellet (SIRIN et al., 2007) em sua versão incremental, junto ao Snap-SPARQL.

As consultas para realização da inferência seguem o padrão de regras apresentado pela ferramenta Dr. Scratch. Assim, são avaliados sete conceitos relacionados ao pensamento computacional, atribuindo uma pontuação de 0 a 3 para cada, onde o somatório das sete notas representa a pontuação final do projeto.

Desta forma, foram desenvolvidas oito consultas, uma para cada conceito avaliado e uma para cálculo da nota final. No geral, todas seguiam uma estrutura similar, apresentada na Listagem 4.1.

Listagem 4.1 – Estrutura de consultas de inferência

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
CONSTRUCT { ?evaluated_concept ctc:ConceptScore ?score }
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?evaluated_concept .
  ?evaluated_concept rdf:type ctc:EvaluatedConcept .
  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .
  # Rule 1
  OPTIONAL{
    # Padrao de grafo para pontuacao de nivel 1 para o conceito
  }
  # Rule 2
  OPTIONAL{
    # Padrao de grafo para pontuacao de nivel 2 para o conceito
  }
  # Rule 3
  OPTIONAL {
    # Padrao de grafo para pontuacao de nivel 3 para o conceito
  }
```

```
  BIND(BOUND(?level_1_block) AS ?has_level_1)
  BIND(BOUND(?level_2_block) AS ?has_level_2)
  BIND(BOUND(?level_3_block) AS ?has_level_3)
  BIND(
    if(?has_level_3,
      3,
      if(?has_level_2,
        2,
        if(?has_level_1,
          1,
          0
  ))) AS ?score)
}
```

Todas as consultas utilizam o operador “Construct” de forma a resultarem na criação de um novo padrão no grafo consultado. Ou seja, o resultado de sua execução é a associação da pontuação adequada para o conceito em questão através do atributo de dados *ConceptScore*.

Para verificar qual o nível adequado para o projeto, foram utilizadas três cláusulas opcionais especificando os padrões esperados do grafo para cada nível, de acordo com a tabela de regras do Dr. Scratch. Assim, posteriormente, é verificado se este padrão foi cumprido, com uma ordem de prioridade do maior nível ao menor, atribuindo o valor adequado à pontuação.

Todas as consultas podem ser conferidas em maiores detalhes no Apêndice A.

4.3 Resultados

Nesta seção, serão apresentados os resultados da avaliação dos projetos. Primeiramente, estes serão apresentados em sua representação ontológica. A seguir, serão executadas as avaliações na ferramenta Dr. Scratch e na ontologia, trazendo a comparação entre seus resultados.

A Subseção 4.3.1 apresentará os resultados do primeiro projeto de testes, *Pong Starter*. Já a Subseção 4.3.2 irá demonstrar os testes do segundo projeto, *Maze Starter*.

4.3.1 Projeto de Testes 1

O primeiro projeto, *Pong Starter*, possui três atores: o palco e duas *Sprites*, representando a bola e o jogador. Neste projeto, o jogador interage movendo uma barreira com o objetivo de rebater a bola, sem deixá-la tocar na parte inferior da tela.

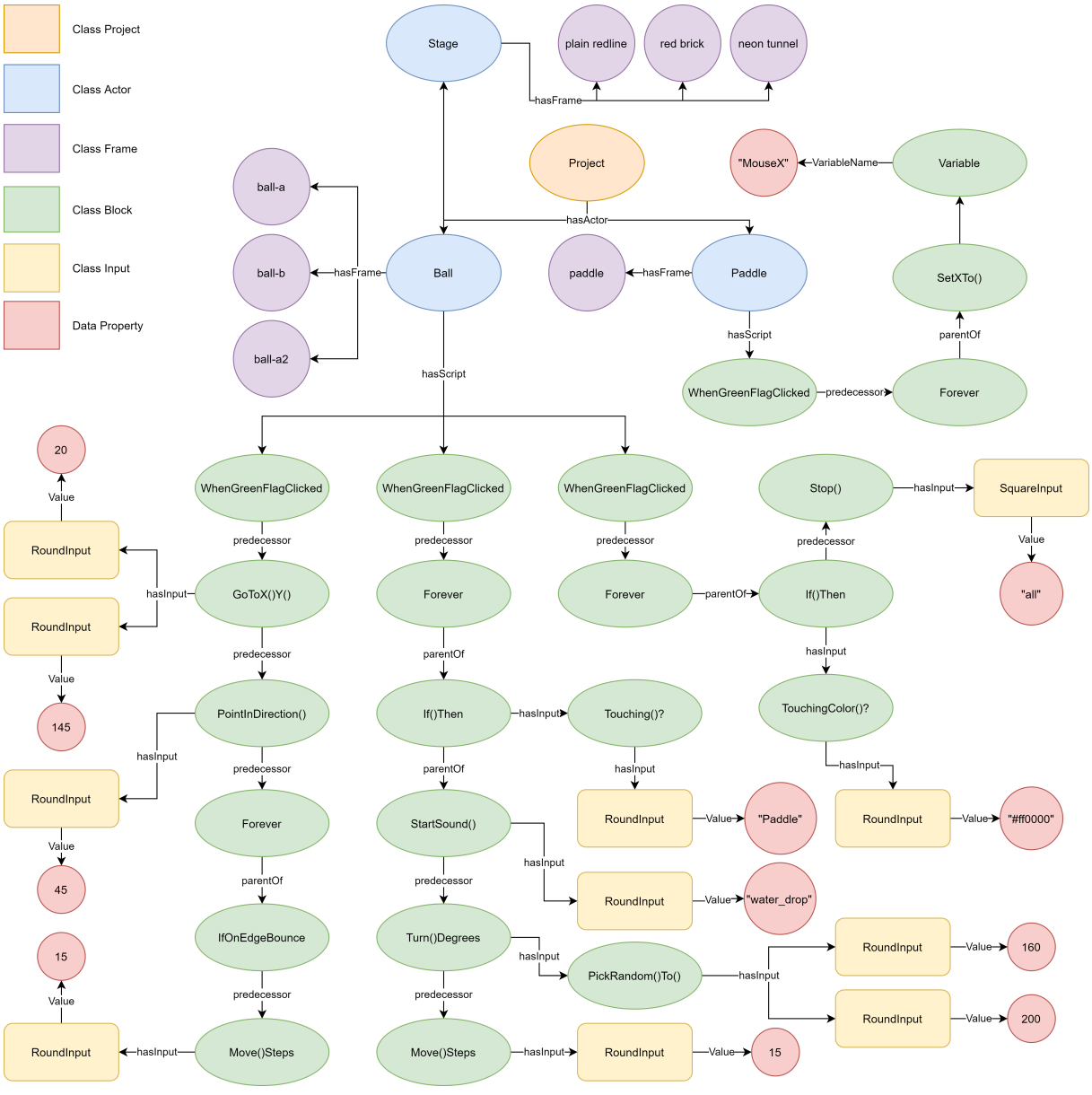


Figura 28 – Representação do Projeto 1

Assim, dois atores possuem blocos de modificação de comportamento: a barreira, realizando a movimentação do jogador através do mouse, e a bola, realizando a sua movimentação e a detecção das colisões. A representação ontológica deste projeto pode ser visualizada na Figura 28.

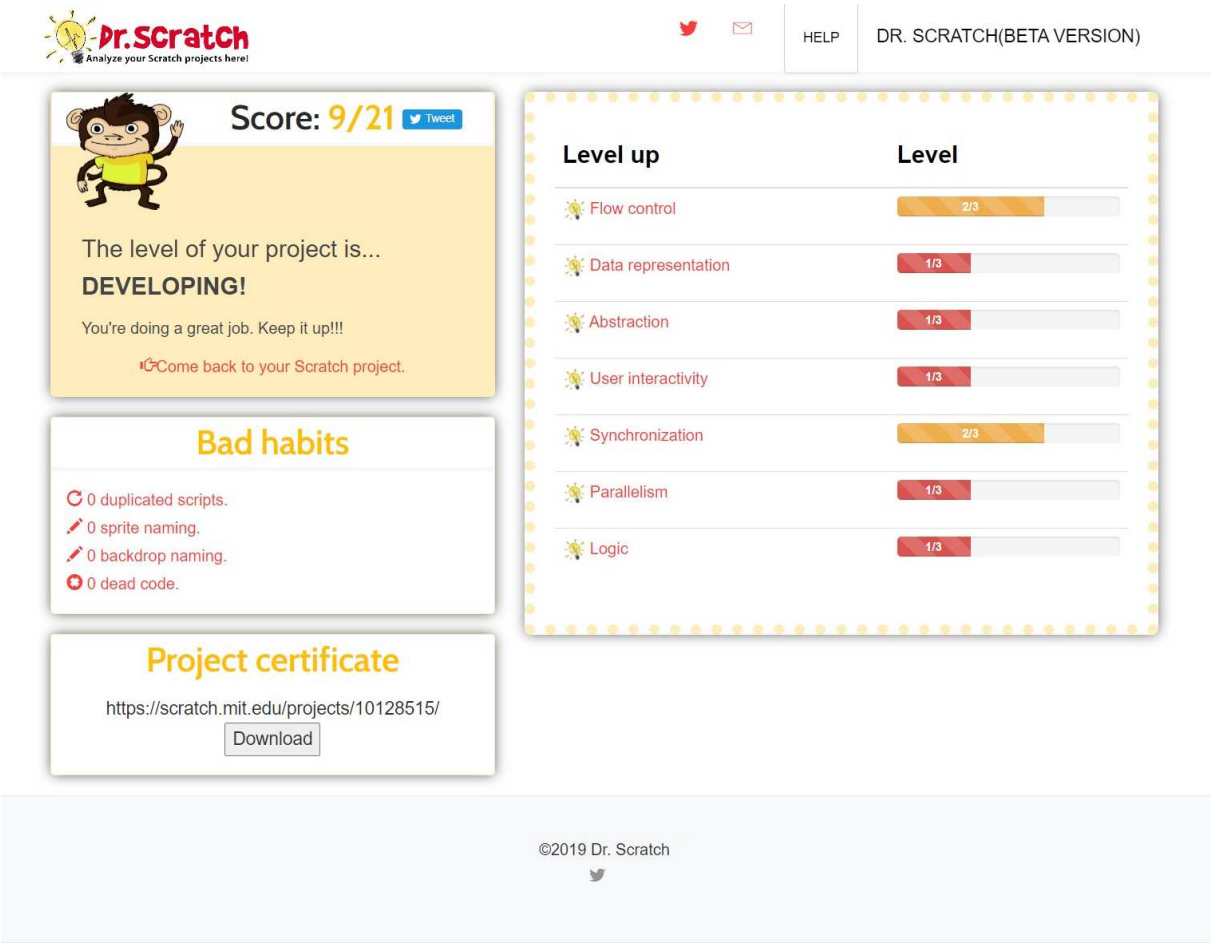


Figura 29 – Avaliação do Projeto 1 na plataforma Dr. Scratch

A seguir, executou-se a avaliação na ferramenta Dr. Scratch diretamente através de sua interface web. O resultado foi uma avaliação do projeto com uma pontuação de 9 pontos, dentre os 21 possíveis, classificando a proficiência demonstrada no projeto como “Em desenvolvimento” (Figura 29). Para a grande maioria das habilidades foi atribuída uma pontuação de 1 ponto na escala de 0 a 3, correspondendo a uma baixa proficiência demonstrada. As habilidades de Sincronização e Controle de Fluxo apresentaram uma pontuação maior, sendo ambas pontuadas como “Em desenvolvimento”.

Para a execução da avaliação na representação ontológica, foram utilizadas as 9 consultas SPARQL desenvolvidas, apresentadas na Seção 4.1 e disponíveis no Apêndice A. Através da execução das oito consultas para pontuação individual dos conceitos e, posteriormente, da consulta para pontuação final, obteve-se a representação

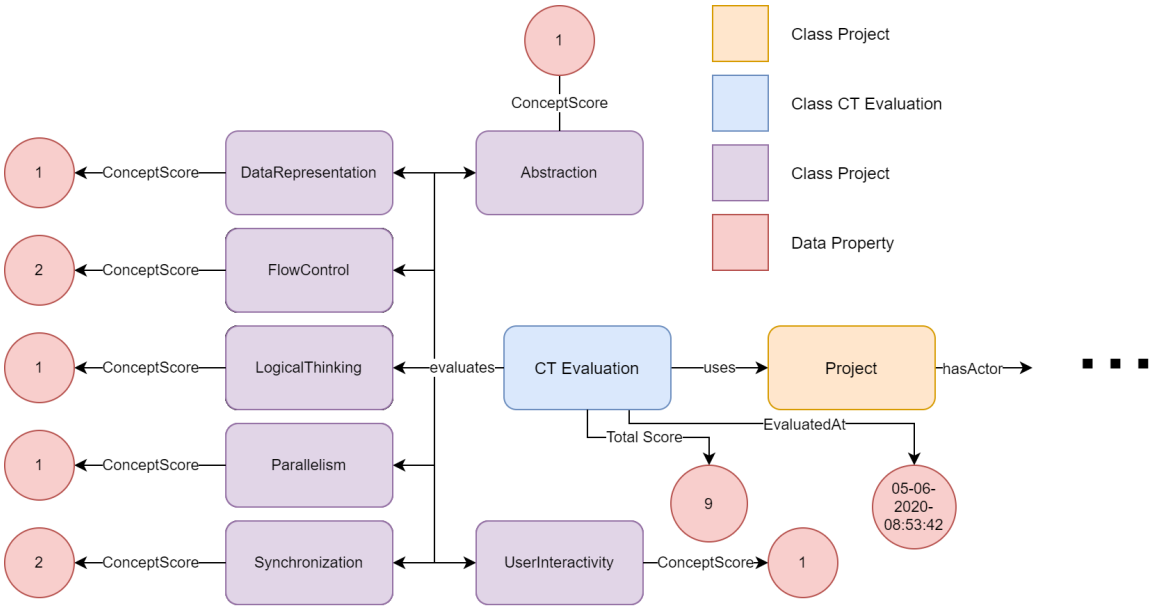


Figura 30 – Avaliação do Projeto 1 na OntoScratch

apresentada na Figura 30. A avaliação realizada através da ontologia apresentou as mesmas pontuações individuais para todas as habilidades em comparação com a realizada diretamente no Dr. Scratch e, consequentemente, também apresentou a mesma pontuação total.

4.3.2 Projeto de Testes 2

Já o segundo projeto de testes, *Maze Starter*, possui também três atores: palco, objetivo e o jogador. Nele, o jogador deve mexer a bola de forma a desviar dos obstáculos e chegar no objetivo. Desta forma, dois atores possuem *scripts*, sendo eles o jogador, encarregado da movimentação e colisão com obstáculos, e o objetivo, encarregado de verificar a condição de vitória.

Apesar de ambos os projetos possuírem o mesmo número de atores, suas estruturas de programação são amplamente distintas. O primeiro projeto possui apenas três fluxos de execução em um mesmo ator, com estes fluxos sendo mais longos e complexos. Enquanto isso, no projeto em questão, a abordagem utilizada privilegia um maior número de fluxos paralelos, com seis deles apenas no ator do jogador principal, sendo estes, em geral, mais curtos e simples. A representação ontológica deste projeto pode ser visualizada na Figura 31.

A seguir, foi executada a avaliação do projeto na ferramenta Dr. Scratch. Como pode ser observado na Figura 32, foi atribuída uma pontuação total para o projeto de 8 pontos dentre os 21 possíveis, categorizando-o também como “Em desenvolvimento”.

As pontuações individuais foram similares às do primeiro projeto, com a exceção para as habilidades “Sincronização” e “Interatividade com o Usuário”. Para a primeira,

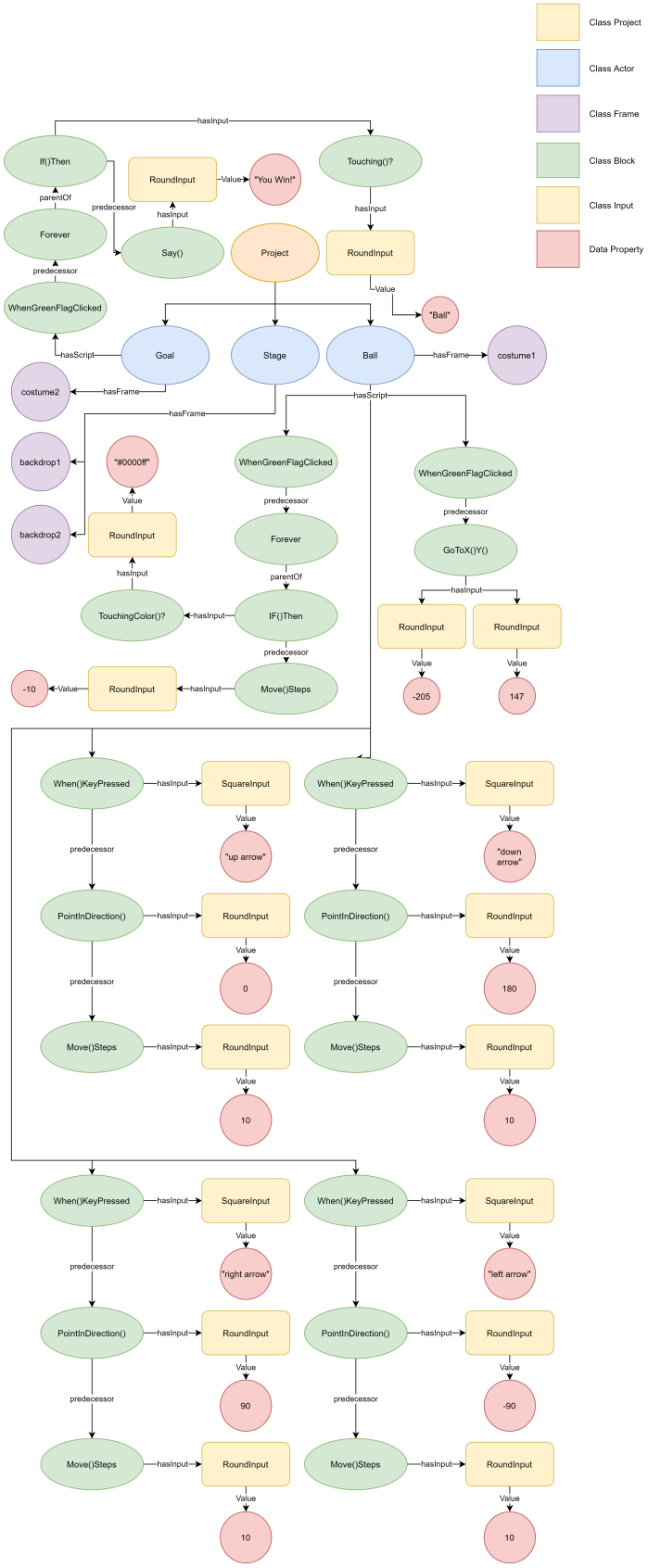


Figura 31 – Representação do Projeto 2

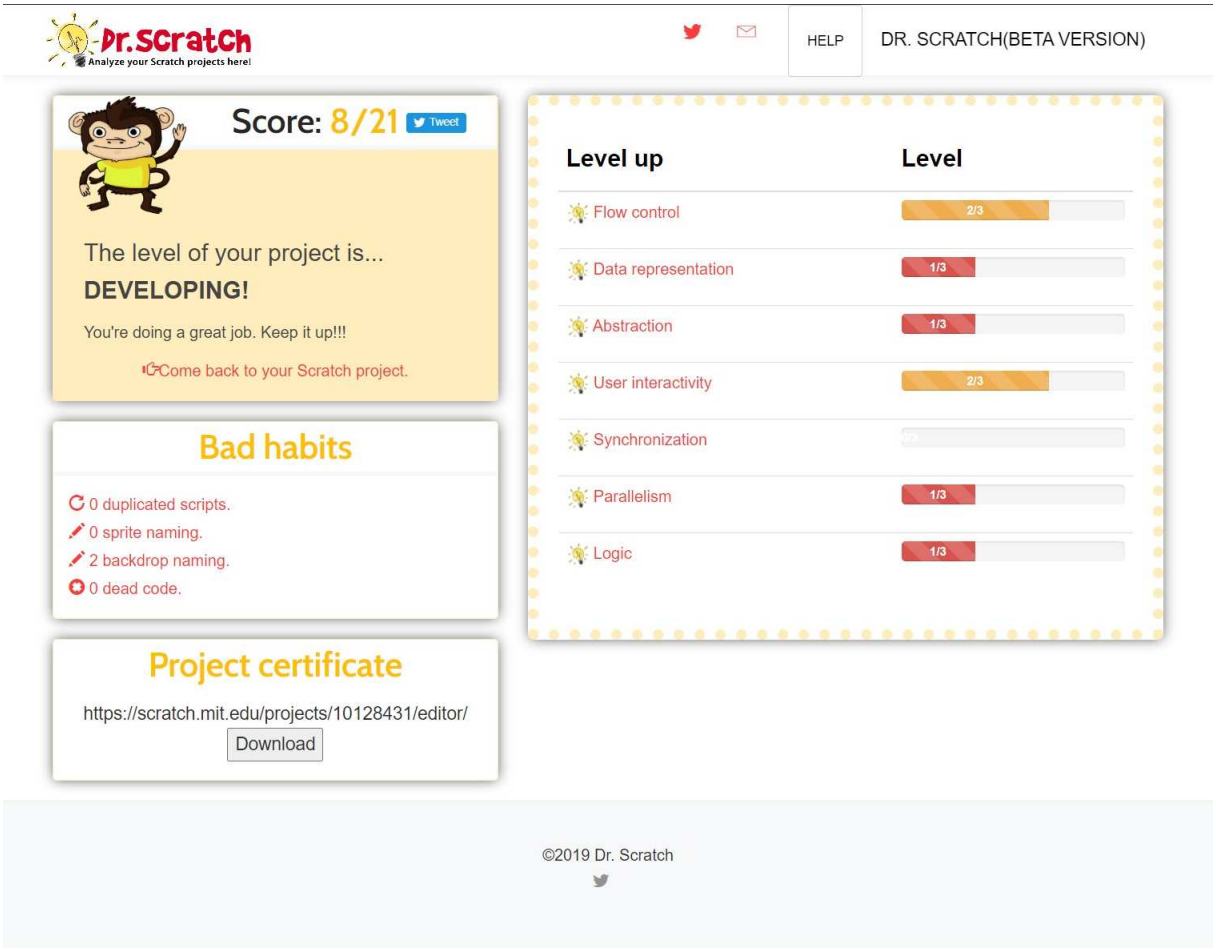


Figura 32 – Avaliação do Projeto 2 na plataforma Dr. Scratch

foi considerado que a mesma não foi demonstrada neste segundo projeto, resultando em 0 pontos. Isto ocorre devido a ausência de qualquer elemento que realize a sincronização entre os comportamentos de dois atores, um contraste em relação ao primeiro projeto.

Já para a interatividade com o usuário, foi atribuída a ela uma classificação de “Em desenvolvimento”, resultando em 2 pontos. Isto ocorre pela utilização de eventos para interação com o usuário diretamente pelo teclado.

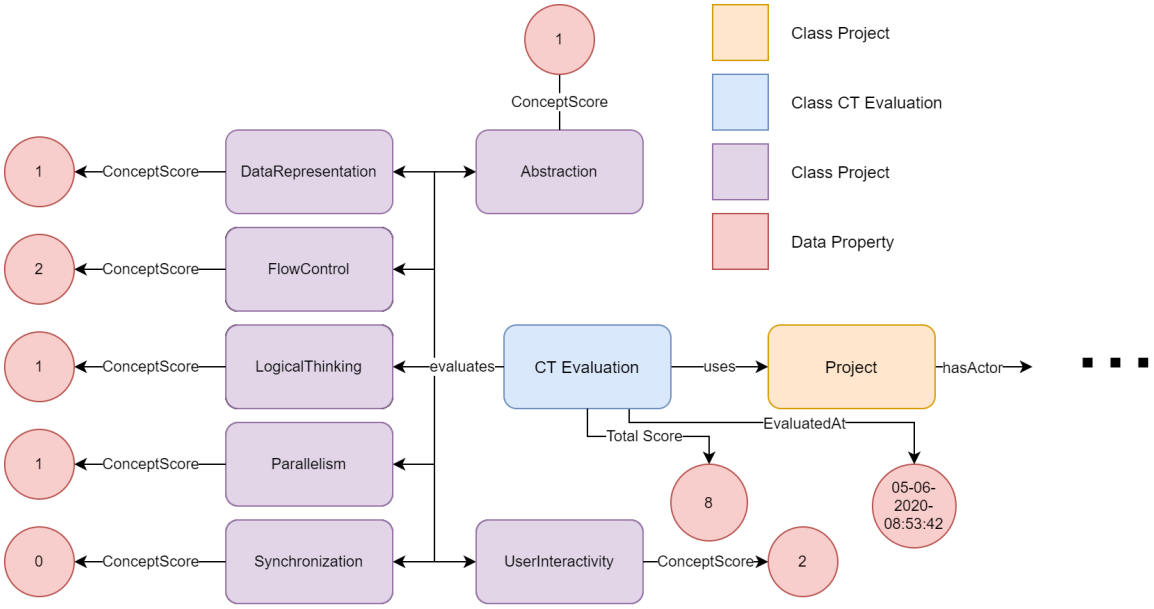


Figura 33 – Avaliação do Projeto 2 na OntoScratch

A seguir, o conjunto de consultas SPARQL foi executado sobre a representação do projeto na ontologia. Esta execução resultou na representação apresentada em Figura 33. Como podemos observar, foram instanciados nodos com as mesmas pontuações da avaliação realizada diretamente na ferramenta para cada habilidade e, consequentemente, também para pontuação total.

4.4 Considerações do Capítulo

Neste capítulo foram apresentados os testes sobre dois projetos reais de Scratch. Para isso estes projetos foram representados na ontologia proposta e, a seguir, avaliados através de consultas SPARQL e na ferramenta Dr. Scratch.

Foi possível observar que a inferência das habilidades de pensamento computacional demonstrada em ambos os projetos utilizando a representação ontológica mostrou-se eficaz, apresentando resultados equivalentes a avaliação realizada na ferramenta especializada Dr. Scratch.

5 CONSIDERAÇÕES FINAIS

A expansão de iniciativas para introduzir conceitos relacionados a computação no ensino à crianças vêm ocorrendo em diversos países, com a aplicação do projeto Clubes de Computação Criativa sendo aplicados desde 2015 na cidade de Pelotas/RS e um exemplo disto. Nestas iniciativas, o uso de ferramentas de programação visual, como o Scratch, vem sendo a principal abordagem, destacando-se como uma ferramenta prática e de baixo custo.

Neste cenário, surge o desafio de armazenar os dados coletados nestas atividades, para que estes sejam trabalhados e gerem evidências do resultado deste trabalho. Como obstáculo para isto, destaca-se a ausência de uma representação de conhecimento capaz de representar estes dados de forma a prover suporte para a análise do progresso do aluno ou da realização de outras inferências.

No âmbito destes desafios, este trabalho teve como objetivo o desenvolvimento de uma representação de conhecimento capaz de caracterizar os dados de um projeto Scratch e de avaliações do pensamento computacional demonstrado nele, provendo suporte para o acompanhamento e para a realização de análises e inferências.

Defende-se que estes objetivos podem ser atingidos através da representação de conhecimento OntoScratch, um conjunto de ontologias para utilização no ensino do pensamento computacional através do Scratch. Para isto, foram desenvolvidas duas ontologias, uma para a representação dos elementos de um projeto, e outra para representação de uma avaliação de pensamento computacional. Esta representação de conhecimento foi desenvolvida com o objetivo de ser facilmente reutilizável e expansível, além de permitir a realização de inferências sobre estes dados, de forma a servir como base para futuros trabalhos.

5.1 Conclusão

A concepção e criação da representação de conhecimento OntoScratch representa e armazena de maneira estruturada o percurso de alunos durante a elaboração de projetos Scratch visando analisar seu processo de aprendizagem de habilidades rela-

cionadas ao pensamento computacional. A avaliação conduzida e apresentada no Capítulo 4 demonstrou sua viabilidade para representação destes conhecimentos, além de sua eficiência para realização de inferências, sendo capaz de reproduzir fielmente, através de consultas SPARQL, a avaliação de pensamento computacional da ferramenta Dr. Scratch.

Além disso, a criação de uma representação de conhecimento que possibilite a representação destes fenômenos permite a análise temporal destes dados, algo que não era possível anteriormente. Através da representação de um ou mais projetos em diversos momentos de tempo, é possível verificar questões como quais habilidades o aluno está apresentando maior dificuldade em desenvolver ou até mesmo, ao agregar dados de vários alunos, realizar a mesma análise para uma turma inteira, permitindo ao educador ter apoio para o planejamento de suas aulas.

5.2 Trabalhos Futuros

Como uma nova representação de conhecimento para o acompanhamento do ensino de pensamento computacional através de projetos Scratch, o presente trabalho apresenta diversas possibilidades de trabalhos futuros.

Inicialmente, visualiza-se a necessidade de teste desta representação para armazenamento de um conjunto de dados reais, coletados através de uma oficina. Isto iria permitir a visualização do trabalho em um cenário de uso real.

Em adição, o principal trabalho futuro, é a conexão do mesmo com coletores de dados que capturam todas as ações realizadas pelo usuário no projeto, como o desenvolvido em (JUNIOR et al., 2019). Esta conexão facilitará a validação da representação com dados reais, como anteriormente mencionado, além de, através da sequência temporal de ações, ser possível reconstruir o estado do projeto em qualquer momento de tempo desejado e representá-lo com a OntoScratch.

Assim, através desta conexão, seria possível representar o projeto periodicamente, recriando o seu estado a cada intervalo de tempo desejado. Este fator permitiria a realização de inferências temporais mais facilmente, como, por exemplo, verificar a evolução de cada uma das habilidades de pensamento computacional através da execução de consultas como as apresentadas na Seção 4.2.

Além disso, outra possibilidade é a utilização da avaliação de progresso dos alunos para auxiliar o educador na tomada de decisão e planejamento dos conteúdos. Através do modelo desenvolvido, seria possível criar mecanismos que recomendariam para o educador quais habilidades deveriam ser reforçadas para uma turma com base nos dados armazenados, como o desempenho e o resultado das atividades dos alunos.

Todos estes fatores anteriores poderiam culminar no desenvolvimento de uma apli-

cação para suporte a decisão para o educador. Esta ferramenta poderia expor os dados ontológicos, permitindo a verificação de dados relativos ao desempenho da turma e de alunos específicos.

Além disso, ferramentas como (JUNIOR et al., 2019) realizam a coleta de ações tomadas no Scratch em tempo real. Assim, seria possível utilizar estes dados para, periodicamente, realizar a representação e avaliação na ontologia de um projeto enquanto este está sendo desenvolvido. Isto permitiria ao educador visualizar o progresso dos alunos em tempo real, possibilitando suporte também durante as atividades.

REFERÊNCIAS

ARAÚJO, C.; LIMA, L. V.; HENRIQUES, P. R. An Ontology based approach to teach Computational Thinking. In: INTERNATIONAL SYMPOSIUM ON COMPUTERS IN EDUCATION (SIIE), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–6.

AVILA, C.; CAVALHEIRO, S.; BORDINI, A.; MARQUES, M. O pensamento computacional por meio da robótica no ensino básico-uma revisao sistemática. In: BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO-SBIE), 2017. **Anais...** [S.l.: s.n.], 2017. v.28, n.1, p.82.

BARR, V.; STEPHENSON, C. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? **Acm Inroads**, [S.l.], v.2, n.1, p.48–54, 2011.

BIZER, C.; SCHULTZ, A. The berlin sparql benchmark. **International Journal on Semantic Web and Information Systems (IJSWIS)**, [S.l.], v.5, n.2, p.1–24, 2009.

BLIKSTEIN, P. O pensamento computacional e a reinvenção do computador na educação. **Education & Courses**, [S.l.], 2008.

BLIKSTEIN, P. Maker movement in education: History and prospects. **Handbook of Technology Education**, [S.l.], p.419–437, 2018.

BLINKSTEIN, P. **O pensamento computacional e a reinvenção do computador na educação. 2008**.

BOE, B. et al. Hairball: Lint-inspired static analysis of scratch projects. In: PROCEEDING OF THE 44TH ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 2013. **Anais...** [S.l.: s.n.], 2013. p.215–220.

BOMBASAR, J.; RAABE, A.; MIRANDA, E. M. de; SANTIAGO, R. Ferramentas para o ensino-aprendizagem do pensamento computacional: onde está Alan Turing? In: BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO-SBIE), 2015. **Anais...** [S.l.: s.n.], 2015. v.26, n.1, p.81.

BORDINI, A. et al. Pensamento computacional nos ensinos fundamental e médio: uma revisao sistemática. In: BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO-SBIE), 2017. **Anais...** [S.l.: s.n.], 2017. v.28, n.1, p.123.

BORST, W. N. Construction of engineering ontologies for knowledge sharing and reuse. , [S.l.], 1999.

BRENNAN, K.; CHUNG, M.; HAWSON, J. Creative computing: A design-based introduction to computational thinking. **Retrieved May**, [S.l.], v.9, p.2012, 2011.

BRENNAN, K.; RESNICK, M. New frameworks for studying and assessing the development of computational thinking. In: AMERICAN EDUCATIONAL RESEARCH ASSOCIATION, VANCOUVER, CANADA, 2012., 2012. **Proceedings...** [S.l.: s.n.], 2012. v.1, p.25.

BROEKSTRA, J. et al. Enabling knowledge representation on the web by extending RDF schema. In: WORLD WIDE WEB, 10., 2001. **Proceedings...** [S.l.: s.n.], 2001. p.467–478.

BROEKSTRA, J.; KAMPMAN, A.; VAN HARMELEN, F. Sesame: A generic architecture for storing and querying rdf and rdf schema. In: INTERNATIONAL SEMANTIC WEB CONFERENCE, 2002. **Anais...** [S.l.: s.n.], 2002. p.54–68.

BROWN, J. S.; BURTON, R. R. Multiple representations of knowledge for tutorial reasoning. In: **Representation and understanding**. [S.l.]: Elsevier, 1975. p.311–349.

CAMPOS, F.; SOSTER, T.; BLIKSTEIN, P. Sorry, I Was in Teacher Mode Today: Pivotal Tensions and Contradictory Discourses in Real-World Implementations of School Makerspaces. In: **Proceedings of FabLearn 2019**. [S.l.: s.n.], 2019. p.96–103.

CHANG, C.-K.; TSAI, Y.-T.; CHIN, Y.-L. A visualization tool to support analyzing and evaluating Scratch projects. In: IIAI INTERNATIONAL CONGRESS ON ADVANCED APPLIED INFORMATICS (IIAI-AAI), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.498–502.

CHRYSAFIADI, K.; VIRVOU, M. Evaluating the integration of fuzzy logic into the student model of a web-based learning environment. **Expert Systems with Applications**, [S.l.], v.39, n.18, p.13127–13134, 2012.

CHRYSAFIADI, K.; VIRVOU, M. Student modeling approaches: A literature review for the last decade. **Expert Systems with Applications**, [S.l.], v.40, n.11, p.4715–4729, 2013.

CLANCEY, W. The role of qualitative models in instruction. **Artificial Intelligence and human learning**, [S.l.], p.49–68, 1988.

CLEMENTE, J.; RAMÍREZ, J.; DE ANTONIO, A. A proposal for student modeling based on ontologies and diagnosis rules. **Expert Systems with Applications**, [S.l.], v.38, n.7, p.8066–8078, 2011.

COX, P. T. Visual programming languages. **Wiley Encyclopedia of Computer Science and Engineering**, [S.l.], p.1–10, 2007.

CSTA, K. Computer science standards. **Computer Science Teachers Association**, [S.l.], 2012.

DE FRANÇA, R. S.; AMARAL, H. J. C. do. Proposta metodológica de ensino e avaliaçao para o desenvolvimento do pensamento computacional com o uso do scratch. In: WORKSHOP DE INFORMÁTICA NA ESCOLA, 2013. **Anais...** [S.l.: s.n.], 2013. v.1, n.1, p.179.

DENNING, P. J.; MARTELL, C. H. **Great principles of computing**. [S.l.]: MIT Press, 2015.

DHARIWAL, S. BlockArt: Visualizing the'Hundred Languages' of Code in Children's Creations. In: **Proceedings of the 2019 on Creativity and Cognition**. [S.l.: s.n.], 2019. p.633–639.

DOUSAY, T. A. Defining and differentiating the makerspace. **Educational Technology**, [S.l.], p.69–74, 2017.

DRIGAS, A. S.; ARGYRI, K.; VRETTAROS, J. Decade review (1999-2009): artificial intelligence techniques in student modeling. In: WORLD SUMMIT ON KNOWLEDGE SOCIETY, 2009. **Anais...** [S.l.: s.n.], 2009. p.552–564.

FELLEISEN, M.; KRISHNAMURTHI, S. Viewpoint Why computer science doesn't matter. **Communications of the ACM**, [S.l.], v.52, n.7, p.37–40, 2009.

FENSEL, D. Ontologies. In: **Ontologies**. [S.l.]: Springer, 2001. p.11–18.

FEURZEIG, W.; PAPERT, S. A.; LAWLER, B. Programming-languages as a conceptual framework for teaching mathematics. **Interactive Learning Environments**, [S.l.], v.19, n.5, p.487–501, 2011.

GRAU, B. C. et al. OWL 2: The next step for OWL. **Journal of Web Semantics**, [S.l.], v.6, n.4, p.309–322, 2008.

GROVER, S.; PEA, R. Computational thinking in K–12: A review of the state of the field. **Educational researcher**, [S.l.], v.42, n.1, p.38–43, 2013.

GUARINO, N. **Formal ontology in information systems**: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy. [S.l.]: IOS press, 1998. v.46.

GUARINO, N.; OBERLE, D.; STAAB, S. What is an ontology? In: **Handbook on ontologies**. [S.l.]: Springer, 2009. p.1–17.

GUARINO, N.; WELTY, C. A. An overview of OntoClean. In: **Handbook on ontologies**. [S.l.]: Springer, 2004. p.151–171.

GUARINO, N.; WELTY, C. Evaluating ontological decisions with OntoClean. **Communications of the ACM**, [S.l.], v.45, n.2, p.61–65, 2002.

HAASE, P.; BROEKSTRA, J.; EBERHART, A.; VOLZ, R. A comparison of RDF query languages. In: INTERNATIONAL SEMANTIC WEB CONFERENCE, 2004. **Anais...** [S.l.: s.n.], 2004. p.502–517.

HAMIM, T.; BENABBOU, F.; SAEL, N. Toward a Generic Student Profile Model. In: THE PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON SMART CITY APPLICATIONS, 2019. **Anais...** [S.l.: s.n.], 2019. p.200–214.

HAMIM, T.; BENABBOU, F.; SAEL, N. Student profile modeling: an overview model. In: INTERNATIONAL CONFERENCE ON SMART CITY APPLICATIONS, 4., 2019. **Proceedings...** [S.l.: s.n.], 2019. p.1–9.

HEMMENDINGER, D. A plea for modesty. **Acm Inroads**, [S.l.], v.1, n.2, p.4–7, 2010.

HOLT, P.; DUBS, S.; JONES, M.; GREER, J. The state of student modelling. In: **Student modelling**: The key to individualized knowledge-based instruction. [S.l.]: Springer, 1994. p.3–35.

HORRIDGE, M.; MUSEN, M. Snap-SPARQL: A java framework for working with SPARQL and OWL. In: INTERNATIONAL EXPERIENCES AND DIRECTIONS WORKSHOP ON OWL, 2015. **Anais...** [S.l.: s.n.], 2015. p.154–165.

HORROCKS, I.; MCGUINNESS, D. L.; WELTY, C. A. Digital libraries and web-based information systems. In: **The description logic handbook**: theory, implementation, and applications. [S.l.: s.n.], 2003. p.427–449.

IQBAL, R. et al. An analysis of ontology engineering methodologies: A literature review. **Research journal of applied sciences, engineering and technology**, [S.l.], v.6, n.16, p.2993–3000, 2013.

ISTE, C. Computational Thinking in K–12 Education leadership toolkit. **Computer Science Teacher Association: http://csta. acm. org/Curriculum/sub/CurrFiles/471.11 CTLeadershipt Toolkit-SP-vF. pdf adresinden alındı**, [S.l.], 2011.

JEREMIĆ, Z.; JOVANOVIĆ, J.; GAŠEVIĆ, D. Student modeling and assessment in intelligent tutoring of software patterns. **Expert Systems with Applications**, [S.l.], v.39, n.1, p.210–222, 2012.

JUNIOR, J. L. N. V.; PRIMO, T. T.; PERNAS, A. M.; JÚNIOR, D. A. M. A Framework for Collecting and Analyzing Interactions in Scratch Projects. In: XIV LATIN AMERICAN CONFERENCE ON LEARNING TECHNOLOGIES (LACLO), 2019. **Anais...** [S.l.: s.n.], 2019. p.50–54.

KAY, J. Stereotypes, student models and scrutability. In: INTERNATIONAL CONFERENCE ON INTELLIGENT TUTORING SYSTEMS, 2000. **Anais...** [S.l.: s.n.], 2000. p.19–30.

LIU, C.-L. Using Bayesian networks for student modeling. In: **Cognitively Informed Systems**: Utilizing Practical Approaches to Enrich Information Presentation and Transfer. [S.l.]: Igi Global, 2006. p.283–311.

LYE, S. Y.; KOH, J. H. L. Review on teaching and learning of computational thinking through programming: What is next for K-12? **Computers in Human Behavior**, [S.l.], v.41, p.51–61, 2014.

MALONEY, J. et al. The scratch programming language and environment. **ACM Transactions on Computing Education (TOCE)**, [S.l.], v.10, n.4, p.1–15, 2010.

MARTINS, A. C.; FARIA, L.; DE CARVALHO, C. V.; CARRAPATOSO, E. User modeling in adaptive hypermedia educational systems. **Journal of Educational Technology & Society**, [S.l.], v.11, n.1, p.194–207, 2008.

MAYER, R. E. Cognitive views of creativity: Creative teaching for creative learning. **Contemporary educational psychology**, [S.l.], v.14, n.3, p.203–211, 1989.

MAYO, M. J. Bayesian student modelling and decision-theoretic selection of tutorial actions in intelligent tutoring systems. , [S.l.], 2001.

MCGUINNESS, D. L.; VAN HARMELEN, F. et al. OWL web ontology language overview. **W3C recommendation**, [S.l.], v.10, n.10, p.2004, 2004.

MORENO-LEÓN, J.; ROBLES, G.; ROMÁN-GONZÁLEZ, M. Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. **RED. Revista de Educación a Distancia**, [S.l.], n.46, p.1–23, 2015.

NGUYEN, L.; DO, P. Learner model in adaptive learning. **World Academy of Science, Engineering and Technology**, [S.l.], v.45, n.70, p.395–400, 2008.

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 and . . . , 2001.

NUNES, D. J. Computação ou informática. **Jornal da Ciência**, [S.l.], v.30, 2010.

OLUK, A.; KORKMAZ, Ö. Comparing Students' Scratch Skills with Their Computational Thinking Skills in Terms of Different Variables. **Online Submission**, [S.l.], v.8, n.11, p.1–7, 2016.

PAPERT, S. **Situating Constructionism. I. Harel, & S. Papert (Eds), Constructionism. Norwood**. [S.l.]: NJ: Ablex Publishing. http://web. media. mit. edu/˜ calla/web_comunidad . . . , 1991.

PEARL, J. **Probabilistic reasoning in intelligent systems**: networks of plausible inference. [S.l.]: Elsevier, 2014.

PÉREZ, J.; ARENAS, M.; GUTIERREZ, C. Semantics and complexity of SPARQL. **ACM Transactions on Database Systems (TODS)**, [S.l.], v.34, n.3, p.1–45, 2009.

PERNAS, A. M.; DANTAS, M. Ontologia Aplicadas à Descrição de Recursos em Ambientes Grid. **INFOCOMP Journal of Computer Science**, [S.l.], v.3, n.2, p.26–31, 2004.

QI, P.; SUN, Y.; LUO, H.; GUIZANI, M. Scratch-DKG: A Framework for Constructing Scratch Domain Knowledge Graph. **IEEE Transactions on Emerging Topics in Computing**, [S.l.], 2020.

RESNICK, M. Give P'sa chance: Projects, peers, passion, play. In: CONSTRUCTIONISM AND CREATIVITY: PROCEEDINGS OF THE THIRD INTERNATIONAL CONSTRUCTIONISM CONFERENCE. AUSTRIAN COMPUTER SOCIETY, VIENNA, 2014. **Anais. . .** [S.l.: s.n.], 2014. p.13–20.

RESNICK, M. et al. Scratch: programming for all. **Communications of the ACM**, [S.l.], v.52, n.11, p.60–67, 2009.

RICH, E. User modeling via stereotypes. **Cognitive science**, [S.l.], v.3, n.4, p.329–354, 1979.

RIVERS, R. Embedded user models—where next? **Interacting with Computers**, [S.l.], v.1, n.1, p.13–30, 1989.

SCRATCH Wiki. Acessado em 07/07/2020, `https://en.scratch-wiki.info/`.

SCRATCH. Acessado em 11/07/2020, https://scratch.mit.edu/statistics/.

SEITER, L.; FOREMAN, B. Modeling the learning progressions of computational thinking of primary grade students. In: ACM CONFERENCE ON INTERNATIONAL COMPUTING EDUCATION RESEARCH, 2013. **Proceedings...** [S.l.: s.n.], 2013. p.59–66.

SEYMOUR, P. Mindstorms; Children, Computers and Powerful Ideas. **New York: Basic Book**, [S.l.], 1980.

SIRIN, E. et al. Pellet: A practical owl-dl reasoner. **Journal of Web Semantics**, [S.l.], v.5, n.2, p.51–53, 2007.

STANSFIELD, J. L.; CARR, B. P.; GOLDSTEIN, I. P. Wumpus advisor 1: A first implementation program that tutors logical and probabilistic reasoning skills. , [S.l.], 1976.

STATHACOPOULOU, R.; MAGOULAS, G. D.; GRIGORIADOU, M.; SAMARAKOU, M. Neuro-fuzzy knowledge processing in intelligent learning environments for improved student diagnosis. **Information Sciences**, [S.l.], v.170, n.2-4, p.273–307, 2005.

STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge engineering: principles and methods. **Data & knowledge engineering**, [S.l.], v.25, n.1-2, p.161–197, 1998.

TROIANO, G. M. et al. Is My Game OK Dr. Scratch? Exploring Programming and Computational Thinking Development via Metrics in Student-Designed Serious Games for STEM. In: ACM INTERNATIONAL CONFERENCE ON INTERACTION DESIGN AND CHILDREN, 18., 2019. **Proceedings...** [S.l.: s.n.], 2019. p.208–219.

TSIRIGA, V.; VIRVOU, M. Initializing the student model using stereotypes and machine learning. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2002. **Proceedings...** [S.l.: s.n.], 2002.

TUCKER, A. A Model Curriculum for K–12 Computer Science: Final Report of the ACM K–12 Task Force Curriculum Committee. , [S.l.], 2003.

VAN HEIJST, G.; SCHREIBER, A. T.; WIELINGA, B. J. Using explicit ontologies in KBS development. **International journal of human-computer studies**, [S.l.], v.46, n.2-3, p.183–292, 1997.

WILSON, A.; HAINEY, T.; CONNOLLY, T. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In: EUROPEAN CONFERENCE ON GAMES BASED LEARNING, 2012. **Anais...** [S.l.: s.n.], 2012. p.549.

WING, J. M. Computational thinking. **Communications of the ACM**, [S.l.], v.49, n.3, p.33–35, 2006.

WINTER, M.; BROOKS, C. A.; GREER, J. E. Towards Best Practices for Semantic Web Student Modelling. In: AIED, 2005. **Anais...** [S.l.: s.n.], 2005. p.694–701.

YADAV, A.; GOOD, J.; VOOGT, J.; FISSER, P. Computational thinking as an emerging competence domain. In: **Competence-based vocational and professional education**. [S.l.]: Springer, 2017. p.1051–1067.

ZANETTI, H.; BORGES, M.; RICARTE, I. Pensamento computacional no ensino de programação: uma revisão sistemática da literatura brasileira. In: BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO-SBIE), 2016. **Anais...** [S.l.: s.n.], 2016. v.27, n.1, p.21.

Apêndices

APÊNDICE A – Consultas SPARQL

Neste apêndice, serão apresentadas as consultas SPARQL utilizadas para inferência do nível computacional em um projeto Scratch. Estas consultas seguem a metodologia de avaliação utilizada pela ferramenta Dr. Scratch e são utilizadas para a avaliação do projeto, detalhada no Capítulo 4.

A.1 Abstração

Listagem A.1 – Consulta para inferência da habilidade “Abstração”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
    ?abstraction ctc:ConceptScore ?score
}
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?abstraction .
  ?abstraction rdf:type ctc:Abstraction .

  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .

  # # Rule 1
    OPTIONAL{
        ?project scratch:HasSprite ?level_1_sprite_1 .
        ?project scratch:HasSprite ?level_1_sprite_2 .

        ?project scratch:owns ?level_1_block_1 .
        ?level_1_block_1 scratch:Scripts ?level_1_sprite_3 .
```

```
      ?project scratch:owns ?level_1_block_2 .
      ?level_1_block_2 scratch:Scripts ?level_1_sprite_4 .
  }
# Rule 2
  OPTIONAL{
      ?project scratch:owns ?level_2_block .
      ?level_2_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          Define()Block
      >
  }
# Rule 3
  OPTIONAL{
      ?project scratch:owns ?level_3_block .
      {
          ?level_3_block rdf:type <
              http://www.semanticweb.org/nicol/ontologies/scratch/
              CreateCloneOf()Block>
      } UNION
      {
          ?level_3_block rdf:type <
              http://www.semanticweb.org/nicol/ontologies/scratch/
              DeleteThisCloneBlock
          >
      } UNION
      {
          ?level_3_block rdf:type <
              http://www.semanticweb.org/nicol/ontologies/scratch/
              WhenIStartAsACloneBlock
          >
      }
  }

  BIND(
    if(
      BOUND(?level_1_sprite_1) && BOUND(?level_1_sprite_2) &&
      BOUND(?level_1_block_1) && BOUND(?level_1_block_2),
      !sameterm(?level_1_sprite_1, ?level_1_sprite_2) &&
      !sameterm(?level_1_block_1, ?level_1_block_2),
```

```
        FALSE
      ) AS ?has_level_1
    )
  BIND(BOUND(?level_2_block) AS ?has_level_2)
  BIND(BOUND(?level_3_block) AS ?has_level_3)

  BIND(
    if(?has_level_3,
      3,
      if(?has_level_2,
        2,
        if(?has_level_1,
          1,
          0
  ))) AS ?score)
}
```

A.2 Controle de Fluxo

Listagem A.2 – Consulta para inferência da habilidade “Controle de Fluxo”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
  ?flow_control ctc:ConceptScore ?score
}
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?flow_control .
  ?flow_control rdf:type ctc:FlowControl .

  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .

  # Rule 1
  OPTIONAL{
```

```
        ?project scratch:owns ?block .
        ?project scratch:owns ?level_1_block .
        ?block scratch:Connects ?level_1_block
    }
  # Rule 2
  OPTIONAL{
        ?project scratch:owns ?level_2_block .
        { ?level_2_block rdf:type scratch:ForeverBlock } UNION
        {
            ?level_2_block rdf:type <
                http://www.semanticweb.org/nicol/ontologies/scratch/
                Repeat()Block
            >
        }
    }
  # Rule 3
  OPTIONAL{
        ?project scratch:owns ?level_3_block .
        ?level_3_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            RepeatUntil()Block
        >
    }

  BIND(BOUND(?level_1_block) AS ?has_level_1)
  BIND(BOUND(?level_2_block) AS ?has_level_2)
  BIND(BOUND(?level_3_block) AS ?has_level_3)

  BIND(
    if(?has_level_3,
      3,
      if(?has_level_2,
        2,
        if(?has_level_1,
          1,
          0
  ))) AS ?score)
}
```

A.3 Interatividade com Usuário

Listagem A.3 – Consulta para inferência da habilidade “Interatividade com Usuário”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT { ?user_interactivity ctc:ConceptScore ?score }
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?user_interactivity .
  ?user_interactivity rdf:type ctc:UserInteractivity .

  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .

  # Rule 1
  OPTIONAL{
    ?project scratch:owns ?level_1_block .
    ?level_1_block rdf:type scratch:WhenGreenFlagClickedBlock
  }

  # Rule 2
  OPTIONAL{
    ?project scratch:owns ?level_2_block .
    {
        ?level_2_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            When()KeyPressedBlock
        >
    } UNION
    {
        ?level_2_block rdf:type scratch:WhenThisSpriteClickedBlock>
    } UNION
    {
        ?level_2_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            Ask()AndWaitBlock
```

```
    >
} UNION
{
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        PointTowards()Block
    > .
    ?level_2_block scratch:hasInput ?input .
    ?input rdf:type scratch:Value .
    ?input scratch:FreeInput "mouse-pointer"
} UNION
{
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Glide()SecsTo()Block
    > .
    ?level_2_block scratch:hasInput ?input .
    ?input rdf:type scratch:Value .
    ?input scratch:FreeInput "mouse-pointer"
} UNION
{
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Touching?Block
    > .
    ?level_2_block scratch:hasInput ?input .
    ?input rdf:type scratch:Value .
    ?input scratch:FreeInput "mouse-pointer"
} UNION
{
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        DistanceTo()Block
    > .
    ?level_2_block scratch:hasInput ?input .
    ?input rdf:type scratch:Value .
    ?input scratch:FreeInput "mouse-pointer"
} UNION
{
```

```
      ?level_2_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          MouseDown?Block
      >
  } UNION
  {
    ?level_2_block rdf:type scratch:MouseXBlock
  } UNION
  {
    ?level_2_block rdf:type scratch:MouseYBlock
  }
}
# Rule 3
OPTIONAL{
?project scratch:owns ?level_3_block .
  {
      ?level_3_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          When()GreaterThan()Block
      >
  } UNION
  {
      ?level_3_block_1 rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          PlaySound()UntilDoneBlock
      >
  } UNION
  {
      ?level_3_block_1 rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          StartSound()Block
      >
  } UNION
  {
      ?level_3_block_1 rdf:type scratch:StopAllSoundsBlock
  } UNION
  {
      ?level_3_block_1 rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
```

```
            Change()EffectBy()Block
        >
    } UNION
    {
        ?level_3_block_1 rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            Set()EffectTo()Block
        >
    } UNION
    {
        ?level_3_block_1 rdf:type scratch:ClearSoundEffectsBlock
    } UNION
    {
        ?level_3_block_1 rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            ChangeVolumeBy()Block
        >
    } UNION
    {
        ?level_3_block_1 rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            SetVolumeTo()PercentageBlock
        >
    } UNION
    {
        ?level_3_block_1 rdf:type scratch:VolumeBlock
    }
  }

BIND(BOUND(?level_1_block) AS ?has_level_1)
BIND(BOUND(?level_2_block) AS ?has_level_2)
BIND(BOUND(?level_3_block) AS ?has_level_3)

BIND(
  if(?has_level_3,
    3,
    if(?has_level_2,
      2,
      if(?has_level_1,
```

```
        1,
        0
))) AS ?score)
}
```

A.4 Paralelismo

Listagem A.4 – Consulta para inferência da habilidade “Paralelismo”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
  ?parallelism ctc:ConceptScore ?score
}
WHERE {
  SELECT ?evaluation ?project ?parallelism (MAX(?score) AS ?score)
  WHERE {
    ?evaluation rdf:type ctc:CTEvaluation .
    ?evaluation ctc:evaluates ?parallelism .
    ?parallelism rdf:type ctc:Parallelism .

    ?evaluation ctc:uses ?project .
    ?project rdf:type scratch:Project .

    # Rule 1
    OPTIONAL{
      ?project scratch:owns ?green_flag_1, ?green_flag_2 .
      ?green_flag_1 rdf:type scratch:WhenGreenFlagClickedBlock .
      ?green_flag_2 rdf:type scratch:WhenGreenFlagClickedBlock
    }

    # Rule 2
    OPTIONAL{
      ?project scratch:owns ?level_2_block_1, ?level_2_block_2 .
      {
        ?level_2_block_1 rdf:type <
```

```
        http://www.semanticweb.org/nicol/ontologies/scratch/
        When()KeyPressedBlock
    > .
    ?level_2_block_2 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        When()KeyPressedBlock
    >
  } UNION
  {
    ?level_2_block_1 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenThisSpriteClickedBlock
    > .
    ?level_2_block_2 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenThisSpriteClickedBlock
    >
  } UNION
  {
    ?level_2_block_1 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenThisSpriteClickedBlock
    > .
    ?level_2_block_2 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenThisSpriteClickedBlock
    > .
    ?level_2_project scratch:owns ?sprite .
    ?sprite rdf:type scratch:Sprite .
    ?sprite scratch:owns ?level_2_block_1 .
    ?sprite scratch:owns ?level_2_block_2
  }
}
# Rule 3
OPTIONAL{
?project scratch:owns ?level_3_block_1 .
  {
    ?project scratch:owns ?level_3_block_2 .
    ?level_3_block_1 rdf:type <
```

```
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenIReceive()Block
    > .
    ?level_3_block_2 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenIReceive()Block
    >
  } UNION
  {
    ?level_3_block_1 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        CreateCloneOf()Block
    >
  } UNION
  {
    ?project scratch:owns ?level_3_block_2 .
    ?level_3_block_1 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        When()GreaterThan()Block
    > .
    ?level_3_block_2 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        When()GreaterThan()Block
    >
  } UNION
  {
    ?project scratch:owns ?level_3_block_2 .
    ?level_3_block_1 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenBackdropSwitchesTo()Block
    > .
    ?level_3_block_2 rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        WhenBackdropSwitchesTo()Block
    >
  }
}

BIND(
```

```
      if(
        BOUND(?green_flag_1) && BOUND(?greenflag_2),
        !sameterm(?green_flag_1, ?greenflag_2),
        FALSE
      ) AS ?has_level_1
    )
    BIND(
        if(
          BOUND(?level_2_block_1) && BOUND(?level_2_block_2),
          !sameterm(?level_2_block_1, ?level_2_block_2),
          FALSE
        ) AS ?has_level_2
    )
    BIND(
        if(
          BOUND(?level_3_block_1) && BOUND(?level_3_block_2),
          !sameterm(?level_3_block_1, ?level_3_block_2),
          FALSE
        ) AS ?has_level_3
    )

    BIND(
      if(?has_level_3,
        3,
        if(?has_level_2,
          2,
          if(?has_level_1,
            1,
            0
    ))) AS ?score)
  }
  GROUP BY ?evaluation ?project ?paralleslism
}
```

A.5 Pensamento Lógico

Listagem A.5 – Consulta para inferência da habilidade “Pensamento Lógico”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
  ?logical_thinking ctc:ConceptScore ?score
}
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?logical_thinking .
  ?logical_thinking rdf:type ctc:LogicalThinking .

  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .

  # Rule 1
  OPTIONAL{
    ?project scratch:owns ?level_1_block .
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        If()ThenBlock
      >
  }
  # Rule 2
  OPTIONAL{
    ?project scratch:owns ?level_2_block .
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        If()ThenElseBlock
    >
  }
  # Rule 3
  OPTIONAL {
    ?project scratch:owns ?level_3_block .
    {
        ?level_3_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            ()And()Block
```

```
        >
    } UNION
    {
        ?level_3_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            ()Or()Block
        >
    } UNION
    {
        ?level_3_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            Not()Block
        >
    }
  }

  BIND(BOUND(?level_1_block) AS ?has_level_1)
  BIND(BOUND(?level_2_block) AS ?has_level_2)
  BIND(BOUND(?level_3_block) AS ?has_level_3)

  BIND(
    if(?has_level_3,
      3,
      if(?has_level_2,
        2,
        if(?has_level_1,
          1,
          0
  ))) AS ?score)
}
```

A.6 Representação de Dados

Listagem A.6 – Consulta para inferência da habilidade “Representação de Dados”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
CONSTRUCT { ?data_representation ctc:ConceptScore ?score }
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?data_representation .
  ?data_representation rdf:type ctc:DataRepresentation .

  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .

  # Rule 1
  OPTIONAL{
    ?project scratch:owns ?level_1_block .
    {
        ?level_1_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            Move()StepsBlock
        >
    } UNION
    {
        ?level_1_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            Turn()DegreesBlock
        >
    } UNION
    {
        ?level_1_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            GoTo()Block
        >
    } UNION
    {
        ?level_1_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            GoToX()Y()Block
        >
    } UNION
    {
        ?level_1_block rdf:type <
```

```
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Glide()SecsTo()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Glide()SecsToX()Y()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        PointInDirection()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        PointTowards()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        ChangeXBy()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        SetXTo()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        ChangeYBy()Block
    >
```

```
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        SetYTo()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        SwitchCostumeTo()Block
    >
} UNION
{
    ?level_1_block rdf:type scratch:NextCostumeBlock
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        SwitchBackdropTo()Block
    >
} UNION
{
    ?level_1_block rdf:type scratch:NextBackdropBlock
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        ChangeSizeBy()Block
    >
} UNION
{
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        SetSizeTo()PercentageBlock
    >
} UNION
{
    ?level_1_block rdf:type <
```

```
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Change()EffectBy()Block
    >
  } UNION
  {
    ?level_1_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Set()EffectTo()Block
    >
  } UNION
  {
    ?level_1_block rdf:type scratch:ClearGraphicEffectsBlock
  } UNION
  {
    ?level_1_block rdf:type scratch:ShowBlock
  } UNION
  {
    ?level_1_block rdf:type scratch:HideBlock
  }
}

# Rule 2
OPTIONAL{
  ?project scratch:owns ?level_2_block .
  {
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Change()By()Block
    >
  } UNION
  {
    ?level_2_block rdf:type <
        http://www.semanticweb.org/nicol/ontologies/scratch/
        Set()To()Block
    >
  }
}
# Rule 3
OPTIONAL{
```

```
    ?project scratch:owns ?level_3_block .
    ?level_3_block rdf:type scratch:ListsType
  }

BIND(BOUND(?level_1_block) AS ?has_level_1)
BIND(BOUND(?level_2_block) AS ?has_level_2)
BIND(BOUND(?level_3_block) AS ?has_level_3)

BIND(
  if(?has_level_3,
    3,
    if(?has_level_2,
      2,
      if(?has_level_1,
        1,
        0
))) AS ?score)
}
```

A.7 Sincronização

Listagem A.7 – Consulta para inferência da habilidade “Sincronização”

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
  ?synchronization ctc:ConceptScore ?score
}
WHERE {
  ?evaluation rdf:type ctc:CTEvaluation .
  ?evaluation ctc:evaluates ?synchronization .
  ?synchronization rdf:type ctc:Synchronization .

  ?evaluation ctc:uses ?project .
  ?project rdf:type scratch:Project .
```

```
# Rule 1
OPTIONAL{
  ?project scratch:owns ?level_1_block .
  ?level_1_block rdf:type <
      http://www.semanticweb.org/nicol/ontologies/scratch/
      Wait()SecsBlock
  >
}

# Rule 2
OPTIONAL{
  ?project scratch:owns ?level_2_block .
  {
      ?level_2_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          Broadcast()Block
      >
  } UNION
  {
      ?level_2_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          WhenIReceive()Block>
  } UNION
  # Add inputs at Stop()Block
  {
      ?level_2_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          Stop()Block
      >
  }
}
# Rule 3
OPTIONAL{
  ?project scratch:owns ?level_3_block .
  {
      ?level_3_block rdf:type <
          http://www.semanticweb.org/nicol/ontologies/scratch/
          WaitUntil()Block
      >
```

```
    } UNION
    {
        ?level_3_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            WhenBackdropSwitchesTo()Block
        >
    } UNION
    {
        ?level_3_block rdf:type <
            http://www.semanticweb.org/nicol/ontologies/scratch/
            Broadcast()AndWaitBlock
        >
    }
  }

  BIND(BOUND(?level_1_block) AS ?has_level_1)
  BIND(BOUND(?level_2_block) AS ?has_level_2)
  BIND(BOUND(?level_3_block) AS ?has_level_3)

  BIND(
    if(?has_level_3,
      3,
      if(?has_level_2,
        2,
        if(?has_level_1,
          1,
          0
  ))) AS ?score)
}
```

A.8 Pontuação Total

Listagem A.8 – Consulta para cálculo da pontuação total

```
PREFIX ctc:
   <http://www.semanticweb.org/nicol/ontologies/2020/4/CTConcepts/>
PREFIX scratch: <http://www.semanticweb.org/nicol/ontologies/scratch/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
SELECT ?evaluation (SUM(?score) AS ?total_score)
WHERE {
  ?concept rdf:type ctc:CTConcept .
  ?concept ctc:ConceptScore ?score .
  ?evaluation ctc:evaluates ?concept
}
GROUP BY ?evaluation
```