UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS DE FRANCISCO BELTRÃO
CURSO DE LICENCIATURA EM INFORMÁTICA

GUSTAVO HENRIQUE BELTRANI SLOMSKI

HELLOARDUBOT: UMA LINGUAGEM DE PROGRAMAÇÃO ESPECÍFICA PARA O ENSINO DE ROBÓTICA COM TECNOLOGIA LIVRE

TRABALHO DE CONCLUSÃO DE CURSO

FRANCISCO BELTRÃO
2021

GUSTAVO HENRIQUE BELTRANI SLOMSKI

HELLOARDUBOT: UMA LINGUAGEM DE PROGRAMAÇÃO ESPECÍFICA PARA O ENSINO DE ROBÓTICA COM TECNOLOGIA LIVRE

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Informática da Universidade Tecnológica Federal do Paraná, como requisito para a obtenção do título de Licenciado em Informática.

Orientador: Prof. Dr. Michel Albonico
Coorientadora: Profa. Dra. Maici Duarte Leite

FRANCISCO BELTRÃO
2021

RESUMO

SLOMSKI, Gustavo. HelloArduBot: Uma Linguagem de Programação Específica para o Ensino de Robótica com Tecnologia Livre. 2021. 54 f. Trabalho de Conclusão de Curso – Curso de Licenciatura em Informática, Universidade Tecnológica Federal do Paraná. Francisco Beltrão, 2021.

A programação é pré-requisito para desenvolvimento de projetos e também ferramenta de ensino-aprendizagem para plataforma Arduíno. Entretanto, programar nem sempre é uma tarefa fácil. Assim, o presente trabalho tem como objetivo o desenvolvimento de uma linguagem de domínio específico para auxiliar no processo de ensino de robótica. Tendo como base a abstração de conceitos de linguagens convencionais aliados a uma sintaxe simples e em português, essa ferramenta traz inovação e auxílio para o ensino de robótica.
Palavras-chave: Robótica. Linguagem de domínio específico. Arduino. Programação.

ABSTRACT

SLOMSKI, Gustavo. OiArduBot: A specific language for teaching open source robotics. 2021. 54 f. Trabalho de Conclusão de Curso – Curso de Licenciatura em Informática, Universidade Tecnológica Federal do Paraná. Francisco Beltrão, 2021.

Programming is a prerequisite for the development of projects and also a tool of teaching-learning for Arduino platform. However, programming isn't always an easy task. So, this work aims to develop a domain-specific language to assist in the robotics teaching process. Based on the abstraction of concepts from conventional languages and combined with a simple syntax in portuguese, this tool brings innovation and assistance to the teaching of robotics.

Keywords: Robotics. Domain Specific Language. Arduino. Programming.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

TIC	Tecnologias da Informação e Comunicação
UML	Linguagem de Modelagem Unificada
DSL	Linguagem de Domínio Específico
AST	Árvore de Sintaxe Abstrata
EMF	Framework de Modelagem Eclipse
IDE	Ámbiente de Desenvolvimento Integrado
MDE	Engenharia de Software Orientada a Modelos
EPL	Licença Pública Eclipse
RAP	Plataforma de Aplicação Remota
RE	Robótica Educacional

SUMÁRIO

1 INTRODUÇÃO

Os equipamentos eletrônicos fazem parte do nosso cotidiano. No entanto, quando considera-se a história da humanidade, esses equipamentos são recentes. A partir da década de 1970, começa-se a ouvir sobre a introdução dos computadores na educação, bem como a preocupação com o uso das tecnologias como ferramenta de apoio no processo de ensino (ATAÍDE; MESQUITA, 2014). Nesse momento, em que o acesso à informação e as tecnologias torna-se mais palpável, abrem-se portas para experiências mediante a utilização do computador em universidades (VALENTE, 1999). Ainda, em meados de 1980, deu-se início a construção de diversas iniciativas que visavam a inserção das tecnologias no âmbito escolar. Assim, surge um novo termo, Tecnologia da Informação e Comunicação (TIC), que refere-se às diferentes formas de tecnologia que, integradas, possibilitam a comunicação entre pessoas e processos, como o compartilhamento da informação no contexto educacional (MENDES, 2008).

A tecnologia, quando em ambiente educacional, pode auxiliar os estudantes no processo de ensino aprendizagem em tomada de decisões e solução de problemas; comunicadores e colaboradores criativos; pessoas que buscam, analisam e avaliam a informação. Assim, estes tornam-se cidadãos bem informados. No entanto, para que os estudantes façam bom uso da tecnologia, é preciso que sejam preparados. Freire (2009) defendia uma educação que instigasse a consciência crítica, libertadora, promovendo o aluno a ler e entender o mundo. Criticando a educação em formato bancário, onde ocorre a memorização mecânica de conteúdos, encorajava a criatividade do educando e do educador, promovendo o processo de aprendizagem e autonomia por meio da troca de saberes. A interação é necessária para aprendizagem, permitindo a construção e reconstrução do saber e, proporcionando um diálogo reflexivo, do qual emerge a consciência de maneira crítica à realidade.

Em meio a esse contexto de autonomia e construção do conhecimento, os robôs educacionais têm adentrado e se apresentam como soluções versáteis e multidisciplinares. Chamada de robótica educacional, a robótica na sala de aula dá mais autonomia ao aluno, que passa a idealizar e construir protótipos em um ambiente de aprendizagem atrativo e palpável. Desde a década de 80 (HAREL; PAPERT, 1991), a robótica educacional vem sendo empregada no ensino de programação de computadores em escolas e universidades. Baseada no construcionismo de Harel e Papert (1991), em que acreditasse no protagonismo do aluno no processo de construção do conhecimento, a robótica educacional traz perspectivas ainda pouco exploradas pelas práticas de ensino.

Nesse cenário, um robô responde a comandos programados no computador (i.e, automação), tornando o resultado da programação mais perceptivo ao aluno. De fato, para que os robôs funcionem, precisam ser programados. Usando-os como caso de uso, é possível que haja uma maior abstração dos conceitos repassados em disciplinas de programação.

1.1 Problema de Pesquisa

Tendo em vista a relevância da robótica educacional, pode-se destacar duas principais plataformas para prática de programação que intensificam a interação da ferramenta com o aluno, e, instigam a construção do conhecimento de forma a entregá-lo mais autonomia: Arduino[1] e LEGO Mindstorms[2]. O LEGO conta com programação em blocos. Contudo, tem um custo elevado[3] em relação ao Arduino[4], podendo-se estimar o valor de um kit Arduíno em aproximadamente 5% do valor de um kit LEGO Mindstorms, dificultando assim o seu acesso. Por sua vez, o Arduino é uma plataforma eletrônica de código aberto, na qual se pode realizar a prototipagem e programação de diversos projetos. Ainda, trás como diferencial seu baixo custo. Há uma grande quantidade de projetos utilizando Arduíno em seu *hub* de projetos[5], dando indícios de ser uma abordagem válida no trabalho com robótica.

A grande questão no que se refere ao Arduino, é a programação. O Arduino conta com uma linguagem de baixo nível, semelhante a C/C++, apresentando sintaxe em inglês e exigindo do usuário um nível razoável de lógica de programação, essa característica acaba por dificultar a aprendizagem de alguns alunos.

Portanto, vemos que a programação de robôs educacionais pode ser feita de duas maneiras: por linguagem de blocos ou por linguagem de programação convencional. Assim, inicialmente o aluno aprende conceitos básicos de lógica de programação e depois se atém à sintaxe de uma linguagem de programação. No entanto, mesmo em disciplina de programação para a graduação, é comum a dificuldade em assimilar essa transição, principalmente devido à grande diferença entre os dois paradigmas. Para um aprendizado cognitivo e autônomo, como é proposto na metodologia construtivista (PIAGET, 2013) e construcionista (PAPERT, 1993), é importante uma linguagem intermediária, que proporcione maior gradatividade ao processo de ensino-aprendizagem de programação.

Na literatura, há linguagens de programação com esse objetivo, como o Portugol IDE (MANSO; OLIVEIRA; MARQUES, 2009), que simplifica a programação e é escrita em português. No entanto, não são linguagens específicas para robótica, nem permitem a compilação para a automação de robôs. Também há linguagens de programação específicas/sistemas para robótica, que são menos complexas e generalistas que linguagens de programação convencionais, mas que atendem a propósitos específicos como robôs industriais (ARNOLD; HENRIQUES; FONSECA, 2002) e enxames de robôs (PINCIROLI; BELTRAME, 2016). Por fim, há a ferramenta RoboEduc (THOMAZ et al., 2009), que permite níveis de abstração na programação de robôs. Entretanto, a evolução da linguagem de programação no RoboEduc acontece de maneira automática entre os diversos níveis, impossibilitando uma percepção apurada das

[1] Website do Arduino: <https://www.arduino.cc/>
[2] Website do Mindstorms: <https://www.lego.com/en-gb/themes/mindstorms>
[3] LEGO Mindstorms: <https://www.amazon.com/LEGO-MINDSTORMS-31313-Educational-Programming/dp/B00CWER3XY>
[4] Kit Arduino: <https://www.filipeflop.com/produto/kit-maker-arduino-iniciante>
[5] Arduino Proejct Hub: <https://create.arduino.cc/projecthub>

mudanças que ocorrem de um nível ao outro, e impedindo que o alunos crie seus próprios algoritmos de computadores de maneira autônoma. Fatores como a independência para criação, suscitando a liberdade e criatividade, são peças fundamentais para um boa execução e aproveitamento do processo de construção e aprendizagem, seguindo princípios de uma abordagem construcionista (DARGAINS; SAMPAIO, 2015).

1.2 Objetivos

Com base no problema de pesquisa, este trabalho tem os seguintes objetivos.

1.2.1 Objetivo Geral

Desenvolver uma linguagem de domínio específico (DSL) para a programação de robôs educacionais intermediária à linguagem de blocos e a linguagem de programação do Arduino.

1.2.2 Objetivos Específicos

- Modelar a sintaxe da linguagem com Eclipse Modeling Framework (EMF) para definir seus limites funcionais;
- Criar tradutor para a linguagem a fim de possibilitar que o código feito em *OiArduBot* possa ser enviado para o Arduino;
- Investigar a eficácia do uso da linguagem em alunos do curso de Licenciatura em informática e escolas.

1.3 Justificativa

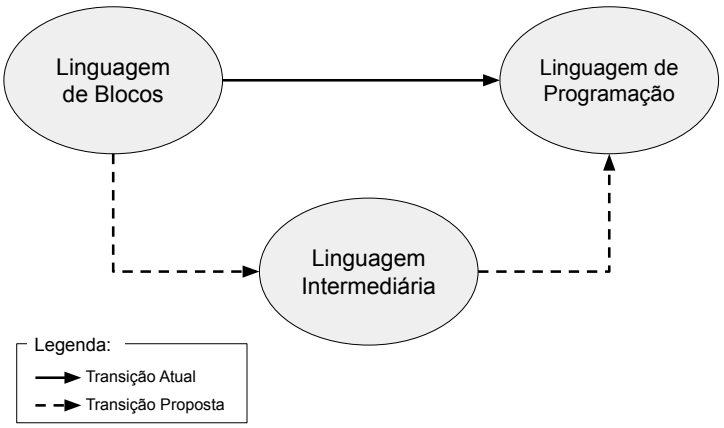
No curso de Licenciatura em Informática, da UTFPR, Câmpus Francisco Beltrão, há duas linhas principais: ciências humanas e ciência da computação. Os conteúdos de programação são uma novidade para quem adentra na faculdade, e, agregado aos fatores de sintaxe complexa e quantidade de conceitos envolvidos, essa atividade acaba se tornando uma matéria difícil mediante essas inquisições (GOMES; HENRIQUES; MENDES, 2008; JENKINS, 2002). O uso de robôs abstrai complexidades, relacionando conceitos teóricos com movimentos no solo, por exemplo, o que facilita a abstração e, por consequência, o aprendizado. Um facilitador é que no curso a Robótica Educacional é bem difundida, como no ensino de disciplinas básicas (e.g., Arquitetura de Computadores), bem como em projetos de extensão nas escolas.

Pela questão do custo-benefício e pela liberdade de modelagem, o Arduino tem se tornado o principal artefato na aplicação da Robótica Educacional dentro do Câmpus. No entanto, há uma demanda quanto à programação do mesmo para iniciantes em programação, por exemplo, no ensino de Algoritmos de Programação, que é o primeiro contato dos estudantes com programação. Uma abordagem bastante difundida é a programação usando blocos, como o

Scratch[6], que busca ensinar lógica de programação com um ambiente de alto nível de abstração. Contudo, após a linguagem de blocos, normalmente os alunos passam para uma linguagem de programação convencional. Assim, há uma quebra de paradigmas muito grande, sendo importante o uso de uma linguagem intermediária.

A Figura 1 ilustra a mudança de paradigma na aprendizagem de linguagens de programação com robótica educacional. No processo atual, o aluno inicialmente aprende a programar em blocos, onde adquire conceitos básicos sobre lógica e fluxo de programação. Em seguida, parte para a linguagem convencional do Arduino, tendo que lidar com conceitos de programação ainda não introduzidos. Na Figura 1, o novo fluxo de aprendizagem, retratado pela linha tracejada, propõe a aplicação de uma linguagem intermediária que permitirá uma mudança gradual de paradigma. Apesar de atualmente ser embasada no Arduino, por ter sua natureza dirigida a modelos, a linguagem apresenta um processo facilitado na migração para outras plataformas abertas.

Figura 1 – Etapas do processo de aprendizagem de programação.



Fonte: Autor

1.4 Organização do Trabalho

Esse trabalho está divido da seguinte maneira: seção 2 revisa a literatura abordando sobre robótica educacional e linguagens de domínio específico. A seguir, ocorre a apresentação de materiais e métodos na Seção 3. Na Seção 4 são elencados os resultados obtidos até então com o projeto. Por fim, a Seção 5 faz o fechamento do estudo e conclui com alguns apontamentos relevantes sobre o trabalho.

[6] Website do Scratch: https://scratch.mit.edu/

2 REVISÃO DE LITERATURA

A Robótica Educacional é uma abordagem didática que engloba diferentes metodologias, mas embasa-se fortemente no construcionismo. Para um melhor entendimento da aplicação de nossa linguagem, nesta seção, entenderemos os pilares do construcionismo e quais conceitos dão sustento para aplicação da robótica educacional em sala de aula.

2.1 Construcionismo

O Construcionismo foi baseado no Construtivismo de Jean Piaget (PIAGET, 2013), que basicamente considera que a aprendizagem é uma construção, e que deve haver diferentes ferramentas de interação para que essa construção possa ocorrer, onde manipular artefatos é a chave para a criança construir o seu conhecimento. Outro aspecto importante é que o professor passa a ser um mediador do processo de ensino-aprendizagem e o aluno passa a ser mais ativo no processo de aprendizagem.

Baseando-se no Construtivismo, Seymour Papert idealizou o construcionismo (HAREL; PAPERT, 1991), que coloca o aluno como projetista e construtor de objetos tecnológicos, normalmente robôs. O construcionismo busca meios de aprendizagem onde o indivíduo/aluno tem como base a sua própria concepção de mundo. Assim, o aluno precisa interagir com o objeto de estudo, viver experiências de tal maneira que a convivência e o meio possam proporcionar a ele aquisição de conhecimento. O estudo de Papert teve como foco principal o ensino de crianças (PAPERT, 1993), mas tem sido pensado e aplicado em várias etapas do currículo escolar (MINSKY; PAPERT, 2017; HENSON, 2015; BOUDOURIDES, 1998).

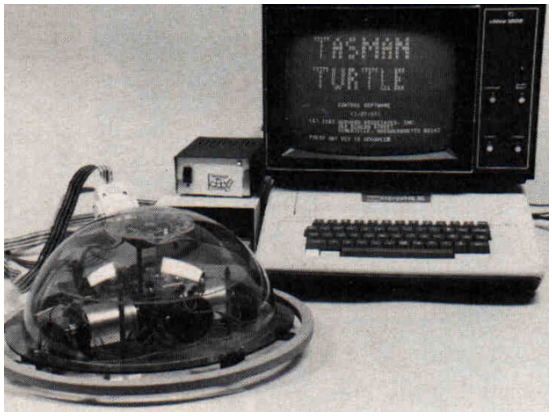
2.2 Robótica Educacional

Apesar de ter sido proposto há décadas, o Construcionismo tem se mostrado uma abordagem bastante atual (ALI et al., 2019; DANIELA; LYTRAS, 2018; SPANGSBERG; BRYNSKOV, 2018). Projetos de ensino de robótica atuais ainda seguem a linguagem educacional LOGO (FEURZEIG et al., 1970), que implementa as ideias construcionistas de Papert, como projetos com Turtlebot (AMSTERS; SLAETS, 2019). O Turtlebot[1] é um robô educacional moderno que imita a tartaruga robô do projeto LOGO, além do LEGO Mindstoms ter derivado desse projeto também.

A Figura 2 ilustra a primeira versão do robô proposto pela equipe do Instituto de Tecnologia de Massachusetts (MIT), que Papert fazia parte, e Figura 3 ilustra uma versão comercial atual, comercializada pela empresa Willow Garage[2].

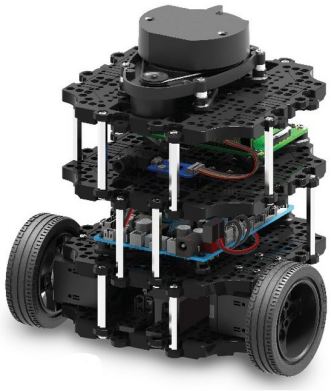
[1] Website do Turtlebot: <http://www.robotis.us/turtlebot-3/>
[2] <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>

Figura 2 – Protótipo inicial do Logo Turtle.



Fonte: (SEYMOUR, 1968)

Figura 3 – Modelo do Turtlebot3.



Fonte: (OVERVIEW, 2020)

Além deles, há muitos trabalhos que ainda evidenciam a importância de projetos práticos no ensino de disciplinas diversas. A construção de robôs simples tem sido comum nesse tipo de atividade (ZHONG; XIA, 2020; SILVA et al., 2020; EGUCHI, 2014; BENITTI, 2012), dando-lhe o nome de Robótica Educacional. A seguir, descrevemos brevemente cada um deles.

2.2.1 Trabalhos de Robótica Educacional

A robótica educacional (RE) é um campo que tem evoluído cada vez mais. Existem inúmeros trabalhos que buscam exemplificar a aplicabilidade da RE na sala de aula, muitos desses, dando indícios de eficácia. A partir desse contexto, partimos para o levantamento de trabalhos que denotam, através de estudos, a implementação da RE como parte do processo de ensino aprendizagem.

Segundo Eguchi (2014), é de suma importância a integração da robótica educacional em forma de ferramenta de aprendizagem no ambiente de ensino. A robótica educacional traz aos alunos opções de criações inovadoras e os instiga a se tornarem ativos na construção de novas tecnologias e não apenas meros consumidores. Ela é capaz de integrar várias disciplinas e prover ao aluno uma maneira diferente de aprendizagem, na qual, ao construir um robô, ele emprega conteúdos adquiridos em sala de modo estimulante. Pensando nessa integração, a RoboParty[3] é um acampamento com duração de 3 dias que tem como objetivo ensinar e aprender a construir robôs autônomos de maneira lúdica. Para tanto, cada equipe recebe um *Bot'n Roll One A*, um kit de robótica baseado em Arduino. Os desafios são realizados em equipe, tendo como princípio a prática. Assim, os participantes são submetidos a tentativas e erros até que seu robô funcione perfeitamente. O intuito não é simplesmente obter um resultado, mas também entender processo o de construção dele.

[3] Website RoboParty: <https://www.roboparty.org/>

Segundo Zhong e Xia (2020), embora a Robótica Educacional seja flexível o suficiente para servir de ferramenta de ensino dos mais variados conteúdos, ainda há a necessidade de explorar esse potencial, estritamente no ensino de matemática. Com base no levantamento realizado pelos autores, as tentativas futuras da aplicação da robótica no ensino da matemática devem prestar atenção nas interações entre robô e aluno, pensando em resolver problemas matemáticos do mundo real assistidos pela robótica que, permite ao aluno desenvolver uma experiência prática duradoura. Contudo, o estudo descreve um quadro promissor para aplicação da robótica. Assim, tendo em vista a aplicabilidade diversificada da robótica nesse campo, entende-se que ela traz grandes perspectivas de auxílio no processo de ensino aprendizagem de matemática.

Um projeto desenvolvido por Silva et al. (2020) demonstra a aplicabilidade da robótica em sala de aula. O mesmo consistia em um curso com duração de 3 meses, especificamente para alunos do 6º ao 9º ano do ensino fundamental. Utilizando a plataforma Arduino e ferramentas como o *Scratch*, o projeto buscava auxiliar alunos com dificuldades em matérias de exatas. As aulas práticas eram realizadas em grupos e auxiliadas por um professor, que relacionava cada aparato do projeto à conceitos educacionais e proporcionava aos alunos a ajuda necessária para desenvolvimento do protótipo. Com base nos resultados obtidos após a finalização do curso, foi realizada uma análise para verificar a eficiência do uso da robótica como ferramenta pedagógica, trazendo resultados relevantes para esse trabalho. Ainda, os autores destacam que:

> o uso da robótica, em conjunto com a linguagem criativa do Scratch, proporciona, com muita facilidade, não só uma ampla área de conhecimento a respeito do tema, mas, também, eles geram inovações que podem estar presentes no dia a dia e que podem ser implementadas no ensino de forma satisfatória, trazendo resultados relevantes.

Portanto, Silva et al. (2020) ainda evidenciam a facilidade que o Arduino propõe na portabilidade de funções e destacam as condições favoráveis de aplicação no ensino público brasileiro que ele tem, devido ao seu baixo custo.

Com base no trabalho de Benitti (2012), grande parte das aplicações da robótica em sala de aula tem focado intimamente em disciplinas relacionadas à robótica. Além de operar de maneira passiva, ela tem sido pouco inovadora, mantendo-se em abordagens tradicionalistas que causam desinteresse em parte do seu público. Ainda assim, a robótica demonstra ser uma abordagem viável. Entretanto, se faz necessário analisar diferentes estratégias de aplicação, de modo a envolver os alunos com os mais diversos interesses. Segundo a pesquisa, ainda existe a indispensabilidade de trazer significado na utilização da robótica em sala de aula. Entretanto, existem trabalhos que evidenciam a viabilidade do uso da robótica para ensinar em áreas que não são estritamente vinculadas a robótica. Portanto, é justo dizer que a robótica apresenta um grande potencial como ferramenta de auxílio no processo de ensino aprendizagem.

2.3 Linguagens para Programação de Robôs

O LEGO *Mindstorms* é uma das plataformas para robótica educacional mais difusas atualmente. Essa linha de robôs, vem comumente com um controlador acompanhado de diversas peças, como sensores e atuadores. Ela dispõe de um *software* para desenvolvimento de seus códigos. O desenvolvimento dos programas é baseado em blocos e apresenta sintaxe em inglês. Apesar de ser em língua estrangeira, a plataforma não apresenta muitos problemas quanto a isso, devido a existência de inúmeros tutoriais prontos. Os blocos contam com funções predefinidas, as quais podem ser configuradas e acopladas a fim de gerar funcionalidades maiores e mais complexas para o robô.

Também muito presente no contexto da robótica educacional, o Arduino é uma plataforma de prototipagem e desenvolvimento de projetos baseada em *hardware* livre. O Arduino consiste em uma placa composta por um microcontrolador (e demais componentes eletrônicos para funcionamento) e pinos de entrada e saída para comunicação. Ele dispõe de uma grande compatibilidade de componentes, sensores e atuadores, o que o torna tão competitivo nesse meio. O Arduino pode ser programado de duas maneiras principais: linguagem de programação convencional ou programação em blocos.

A linguagem de programação convencional do Arduino é baseada no C e C++, duas linguagens de programação extremamente conhecidas no âmbito da informática. Essa linguagem é de baixo nível e apresenta sintaxe em inglês. Devido a essas características, para desenvolver um projeto utilizando Arduino é necessário um nível razoável de programação, apontando assim uma das suas principais dificuldades.

Ardublock é uma ferramenta de programação em blocos para Arduino. Por sua vez, ela consiste em blocos de código que podem ser encaixados uns aos outros a fim de desenvolver um fluxo de programação. Esses blocos apresentam funcionalidades prontas e, alguns deles, necessitam ou permitem alteração de parâmetros. A sintaxe também é inglês. A abordagem é bastante eficaz, porém limitada as funções predefinidas dos blocos, impossibilitando ir além.

Segundo Manso, Oliveira e Marques (2009), o Portugol IDE é um ambiente de aprendizagem em português que dispõe de duas linguagens, uma linguagem algorítmica e outra fluxográfica. Vamos nos deter apenas à primeira. Ainda, o autor destaca que a linguagem algorítmica faz uso do português estruturado, comumente conhecido como pseudo-código ou portugol, para descrever de forma clara ao computador como executar determinadas instruções. Esse ambiente de aprendizagem se mostra bastante intuitivo e consolidado. Entretanto, vale salientar que essa plataforma se limita ao desenvolvimento de algoritmos para computador.

O Quadro 1 mostra um comparativo entre as linguagens apresentadas, levando em consideração os requisitos que consideramos relevantes para nosso trabalho. No quadro, vemos que nenhuma delas atende todos os requisitos.

Quadro 1 – Comparação das linguagens.

Linguagem/Ferramenta	Sintaxe em Português	Tipo	Customização de Funções	Para Robótica
ArduBlock	Não	Blocos	Não	Sim
LEGO Mindstorms	Não	Blocos	Não	Sim
Linguagem do Arduino	Não	Convencional	Sim	Sim
Portugol	Sim	Simplificada	Sim	Não

2.4 Linguagem de Domínio Específico (DSL)

Segundo Deursen, Klint e Visser (2000), uma linguagem de domínio específico é "uma linguagem de programação ou linguagem de especificação executável que oferece, por meio de notações e abstrações apropriadas, poder expressivo focado e normalmente restrito a um domínio de problema particular". Sendo assim, o uso de linguagens de domínio específico pode otimizar a realização de determinada tarefa, uma vez que é focada em um objetivo específico, diferentemente das demais linguagens com abordagens distintas.

Com base no trabalho de Riquelmo et al. (2019), pode-se notar que a utilização de DSL se dá nos mais diversos temas. Visando contribuir com o processo de desenvolvimento de modelagem conceitual de banco de dados, Riquelmo et al. (2019) propõe uma DSL para auxiliar nessa atividade. Com base em alguns requisitos definidos no decorrer do trabalho e referenciais teóricos, o autor destaca que o uso de DSL pode otimizar o tempo e a produtividade de uma tarefa, fato que ocorre devido a abstração dos métodos de realização da atividade, exigindo conhecimento de um material mais enxuto para desenvolvimento da mesma. O trabalho foi desenvolvido utilizando o plugin Xtext do Eclipse *IDE* e resultou em um protótipo de linguagem de domínio específico para auxílio na modelagem conceitual de banco de dados.

No âmbito da robótica, pode-se destacar a linguagem de domínio específico RoboX (HONDA; JUMES, 2005). Esse trabalho teve como objetivo o desenvolvimento de uma DSL para programação de robôs usando a ferramenta Visual Studio DSL Tools. O resultado foi uma linguagem bastante sólida, abstraindo diversos conceitos de programação e possibilitando que até mesmo iniciantes na programação consigam desenvolver projetos com ela. O trabalho apresenta diversos projetos passiveis de serem desenvolvidos usando RoboX.

Elenca-se alguns pontos que essas DSLs, especificamente a RoboX, não compreendem. A documentação da linguagem é um quesito faltante. Mesmo que bem estruturada, a RoboX ainda apresenta uma quantidade pequena de funcionalidades, limitando a criatividade do seu usuário. Outro ponto importante é o tratamento de erros, qual não é realizado de maneira eficaz, capaz de impedir que erros na programação ocorram antes da compilação (HONDA; JUMES, 2005). Por fim, a linguagem não gera um código compatível com outros robôs comerciais. Isso significa que ela está isolada dentro dos limites de desenvolvimento do seu ambiente.

2.5 Considerações sobre os trabalhos relacionados

A robótica educacional tem se apresentado como uma abordagem eficaz dentro do ambiente educacional. Tendo em vista sua preocupação em colocar o aluno como agente ativo no processo de aprendizagem, ela traz uma didática pouco aplicada no ensino público vigente. Entretanto, esse princípio ainda carece de aplicações e estudos bem estruturados e relevantes.

Muitos trabalhos de robótica educacional remetem a conceitos e até a própria programação de robôs. Essa ampla abordagem, é a porta de entrada para inúmeros trabalhos que buscam auxiliar no processo de ensino aprendizagem de programação. É nesse âmbito que o presente projeto se insere. O uso de DSL na robótica educacional ainda é vago. Sem muitos projetos, a *OiArduBot* se apresenta como uma proposta inovadora no contexto. Visando preservar os conhecimentos já adquiridos e iniciar o aluno na apropriação de conceitos da linguagem de programação convencional do Arduino, este trabalho apresenta fatores relevantes no auxílio ao processo de aprendizagem de programação.

3 MATERIAS E MÉTODOS

Os materiais e métodos empregados no desenvolvimento desse trabalho, foram elencados com base nas competências desenvolvidas no curso e outras trabalhadas durante o processo de construção do TCC, com auxílio do orientador. Os conceitos de meta-modelos e linguagens de domínio específico foram alguns dos desafios superados durante as etapas de desenvolvimento da solução proposta neste trabalho. Buscou-se por ferramentas livres, de fácil acesso e que não tenham custo, para que a linguagem possa ser melhorada em projetos futuros dentro da universidade.

3.1 Materiais

O Arduino é a base da nossa pesquisa, sendo a plataforma onde se executa o software desenvolvido na linguagem OiArdubot. Para o desenvolvimento da linguagem *OiArduBot* foi necessário a utilização de uma *IDE* (Ambiente de Desenvolvimento Integrado, traduzido para português), tendo sido escolhido o Eclipse [1], por este prover uma gama de ferramentas/*frameworks* para criação de linguagens de domínio específico, os quais também permitem o desenvolvimento de plugins, o que saluta para evolução futura do projeto. Ao que se refere a plataforma, a linguagem OiArduBot foi desenvolvida com os *frameworks Xtext* e *Xtend*, disponíveis na IDE Eclipse.

Todos os materiais utilizados no desenvolvimento da linguagem são descritos com mais detalhes na sequência.

3.1.1 Arduino

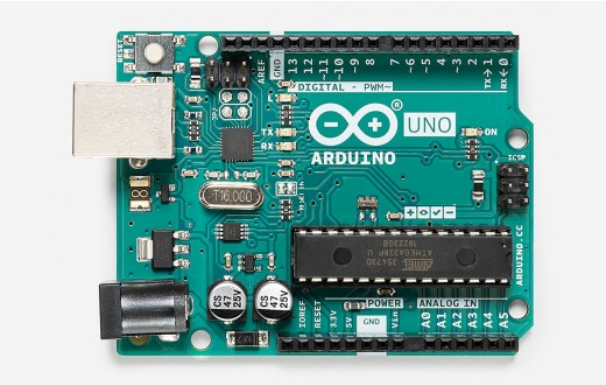
De forma geral, o Arduino[2] é uma placa eletrônica de baixo custo, como a ilustrada na Figura 7, que possibilita o desenvolvimento de diversos projetos eletrônicos. O Projeto Arduino propõe o conceito de hardware livre, que remete a projetos eletrônicos baseados nas premissas do software livre e disponibilizados, normalmente de forma gratuita, para a comunidade. No caso do Arduino, uma gama desses projetos pode ser encontrada no Arduino Project Hub[3].

Grande parte dos projetos no Arduino Project Hub são de robótica, baseando-se em dispositivos com sensores e atuadores, os quais são facilmente acoplados às placas Arduino por meio de sua estrutura de portas (ou pinos). Basicamente, sensores são circuitos eletrônicos que captam informações do ambiente, enquanto atuadores são as partes do robô que interagem com o ambiente externo, normalmente em forma de movimento ou som (MCKERROW; MCKERROW, 1991). Por exemplo, um sensor poderia capturar a luminosidade de um ambiente

[1] Website do Eclipse *IDE*: <https://www.eclipse.org/>
[2] Website do Projeto Arduino: <https://www.arduino.cc/>
[3] Arduino Project Hub: <https://create.arduino.cc/projecthub>

e quando ela diminuir a um certo nível, um atuador acionaria o interruptor para ligar uma lâmpada.

Figura 4 – Placa Arduino modelo UNO.



Fonte: Arduino Website

3.1.1.1 Projetos de Robôs Educacionais com Arduino

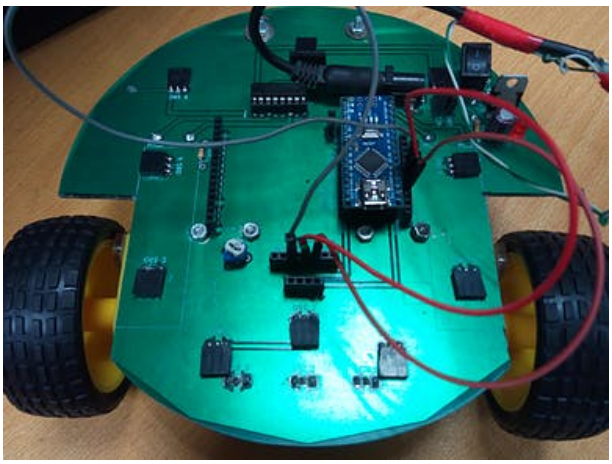
O Arduino tem ganhado notoriedade no âmbito educacional (SILVA et al., 2020; AGUIAR, 2017), principalmente por seu custo reduzido e gama de dispositivos sensores e atuadores suportados. Para fins de ilustração, nesta seção são mostrados dois exemplos de robôs educacionais escolhidos pelo autor no Arduino Project Hub.

Robô educacional de baixo custo[4]: Conforme a síntese do projeto, alguns países em desenvolvimento, como é o caso do Paquistão, apresentam dificuldades na aquisição de robôs para sala de aula, devido ao seu orçamento limitado. Assim, o projeto tem como objetivo desenvolver um robô de baixo custo e proporcionar sua aplicação no processo educacional. A Figura 5 mostra o robô proposto no projeto, que é simples, mas tem uma proposta chamativa. Suas funcionalidades consistem em mover-se para frente e para trás por meio de um motor (atuador), detectar luminosidade e calcular distâncias por meio de sensores.

BecDot braille[5]: O projeto propõe um robô para ensino de braile para crianças com deficiência visual, mostrado na Figura 6. O robô espera que a criança coloque sobre ele um objeto entre 4 (objetos) pré-definidos, que tenham o nome composto por 4 letras ou menos. O robô possui dois tipos de sensores, um que identifica qual é o objeto selecionado, e outro que identifica quais são os dígitos em braille digitados pela criança, dando uma resposta de som quando ela acertar a escrita do nome do objeto selecionado.

[4] Robô educacional de baixo custo: <https://create.arduino.cc/projecthub/utk6533/>
[5] BecDot braille: <https://create.arduino.cc/projecthub/jacob-lacourse/>

Figura 5 – Robô educacional de baixo custo.



Fonte: Arduino Project Hub

Figura 6 – Robô para ensino de braille.

Fonte: Arduino Project Hub

3.1.2 Framework de Modelagem do Eclipse (EMF)

A Estrutura de Modelagem Eclipse (EMF)[6] consiste em um conjunto de ferramentas com objetivo de gerar código por intermédio de modelos. Ela é composta por duas partes principais, o *core framework* que serve para criação de classes e implementação e o *EMF.edit*, que serve para criação e edição dos modelos. Em um aspecto geral, o EMF permite transformar modelos em estruturas denominadas metamodelos *ECore*, seguindo uma base hierárquica. Um metamodelo *Ecore*, pode provir de um modelo definido com auxilio da EMF. Posteriormente, o EMF permite a geração de código com base na estrutura ECore criada. Assim, o paradigma do EMF é simples e objetivo: criação de um metamodelo, que consiste basicamente em um subconjunto de diagramas UML (Unified Modeling Language - Linguagem de Modelagem Unificada) e que possibilitem sua transformação em código. A UML é uma linguagem de

[6] Site EMF: <https://www.eclipse.org/modeling/emf/>

modelagem largamente utilizada na área de desenvolvimento de software, possibilitando a criação de diversos diagramas que descrevem modelos, como, por exemplo, diagramas de caso de uso, que servem para ilustrar as interações que as funcionalidades do sistemas podem exercer entre si e com o usuário.

Meta-modelos Ecore: no EMF, o metamodelo, ou modelo de modelo, objetiva a análise e desenvolvimento de um modelo que estabeleça regras e afins, conduzindo a construção de um sistema específico. É importante destacarmos que o metamodelo não se dá somente como parte da documentação do software, na qual pode-se extrair informações fundamentais do programa, como funcionalidades, mas também como implementação (BORELLI; CARVALHO, 2016), servindo como base para o desenvolvimento. A modelagem de um sistema pode ocorrer de diferentes formas, e o desenvolvimento de modelos, ou diagramas, como são comumente conhecidos na área de engenharia de software, são peças fundamentais para dirigir o processo de desenvolvimento. Sob demanda do EMF, conseguimos construir esses metamodelos. Nesse ponto, eles são concebidos com intuito de modelar as classes e relações que a linguagem terá.

Assim, como dito anteriormente, podemos transformar o metamodelo em parte da nossa linguagem, reaproveitando suas hierarquias e relações. Entretanto, nem sempre a transformação do modelo nos dá uma estrutura 100% eficaz, sendo necessárias algumas alterações no código. Resumidamente, o EMF é uma estrutura que permite a criação e manipulação de metamodelos, permitindo assim a extração de requisitos e funcionalidades para criação e validação de uma futura linguagem de programação.

3.1.3 Xtext

O Xtext[7] é um framework para desenvolvimento de linguagens de programação e linguagens de domínio específico. Isso se dá por meio de uma linguagem incorporada ao framework. Essa linguagem permite que o desenvolvedor defina uma gramática e, posteriormente, gere um modelo de classe. O intuito é que se produza uma sintaxe concreta, buscando representar o modelo semântico da mesma. A linguagem gramatical é a pedra angular do Xtext (Eclipse Foundation, 2020). Uma grande vantagem relacionada ao Xtext é a sua integração com a Eclipse IDE, possibilitando trabalhar com outros recursos, como *plugins*, que agregam praticidade e velocidade no desenvolvimento do projeto.

Como descrito, o EMF possibilita a transformação do metamodelo para código. Esse processo consiste na importação do metamodelo que irá retornar como saída um código em Xtext, esse que por sua vez atende as restrições e relações assumidas no modelo. Todas as definições feitas até então no metamodelo, agora são inferidas dentro de uma gramática. Assim, podemos afirmar que o Ecore se equipara ao código desenvolvido em Xtext e, que podemos denominá-lo como uma representação gráfica do modelo.

As classes que foram definidas no metamodelo, bem como suas relações e cardinalidades são agora repassadas ao Xtext. De tal forma, temos agora em mãos parte da linguagem

[7] Website do Xtext: <https://www.eclipse.org/Xtext/>

implementada em Xtext, o que nos possibilita a realização de ajustes mais específicos e também a agregação de novas funcionalidades.

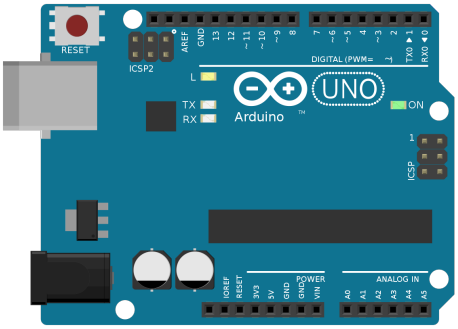
3.1.4 Xtend

Xtend[8] é uma linguagem de tipagem estática desenvolvida por Sven Efftinge, Sebastian Zarnekow, tomando como base o Java. A linguagem tem como objetivo gerar código, porém em outra linguagem. Seu enfoque é geração de código Java, entretanto, tem se mostrado perspicaz e efetiva para outras linguagens. De modo geral, o Xtend faz a tradução de um código de entrada em um código de saída em outra linguagem específica. Ele é o intermédio entra a gramática do OiArduBot e o código que o Arduino espera receber.

Com base nesses materiais, é possível escrevermos um código extremamente simples em OiArduBot e carregarmos ele para uma placa Arduino, obtendo sua operação.

Para carregar o código gerado no robô utiliza-se a Arduino *IDE*. Este trabalho foca especificamente na placa Arduino Uno (Figura 7), que é intermediária e permite o desenvolvimento de projetos avançados.

Figura 7 – Modelo Arduino Uno.



Fonte: (ARDUINOUNO, 2018)

3.2 Métodos

A linguagem *OiArduBot* é modelada com o Eclipse Modeling Framework (EMF), onde define-se toda a informação essencial do modelo textual (gramática da linguagem). A gramática é implementada em Xtext.

A Figura 8 ilustra o processo de edição e compilação do *OiArduBot*. O código (arquivo '.oab') é escrito em um editor próprio, que faz a análise sintática/gramatical em tempo real. Então, ele é transformado em linguagem C++ para Arduino (arquivo '.ino'), que pode ser executado diretamente nos robôs educacionais.

[8] Website do Xtend: <https://www.eclipse.org/xtend/>

Figura 8 – Edição e compilação da linguagem *OiArduBot*.

Fonte: Autor

3.2.1 Engenharia Dirigida a Modelos/Model-driven Engineering

Com o crescente avanço do desenvolvimento de sistemas, bem como sua complexidade, se faz necessária a aplicação de uma ferramenta capaz de auxiliar nesse processo. Assim, modelar se torna uma prática essencial dentro desse ambiente. A engenharia dirigida por modelos (Model-driven Engineering - MDE) é uma abordagem que conduz o processo de desenvolvimento de software por intermédio de modelos (GADELHA et al., 2016). Essa prática já vem há tempo sendo empregada dentro da engenharia de software e, seu uso tem consolidado cada vez mais a necessidade de adoção de práticas como essa no âmbito de desenvolvimento de software (BUARQUE, 2009). Maior produtividade, manutenibilidade, interoperabilidade e menor custo de evolução de software são algumas das vantagens oportunizadas pela MDE. Um estudo evidência esses benefícios ao comparar a construção de um mesmo software usando duas abordagens distintas: a primeira usando MDE e a segunda aplicando o desenvolvimento convencional com orientação a objetos (BUARQUE, 2009).

O MDE propõe que o desenvolvimento de software se dê por meio de modelos. Assim, toda a abstração conceitual e especificações se dariam pelo modelo, tendo em vista um processo de execução para implementação do software. O processo de desenvolvimento de software dirigido por modelo é bem intuitivo, se baseando totalmente no desenvolvimento, reutilização e transformação de modelos. A definição de um modelo é o primeiro passo. O modelo deve representar a estrutura, funcionalidades e possíveis comportamentos do sistema. A partir do modelo, partimos para o segundo passo: a compilação. O modelo por si só não dispõe de compilação, assim como no desenvolvimento convencional de software, que é dado por linguagens de alto nível, é necessária a presença de um compilador para realizar a tradução das informações a ele concedidas. Assim, com base em um mapeamento do modelo, a MDE propõe a tradução do modelo para uma linguagem de alto nível. Independente da linguagem é possível a reutilização do modelo; pode-se usar o modelo para gerar um escopo de sistema em Java, bem como posso fazer o mesmo para C++. Um último passo precisa ser dado antes de chegar ao resultado, os ajustes e adaptações do código. Por existirem poucas ferramentas que implementem a MDE e também por sua implementação, é necessário que em alguns casos se façam ajustes no código, quesitos comuns de programação, e adaptações, esse se destaca quando há a reutilização de vários modelos e se faz necessária a integração dos mesmos.

A partir desse ponto, onde temos o código-fonte, damos sequência como em qualquer

outro sistema: compilação e execução.

A implementação da MDE no processo de desenvolvimento além de oportunizar a sua aplicação em diferentes linguagens de programação, serve como documentação do sistema.

3.2.2 Modelagem da Linguagem

O Xtext possui uma grande infraestrutura de integração com a Eclipse IDE, desde analisador de código até as características mais comuns de uma IDE. É importante ressaltar que o Xtext depende do EMF e utiliza seus modelos como representação na memória, seja qual for o arquivo de texto analisado (MOURA et al., 2020). Assim, a medida que o analisador do Xtext consome a gramática, ele gera instâncias de modelos Ecore na memória [9], os quais representam uma árvore de sintaxe abstrata (ou Abstract Syntax Tree - AST). Árvores de sintaxe abstrata simbolizam a estrutura simplificada de uma determinada linguagem de programação. Não se detém a pequenos detalhes como, por exemplo, palavras obrigatórias em uma função, mas sim com a estrutura lógica e funcional da linguagem, trazendo a ideia especificamente de uma árvore estrutural da linguagem. Existe relação entre a gramática definida no Xtext e os modelos gerados pelo EMF. Isso é possível graças à integração disposta pela IDE, a qual acaba por gerar um ecossistema de desenvolvimento completo de DSL. A partir da gramática definida, podemos extrair do Xtext um metamodelo. A linguagem usada para aquisição do metamodelo é denominada Ecore, i.e, é em si o modelo da nossa linguagem. No Xtext, os metamodelos são inferidos a partir da gramática[10] ou importados a partir de um modelo. Como descrito anteriormente, ele é parte fundamental do EMF.

O modelo representa as estruturas e funcionalidades que o sistema deve conter. Nem sempre as limitações descritas no modelo são levadas a risca, i. e, as regras definidas na modelagem não são atendidas de maneira integral.

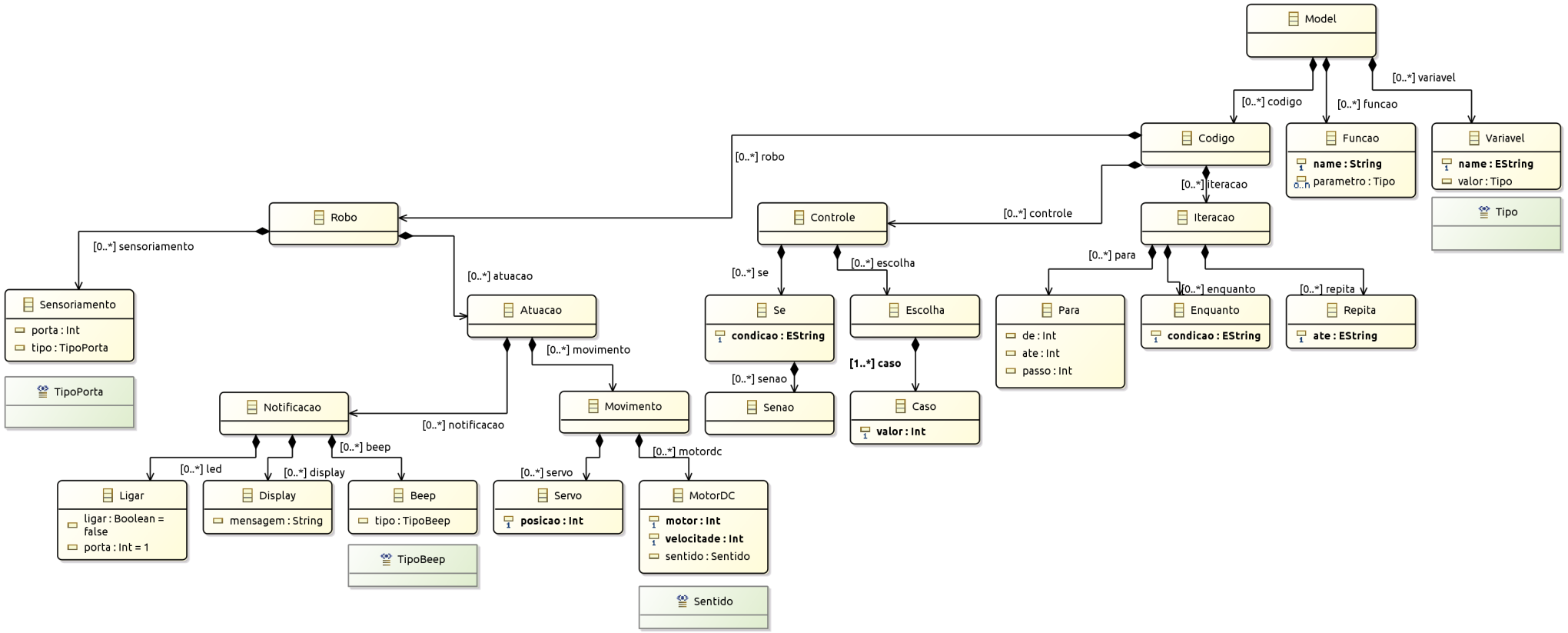
No presente trabalho, optamos por seguir a modelagem, na medida do possível, respeitando a divisão de classes imposta pelo modelo. O modelo criado representa as classes e relações que a gramática do sistema deve tomar como base. O modelo atual demonstra a estrutura definida na linguagem. É notável que existe uma relação entre as classes e também que baseia-se em uma hierarquia, na qual todas as classes estão abaixo do modelo principal. O intuito dessa representação é abstrair conceitos sintáticos e usar o modelo para fins de validação e interpretação, proporcionando uma visão *topdown* da linguagem.

A figura 9 apresenta o modelo da linguagem acima descrito. Esse modelo foi gerado usando o EMF. Reforçando, a modelagem serve como base para a construção da nossa linguagem, uma vez que ela define as regras e interações dela. Como pode-se notar, o modelo, muito semelhante ao diagrama de classes da UML, consiste em um diagrama que implementa as classes e relações da gramática. As representações feitas aqui serão refletidas diretamente no Xtext. Essas classes serão posteriormente transformadas em código, em um processo

[9] <https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html#metamodel-inference>
[10] EMF<https://www.eclipse.org/Xtext/documentation/308_emf_integration.html>

simples e objetivo, como descrito anteriormente. Para tanto, o metamodelo da linguagem é de tal importância que é o documento que dá capacidade à criação da linguagem. A seguir abordaremos cada uma das partes do modelo.

Figura 9 – Modelo usado para gerar a sintaxe da linguagem *OiArduBot*.

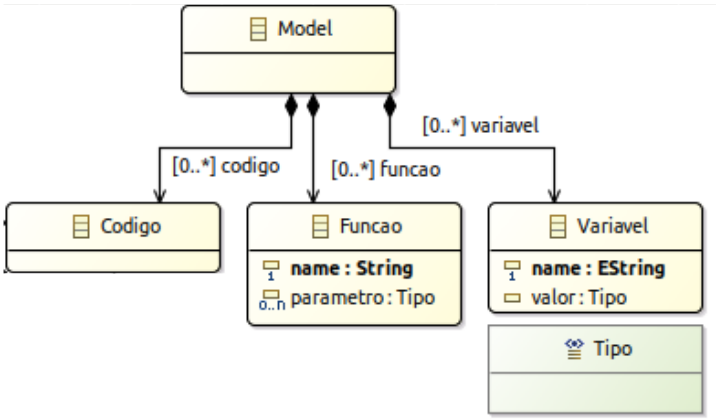


Fonte: Autor

3.2.2.1 Modelagem das Partes Principais da Linguagem

Para o desenvolvimento da OiArduBot tomamos como premissa que todas as classes são submissas a uma classe principal, que ocupa o topo da hierarquia (bloco *Model*), como demonstra a figura 10. A base é composta por três blocos (classes): **Codigo**, **Funcao** e **Variavel**. O **Codigo** é a classe que possibilita a escrita de código na nossa linguagem; é com base nela que definimos o que pode e deve ou não ser feito em OiArduBot. Analisando novamente a figura 9, podemos notar que ela (classe) é responsável por agregar as estruturas de controle e iteração da linguagem, bem como as atividades que o robô pode realizar por meio desta. A **Funcao** é o bloco que implementa as funções que podem ser criadas dentro da linguagem, como em outras linguagens de programação procedurais. É comum que linguagens de programação permitam a implementação de funções (aqui, não nos detemos a distinção de função e procedimento). Dentre suas vantagens, elas possibilitam melhor organização do código e implementação de princípios como o DRY (Don't repeat yourself - Não repita você mesmo), que consiste na ideia de reaproveitamento de código, utilizando-se assim da criação de funções genéricas para realização de determinadas tarefas. A classe **Variavel** é fundamental na estrutura da linguagem. Tendo em vista as demais linguagens relatadas no presente trabalho, bem como a relevante pesquisa elaborada em cima das mesmas, concluímos que a existência de um bloco dedicado especificamente a declaração e assimilação de valores a variáveis seria uma abordagem conveniente.

Figura 10 – Primeiro nível do modelo da linguagem *OiArduBot*.



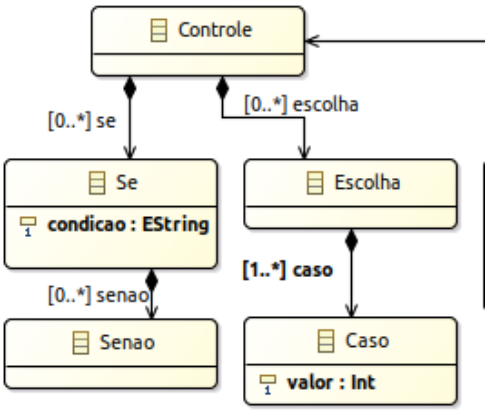
Fonte: Autor

Se atentarmos aos detalhes do modelo, iremos notar que existem alguns parâmetros descritos logo abaixo do título (nome) do bloco. Esses valores são as inserções de código que a classe está apta a receber. No caso da Variavel, são permitidos dois parâmetros: *name* e *valor*. Esses valores representam o nome da variável e seu tipo, respectivamente, de forma qual podemos subentender que a declaração de uma variável é subordinada a existência dos mesmos. Assim, podemos notar por intermédio desse exemplo, como o modelo interfere e define a estrutura gramatical da linguagem.

3.2.2.2 Modelagem das Estruturas de Controle

A Figura 11 ilustra a modelagem das estruturas de controle na linguagem OiArduBot.

Figura 11 – Modelo das estruturas de controle na linguagem *OiArduBot*.



Fonte: Autor

Hierarquicamente, as estruturas de controle estão abaixo do Codigo, logo, para o modelo, estão introduzidas dentro dessa classe. As estruturas de controle visam direcionar o fluxo de execução, ditando qual parte do código deve ou não ser executada. Denotamos dois grupos principais dentro desse tema: comandos de seleção e comandos de iteração. Os comandos de seleção remetem a estruturas condicionais, como se, senão, escolha-caso, que abordaremos nesse tópico. Os comandos de iteração serão relatados no item subsequente.

As estruturas de controle são implementadas mediante uma série de condicionais, que por sua vez são implementadas na OiArduBot sob os termos **se-senao**, e **escolha-caso**. Essas representações tem como objetivo a própria semântica. Por exemplo, se ocorrer determinada situação realize a primeira ação, senão realize a segunda. As decisões são tomadas a partir das condições impostas, as quais podem ser expressões relacionais ou lógicas[11].

Se, Senao: Consiste em dois blocos que, mediante a validação ou não da condição, direcionam o fluxo de execução.

Escolha, Caso: De forma semelhante a anterior, essa estrutura implementa um código específico com base no resultado da condicional. Para implementação é escolhido um parâmetro - **escolha**, que é comparado com valores pré-definidos pelo desenvolvedor - **caso**. O fluxo de execução é direcionado pelo bloco em que a condição coincidir.

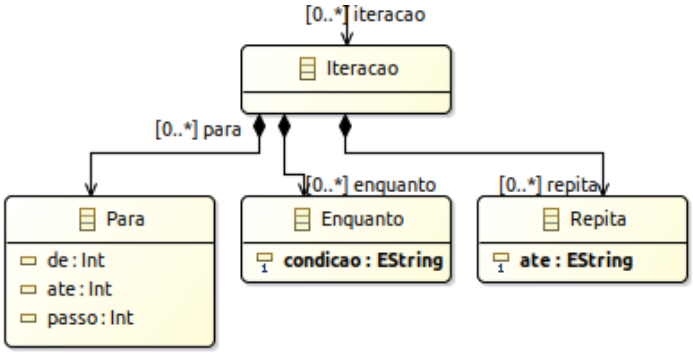
3.2.2.3 Modelagem das Estruturas de Iteração

A Figura 12 ilustra a modelagem das classes do controle de iteração da linguagem. Como padrão abordado no desenvolvimento do projeto, as classes que implementam as estruturas de iteração ou, como também chamadas, laços de repetição, são descendentes de uma classe

[11] <http://www.univasf.edu.br/~ricardo.aramos/disciplinas/AlgProgAgr_2011_1/cap03AnaEmilia.pdf>

principal. Essa hierarquia busca implementar uma centralidade e divisão no código, fazendo com que a classe superior seja composta pelas funcionalidades que lhe competem.

Figura 12 – Modelo das estruturas de iteração na linguagem *OiArduBot*.



Fonte: Autor

As estruturas de iteração remetem a repetição, ou, iteração de um bloco de código em *n* vezes. Na programação, apresentam grande impacto sobre o fluxo e os dados do programa. Os mais conhecidos são: **Para**, **Enquanto** e **Repita**. De acordo com o modelo apresentado na figura 12, a linguagem conta com a implementação de todos os laços citados.

Seguindo o padrão adotado, as estruturas de iteração são 'filhas' da classe **Iteracao**, que por sua vez faz a implementação das mesmas. Vejamos cada uma delas.

Para: Composto por duas palavras-chaves (de, ate) e três parâmetros. O Para implementa um contador implícito, que incrementa uma variável a cada repetição com o valor definido internamente pela OiArduBot (valor esse que é 1, assim, cada repetição a variável será incrementada em 1). Esse *loop* ocorre até que a variável de controle atinja o valor final.

Enquanto: Essa estrutura segue o mesmo princípio da anterior, a grande diferença, e é propriamente isso a disparidade entre as mesmas, é que o **Enquanto** não implementa um contador implicitamente, aqui, o desenvolvedor precisa programá-lo. É necessário uma expressão que defina até quando o laço irá repetir. Essa expressão consiste em uma relação lógica, geralmente entre uma variável e um valor ou outra variável, por exemplo: **numero>2**.

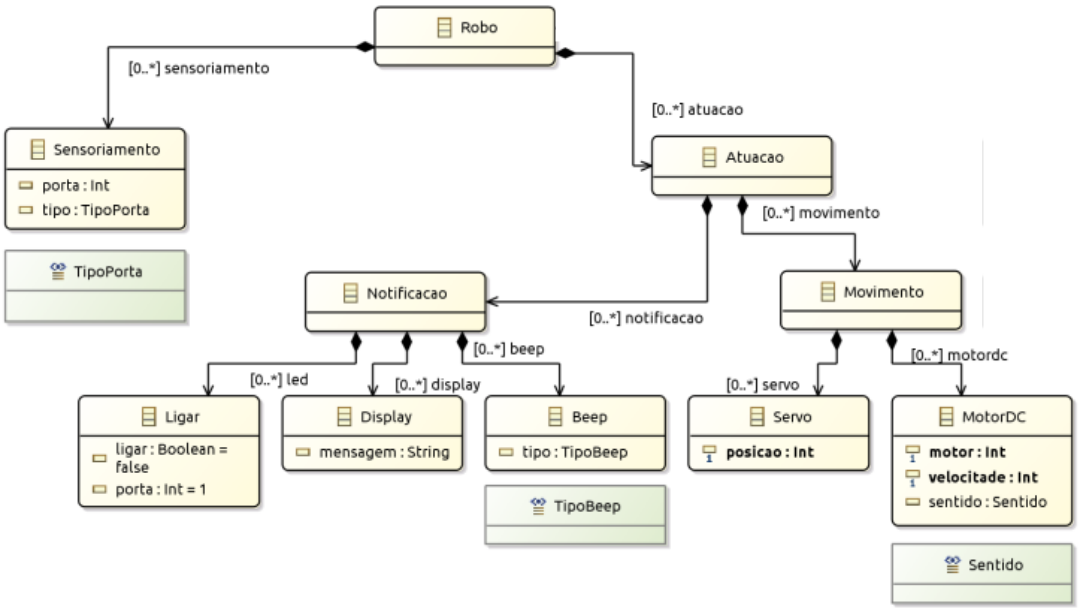
Repita: esse laço de repetição se assemelha muito ao enquanto, a única diferença é que ele se trata de um *loop* pós-testado, i. e, a expressão condicional que determina até quando o laço deve iterar fica no término do bloco, realizando a verificação após a primeira implementação do código descrito dentro do bloco.

3.2.2.4 Modelagem dos Comandos de Interação com o Robô

A Figura 13 modela a estrutura de funcionalidades de interação do robô. A classe principal denominada *Robo*, é composta por outras duas classes, que adentraremos contextualmente mais a frente.

O robô pode exercer diversas interações com o ambiente, seja pegando informações ou atuando sobre o mesmo, explicamos mais sobre isso na seção 3.1.1. Com base no modelo

Figura 13 – Modelo das interações com o robô na linguagem *OiArduBot*.



Fonte: Autor

das interações com o robô, podemos notar que existem diversas classes de implementação, as quais são divididas em dois blocos principais: **Sensoriamento** e **Atuacao**.

A classe **Sensoriamento** se volta para captação de dados ambiente. Por meio dela são implementadas funções de leitura de informações, com as quais podemos utilizar alguns componentes de sensoriamento do Arduino. De maneira didática, imagine um botão de pressão que, ao ser pressionado, acende um LED. Agora, vamos fazê-lo comunicar com nosso Arduino. Para tanto, precisamos de ao menos quatro conexões, duas para alimentação (GNC e VCC) e duas para comunicação. Nos atentaremos aos pinos de comunicação. A comunicação desse componente ocorre pela porta digital da placa, sendo necessário então a definição no código do tipo, e, qual porta ele está conectado. Por meio da classe de Sensorimento, podemos de forma sucinta chamar uma função denominada lerDigital() que é preparada para receber informações. A função **ler()** recebe apenas um valor por parâmetro. É necessário apenas informar o número da porta, já que a própria função indica que o sensor utiliza uma porta digital (o Arduíno conta com dois tipos de porta: digital e analógica. Esse simples código é capaz de ler as informações enviadas pelo botão para o Arduíno através de uma porta digital, como quando ele é pressionado, por exemplo.

A classe **Atuacao** remete as atividades que o robô pode exercer sobre o meio. Ela é responsável pelos movimentos e notificações que ele realiza. Por meio da implementação de outras classes, a linguagem permite ao usuário desenvolver uma quantidade limitada de atividades. Abordaremos a seguir as duas classes principais.

Notificacao: As funções de notificação são ações realizadas com intuito de passar informações ao meio externo. Isso geralmente ocorre de maneira visual, por meio de um visor ou LED (*Light*

Emitting Diode), ou sons (no Arduino comumente utilizamos o buzzer - campainha). Para tanto, são implementadas algumas funções que permitem ao usuário realizar essa interação. Em continuidade ao caso de uso do nosso botão de pressão e um LED, agora que já sabemos como implementar a leitura do estado botão, vamos descobrir como acender o LED. A ideia consiste em acender o LED quando o botão for pressionado. Assim, por meio da leitura do botão, podemos agora fazer uso de outra função denominada **ligar()**, essa por sua vez, pertencente à classe *Notificacao*. A função **ligar()** recebe apenas um parâmetro: o pino (ou porta) em que o componente está conectado. Portanto, precisamos somente informar em qual pino do Arduíno o LED está conectado na chamada da função e pronto, temos a luz acesa.

Movimento: Implementa as funções que permitem ao robô realizar movimentos. Por exemplo, se quisermos que nosso robô ande para frente determinada distância, fazemos uso das funcionalidades dessa classe.

3.2.3 Gramática da Linguagem

Como exemplo, a Lista 3.1 descreve de forma abreviada a sintaxe da linguagem OiArduBot para a estrutura de controle **Se**, conforme o modelo.

Lista 3.1 – Gramática da estrutura Se.

```
Model:
        ...
        codigo+=Codigo*;
Codigo:
        ...
        (controles+=Controle)?;
Controle:
        ses+=Se?
        ...;
Se:
        'se' valorEsquerda=VariavelValor relacao=Logico
        valorDireita=VariavelValor 'entao'
                (code+=Codigo)*
        (senao=Senao)?
        (fimse=Fimse)
;

Fimse:
        name='fimse'
;
```

O desenvolvimento de um metamodelo é o ponto de partida para o desenvolvimento da nossa linguagem. O mesmo, retratado pela figura 9, se equipara ao escopo da nossa linguagem apresentado na lista 3.1, uma vez que ele serve de base para o desenvolvimento. Na prática, o metamodelo precisa de uma representação gramatical. Esse é o ponto de encontro onde o Xtext e o metamodelo se 'conectam', no qual os modelos definem as regras - uma vez que são eles quem designam os limites através da modelagem, e o *framework* possibilita a implementação. Sendo assim, o modelo é implementado no desenvolvimento da linguagem.

A lista 3.1 apresenta parte da gramática implementada. Nela, podemos notar a presença de vários termos anteriormente abordados. Isso se dá pela inferência das classes do metamodelo dentro do código, a partir do processo de transformação. Assim, a estrutura modelada agora se amplia dentro de um código, o qual possibilita a realização de alterações e ajustes que o modelo não foi capaz de concluir.

Na primeira linha da lista podemos notar a presença da palavra *Model*. Essa classe é definida no modelo como início da regra, i. e, fica no nível mais alto da árvore hierárquica. É dentro do escopo do *Model* que são definidas as estruturas que a linguagem pode conter. Ele implementa a classe *Codigo*, que por sua vez é composta pela classe *Controle*, e assim sucessivamente, formando uma árvore de implementações. Vamos entender melhor essas configurações.

Primeiramente, precisamos ter em mente que o Xtext faz uso de cardinalidades na implementação das classes. A cardinalidade é o número de elementos do conjunto, que, de certa forma, define quantas vezes esse bloco pode ser chamado no desenvolvimento do código. Por exemplo, é necessário que o usuário possa fazer zero ou várias implementações da estrutura de controle **Se**, assim, sua cardinalidade deveria ser definida entre nenhuma e várias. Essas especificações são dadas por meio de símbolos que aparecem ao final da linha. São eles:

- **?** - representa zero ou uma implementação;
- ***** - representa zero ou mais implementações;
- **+** - uma ou mais implementações, e
- **Sem operador** - que indica exatamente uma implementação.

De posse desse conhecimento, vamos entender de maneira breve o código contido na lista 3.1. Como dito anteriormente, o *Model* faz a implementação do *Codigo*, o qual é explicitado na linha três. O *Codigo* pode não existir, como pode existir inúmeras vezes, como podemos concluir com a presença do * no final da linha. Isso ocorre porque ele é composto pela classe *Controle*. O *Controle* é quem permite o uso das estruturas de controle da linguagem, como o *Se*. Nesse ponto começa a ficar claro a equivalência que o modelo tem com a gramática.

O *Controle* serve como uma base que implementa as estruturas **Se** e **Escolha**, definidas pela gramática. Assim, as mesmas são chamadas, de certa forma, dentro do modelo Controle. Para todos os efeitos, as implementações feitas dentro de um bloco possuem cardinalidade.

É curioso o fato de algumas regras terem zero ou várias definições enquanto algumas são de aplicação única. Isso se dá porque a OiArduBot conta com algumas especificações

gramaticais, tendo como objetivo único facilitar a divisão do código, deixando-o mais limpo e organizado.

A lista 3.1 nos mostra ainda, como é construída a gramática da classe **Se**. Essa classe, como o nome sugere, faz a construção da estrutura de controle **Se**. A gramática é sucinta e objetiva, sendo composta por poucos componentes. A primeira linha do bloco (linha 11) está especificando que a estrutura deve se iniciar com a palavra 'Se', seguida de uma condição. A condição, requer que alguns parâmetros sejam informados, os quais comumente são: uma variável, uma expressão lógica e um valor ou outra variável. Por exemplo, **numero>2** é uma condição válida para essa estrutura. Abaixo, temos a definição do código que pode ser implementado dentro do bloco, ou seja, o trecho de código que deve ser executado caso a condição seja válida. O código é aberto, logo, o usuário pode se utilizar das funções e estruturas que a linguagem dispõe. De forma facultativa, a presença do **Senao** é definida, como podemos notar com a especificação do símbolo ? na linha 13. Essa estrutura, como o *Se*, permite que um bloco de código seja executado. Entretanto, o **Senao** só é chamada caso a condição não seja verdadeira. Assim como o Se, o Senao também é uma classe dentro do nosso modelo.

Abaixo é mostrado um exemplo de pseudo-código escrito em OiArduBot, conforme a gramática descrita acima.

```
se valor > outroValor entao
    //Execute isso
senao
    //Execute isso
fimse
```

O exemplo acima apresentado, que ilustra a prática da nossa sintaxe, implementa a estrutura de controle **Se**. Os termos são sugestivos, sendo intuitivo o seu significado. Num primeiro momento, fazemos a chamada da estrutura com a palavra chave *se*. Em sequência criamos uma condição; a condição pode ser entendida de maneira literal, servindo como uma espécie de teste para saber qual trecho de código executar. Caso a condição se concretize, i. e, seja verdadeira, então execute o código logo abaixo. Caso a condição seja falsa, ou melhor dizendo, não atenda a condição imposta, o código executado é aquele que vem em seguida da palavra **senao**. O termo **fimse** vem para expressar o término da nossa estrutura de controle.

4 RESULTADOS

Todos os artefatos apresentados nesse TCC estão disponíveis sob licença de código aberto (MIT[1]) no repositório GitHub <https://github.com/IntelAgir-Research-Group/OiArduBot>. Portanto, todos os procedimentos abaixo podem ser reproduzidos/replicados de maneira a melhor entender o conceito da linguagem. Assim, o presente trabalho menciona os seguintes resultados:

- Um modelo de linguagem de alto nível e em Português para o uso de robôs em disciplinas introdutórias de programação;
- Um pacote de replicação da linguagem, disponível sob licença open-source;
- Um editor web para a linguagem;
- Um tradutor da linguagem OiArduBot para a linguagem tradicional do Arduíno;
- Documentação para que outros pesquisadores/desenvolvedores possam melhorar o projeto.

4.0.1 A Linguage OiArduBot

A OiArduBot dispõe de um grupo limitado de funcionalidades. Contudo, são mais que suficientes para que possa-se iniciar o desenvolvimento de um projeto simples com Arduíno.

Tomemos como caso de uso um projeto que consiste no acionamento de um *buzzer* por meio de um botão de pressão.

Lista 4.1 – Caso de uso implementado em OiArduBot.

```
Variaveis
Numero campainha <- 3;
Numero botao <- 2;
Numero valorBotao <- 0;
Inicio
valorBotao <- lerDigital(botao);
se valorBotao = 1 entao
    ligar(DIGITAL, campainha);
senao
    desligar(DIGITAL, campainha);
fimse
esperar(10000);
Fim
```

A primeira linha do código em OiArduBot na figura, apresenta um trecho de código já estipulado que decorremos na subseção anterior, o bloco de variáveis. Aqui, podemos ver na prática como esse contexto é executado: um nome seguido da atribuição de um valor. O nome

[1] <https://opensource.org/licenses/MIT>

representa a nomeclatura literal da variável, como deverá ser mencionado no código. O valor após o sinal de atribuição da linguagem (<-), representa o conteúdo da mesma. Nesse caso, temos que a variável *campainha* possui o valor 3, a variável *botao* possui o valor 2 e a variável *valorBotao* contém o valor 0. Seguindo, temos a palavra chave *Inicio* que indica o começo do bloco do nosso pseudo-código. Todas as linhas que estiverem entre os limitantes *Inicio* e *Fim*, compreendemos como o núcleo do nosso código. Nos prenderemos a essa parte.

Vejamos linha a linha. A linha demonstra que a variável *valorBotao* recebe o retorno de uma função. A função em questão é a **lerDigital()**, que executa a leitura de informações em uma determinada porta. Contudo, essa funcionalidade necessita a passagem de um parâmetro (porta), a qual é realizada por meio da variável *botao* entre os parênteses que sucedem a mesma. Com isso, temos que a variável *valorBotao* contém o valor que é constantemente informado na porta 2. A linha seguinte implementa a estrutura de controle **se**. A mesma está sujeita à condição de *botao* ser igual à 1. Retomemos o funcionamento dessa estrutura. Primeiro, temos a chamada da condição evidenciada pela palavra chave **se**. Em seguida, entre os parênteses, temos a condição dessa estrutura. A condição é dividida em três partes: valor, expressão e valor. Entende-se que o valor pode ser muito bem representado por uma variável, visto que a mesma possui um valor. Sendo assim, se o resultado da condição for verdadeiro, o código que está na linha conseguFinte é executado. Essa situação busca capturar o momento em que o botão é pressionado. Caso ele seja pressionado, o valor da variável *valorBotao* será 1, do contrário será 0. Contando que o botão tenha sido pressionado, a linha 8 será executada, fazendo a chamada da função **ligar()**. Essa função, por sua vez, ativa (liga), a *campainha*, esperando somente que lhe seja passada o tipo e o número da porta em que o componente está conectado. Ao soltar o botão, o valor da variável *valorBotao* se tornará 0, e, consequentemente, o fluxo de programação será direcionado para o bloco **senao** da estrutura. Esse trecho segue o mesmo princípio, assim, invoca a função **desligar()** que desativa a *campainha*. Por fim, na linha 11 temos de maneira explícita o fim da estrutura, evidenciado pela palavra *fimse*.

Adiante, na linha 12, ocorre a chamada de uma função cuja assinatura é **esperar()**. Essa função tem como objetivo parar a execução do código por um tempo determinado, representado sempre em milissegundos. Nesse caso em específico, a execução do código será interrompida em 10 segundos. Na última linha temos o término do nosso código, o qual é declarado pela palavra chave *Fim*.

Como descrito anteriormente, um número suficiente de funções foram desenvolvidas. As funcionalidades foram pensadas para de certa maneira diminuir a dificuldade que o usuário teria ao ter que realizar alguma tarefa com o Arduíno. Assim, desenvolveu-se cada função para ser o mais sucinta possível, dentro do quadro de entendimento que obtivemos para realização do TCC. Dentre as funcionalidades, destacamos as funcionalidades para leitura de sensores e atuadores, sejam esses comunicados por portas digitais ou analógicas. Além das funções de leitura, dispõe-se de funções de escrita para os mesmos. O que fica evidente nos exemplos apresentados durante o trabalho.

Também presentes na linguagem, os laços de repetição e as condicionais são funcionalidades básicas que, pode-se dizer, são esperadas em qualquer linguagem ou pseudolinguagem de programação. Para tanto, foram implementadas todas as condicionais já descritas, como **se, senao** e **escolha**. Quanto aos laços de repetição, o **para** e o **enquanto**, mais conhecidos, foram implementados juntamente com um terceiro iterador, o **repita**. Esse último laço de repetição é raramente visto em uso, já que seu uso pode ser contornado por uma boa lógica de programação. Ele se equipara ao *do-while* usado na linguagem do Arduíno. A grande diferença entre o **enquanto** e o **repita**, é a verificação da condição antes ou depois de executar o código dentro do laço, respectivamente.

Temos em vista especializar e direcionar a linguagem cada vez mais para a robótica educacional. Com isso, visamos o futuro desenvolvimento de funções específicas para robôs educacionais. A linguagem está em constante desenvolvimento, como qualquer outra, tendo já uma base firme e tomando forma. Assim, acreditamos que a continuidade da linguagem, aliada a disponibilidade de colaboração da comunidade, pode levar a OiArduBot por caminhos satisfatórios.

4.0.2 Editor da linguagem

O Xtext dispõe de um editor web, que pode ser implementado de múltiplas maneiras. Aqui, vamos tomar como auxílio o *plugin* **DslForge**[2]. Esse projeto é de código aberto sob a licença pública Eclipse (EPL) e fornece um ambiente de edição web para linguagens de domínio específico textuais. Com base no Xtext, o editor detém uma quantidade específica de funcionalidades, que são tradicionalmente encontradas nos editores mais populares entre a comunidade de desenvolvimento. Destacamos:

- Realce de sintaxe,
- Validação de sintaxe,
- Assistente de conteúdo,
- Validação semântica do lado do servidor,
- Propostas de modelo,
- Texto flutuando,
- Atalhos de teclado padrão,
- Suporte para desfazer / refazer,
- Dobramento de código

A aplicação dessa ferramenta só é possível graças à outras tecnologias providas no Eclipse IDE, como o RAP (*Remote Application Platform*). A utilização desse componente é que torna possível a comunicação e validação em tempo real da linguagem, por exemplo.

Entretanto, para que essa funcionalidade esteja disponível, é necessário que um arcabouço de configurações sejam realizadas. Um tutorial de como configurar o Dslforge no Eclipse pode ser acessado em: <https://github.com/plugbee/dslforge/>. Com o ambiente

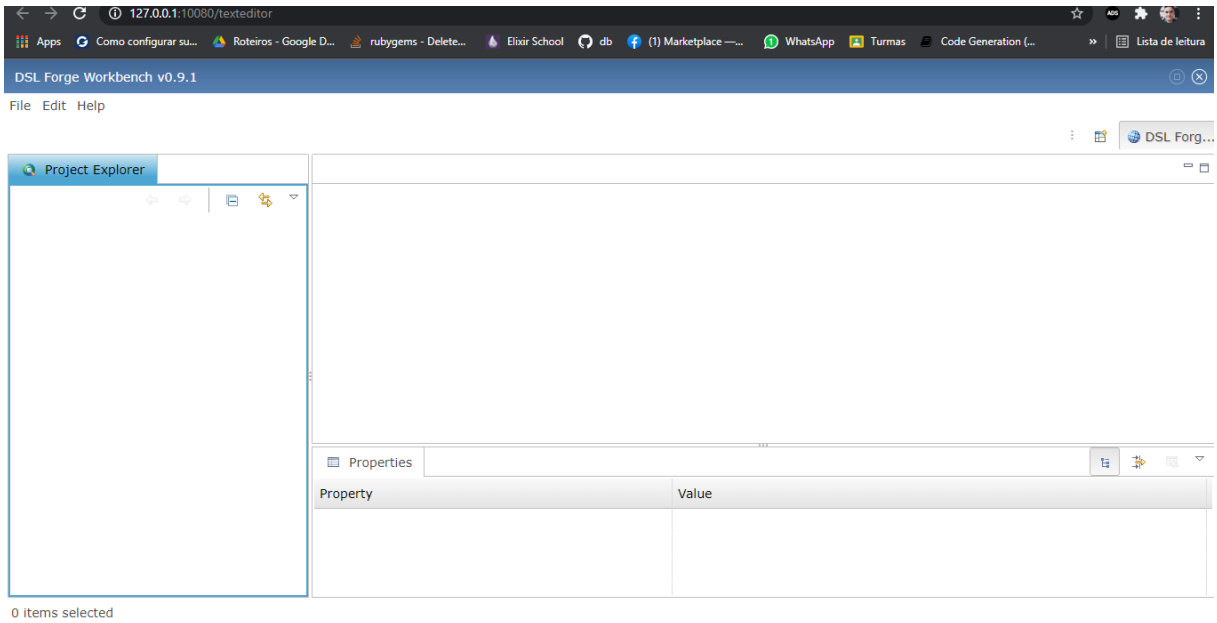
[2] <https://dslforge.org>

preparado, podemos começar a utilizar nosso editor. Ele se comporta como um editor de código comum, apresentando as funcionalidades mais conhecidas. Em uso, o consumo computacional na interação do cliente com o servidor é alto, visto que há uma grande troca de informações para garantir a validação, complementação, realce de sintaxe e assistência em *real time*.

Tendo a gramática do Xtext como entrada, o editor extrai uma série de informações e regras para realizar a análise integrada, tendo a possibilidade de acionar o analisador somente no lado do cliente.

Assim, após executar o editor web a partir do Eclipse IDE, temos como resultado o ambiente demonstrado na figura abaixo.

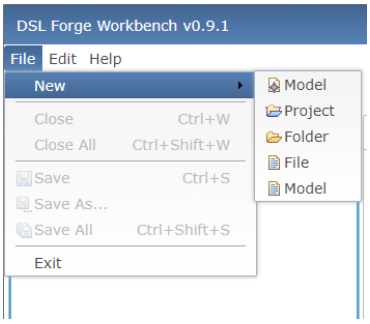
Figura 14 – Editor web para DSL usando Dslforge.



Fonte: Autor

Podemos notar que existem três opções no canto superior esquerdo, como demonstra a figura 15, que basicamente retornam todas as funcionalidades básicas necessárias para trabalhar com o editor.

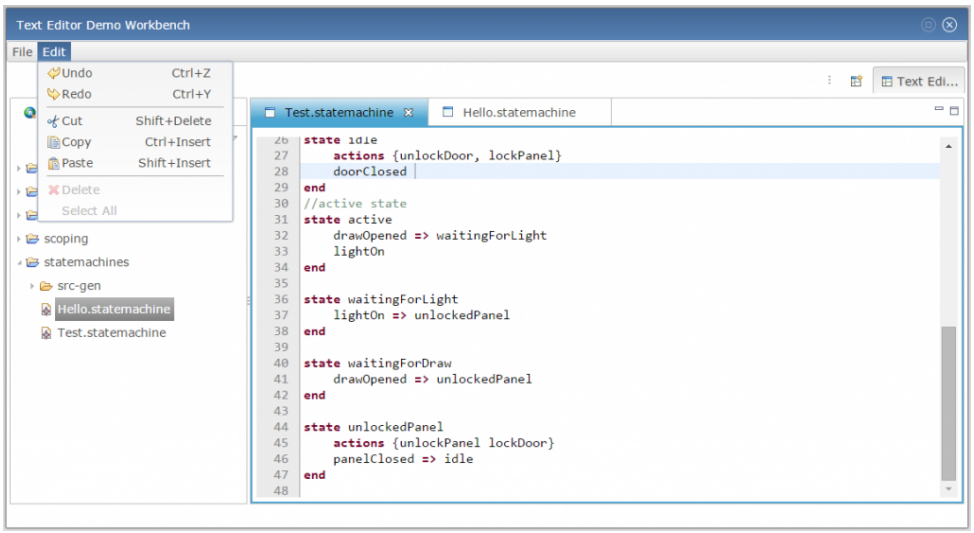
Figura 15 – Menu do editor web para DSL usando Dslforge.



Fonte: Autor

É possível criar um novo projeto, arquivo, modelo ou pasta por meio do editor. Após a criação do arquivo, iremos notar a presença dele na tela, já aberto para edição. Tomando como base o exemplo dado na apresentação de um trabalho pela equipe da Dslforge[3], ilustrado pela figura 16, podemos notar o destaque de código que o editor fornece.

Figura 16 – Editor de arquivo.



Fonte: https://dslforge.org/getting-started-generate-xtext-rap-editor/

Concluindo, temos um editor online que permite trabalharmos com a linguagem de maneira interativa, possibilitando funcionalidades interessantes antes improváveis para uma linguagem de domínio específico.

4.1 Compilador da Linguagem

O compilador é uma parte muito importante no trabalho. Ele é responsável por fazer a tradução do código em OiArduBot para a linguagem convencional do Arduino. Para tanto, existem uma série de passos, no entanto simples, a serem seguidos para realizar essa tarefa.

Como mencionado anteriormente, parte das ferramentas utilizadas utilizam como base o Java. Assim, para realizar a tradução do nosso código teremos que exportar um arquivo de extensão *.jar*. Não é complexo. Essa extensão de arquivo é considerada pela comunidade de desenvolvimento como um executável da linguagem Java, reunindo classes, arquivos e a informação de por onde o mesmo deve começar a ser executado. Assim, nosso primeiro passo é gerar um executável Java da nossa linguagem a partir do arquivo *Main.java*, que é criado quando executamos nossa linguagem.

O próximo passo, consiste na execução do arquivo que acabamos de gerar. Considerando nosso ambiente preparado, temos de estar na mesma pasta que o arquivo *.jar* gerado anteriormente. Para exemplificar melhor essa etapa, vamos considerar que nosso arquivo chama-

[3] <https://dslforge.org/getting-started-generate-xtext-rap-editor/>

se *OiArduBot-1.0.0-SNAPSHOT.jar*. Com o terminal aberto, no mesmo diretório do arquivo, executaremos o seguinte comando:

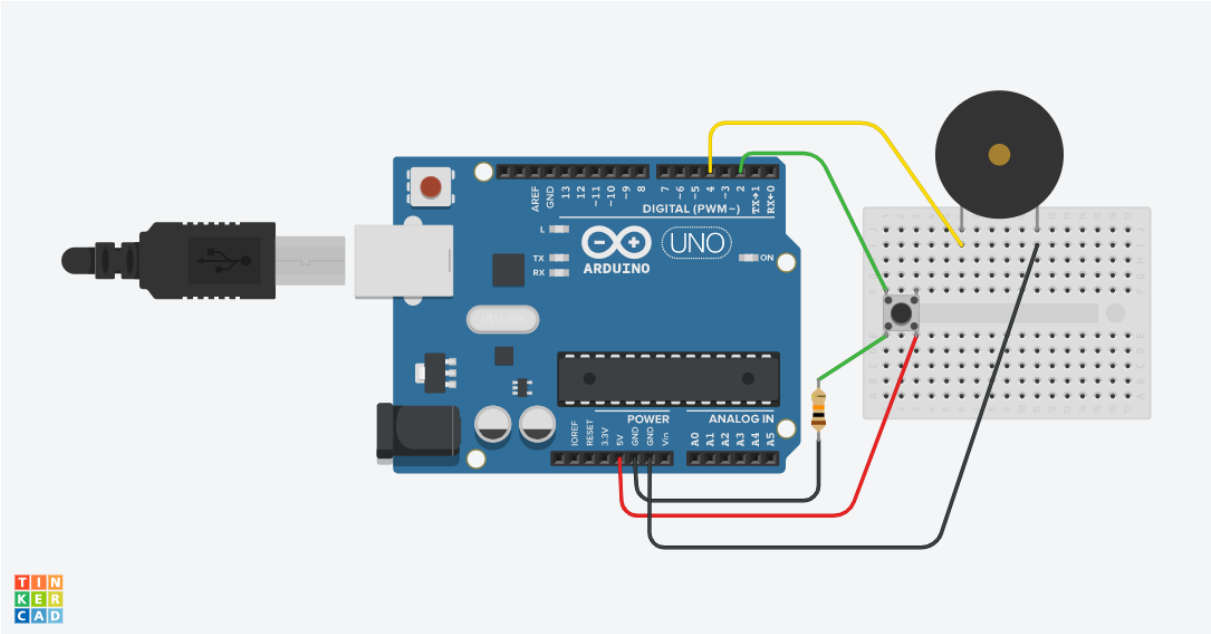
```
$ java -jar OiArduBot-1.0.0-SNAPSHOT.jar codigo.oab
```

Esse comando irá executar nosso arquivo *.jar* e terá como saída um arquivo com a extensão necessária para enviar para o Arduíno: *.ino*. Esse arquivo será salvo na pasta atual. O mesmo, é o arquivo que contém o código na linguagem convencional do Arduino, gerado com base no código escrito em OiArduBot. Vale salientar, que podemos customizar a pasta onde será salvo, e também o nome do arquivo que é gerado após a execução dos passos anteriores.

4.2 Testando o Código

De posse do nosso código na linguagem convencional do Arduíno, gerado a partir dos processos anteriores, podemos dar continuidade no desenvolvimento do projeto. Vamos abordar como poderia ser realizado o processo de compartilhamento do código com a plataforma Arduíno. Retomando nosso caso de uso apresentado na seção 4.0.1, agora iremos introduzir o envio do código para o Arduino. Para realização dessa tarefa será necessário criarmos uma conta na plataforma Tinkercad <https://www.tinkercad.com/>. Com o cadastro realizado, vamos no menu lateral (à esqueda), acessar a seção **Circuitos** e clicar em **Criar novo circuito**. A ferramenta é bem intuitiva, em poucos minutos podemos criar o circuito do nosso projeto, como é representado na figura abaixo.

Figura 17 – Protótipo na plataforma Tinkercad.



Fonte: Autor

Após o circuito pronto, vamos partir para a codificação do nosso projeto. O Tinkercad possibilita, além da criação do circuito, uma interação com ele, agindo como se fosse um Arduino

físico. Assim, podemos codificá-lo e observar seu comportamento. Para codificar o Arduino através da plataforma, vamos na opção **código** (localizada no canto superior direito da tela) e selecionar a opção texto no campo de seleção da tela que irá aparecer. Ao fazer isso, podemos notar que um editor de código se abre. Assim, copiaremos o código do nosso arquivo codigo.ino e colocá-lo no editor. Para executar é simples, basta clicarmos em **Iniciar Simulação**, ao lado do botão código. Pronto, temos nosso código rodando! Em apenas alguns passos podemos desenvolver um projeto simples com a nossa pseudolinguagem e vê-lo funcionando de maneira prática.

Essa sequência de passos nos dão a possibilidade de produzir inúmeros trabalhos em cima do Arduino, sem mesmo precisar ter um desses em mãos, muito menos ter dificuldades desnecessárias com a programação.

4.3 Editando a Sintaxe da Linguagem

Se alguém, em um trabalho futuro quiser melhorar a nossa linguagem, poderá baixar o projeto (disponível em nuvem), realizar alterações e enviar suas contribuições. Para tanto, é necessário que se tenha uma conta na plataforma Github. A plataforma é largamente utilizada pela comunidade de desenvolvedores para armazenamento e versionamento de projetos, códigos e afins. Uma vez registrado, o usuário deve acessar o endereço <https://github.com/IntelAgir-Research-Group/OiArduBot>. O endereço irá abrir uma página no Github. Nela, deve-se ir no botão *Código*. Irá aparecer diversas opções para baixar, ou, clonar o projeto (opção para quem tem um ambiente para github instalado no computador). O usuário deve escolher a opção que melhor se encaixa ao seu ambiente de desenvolvimento, tendo disponibilidade de *download* do projeto no formato .zip, também.

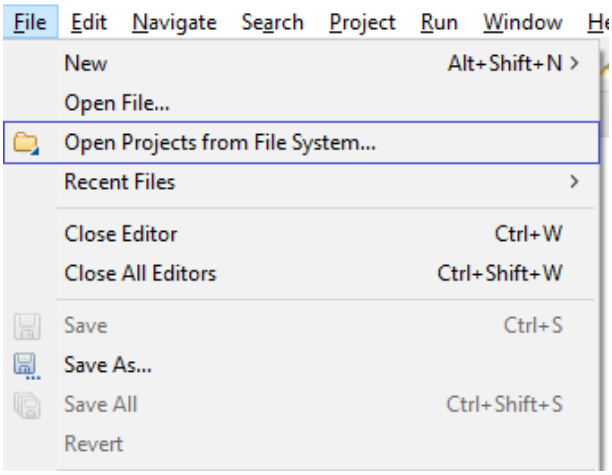
Após ter baixado o projeto, daremos início à inclusão. Vamos abrir o projeto na Eclipse IDE. Para importar o projeto é bem simples, estou usando a versão em inglês do software, que é a padrão. Basta ir em *File* no canto superior esquerdo, e em seguida, selecionar a opção *Open Projects from File System...*, conforme demonstra a figura 18.

Esse passo irá abrir uma outra tela, representada pela figura 19, onde ocorre o próximo passo. Agora, vamos selecionar o projeto do OiArduBot que acabamos de baixar. Assim, basta clicarmos em *Directory*, destacado pelo retângulo azul, procurarmos pela pasta e a selecionarmos.

Em alguns passos curtos, nosso ambiente já está preparado para trabalharmos no projeto. Destaco aqui, que estamos levando em consideração que o usuário já tenha a Eclipse IDE instalada e configurada para trabalhar com o desenvolvimento de DSLs, compreendendo os *plugins* necessários, como o Xtext. Caso ainda não tenha o Eclipse IDE instalado, pode acessar o site oficial em <https://www.eclipse.org/downloads/> e realizar o download. Um tutorial para instalação do Xtext (requisitado a instalação do Eclipse) pode ser encontrado em <https://www.eclipse.org/Xtext/download.html>.

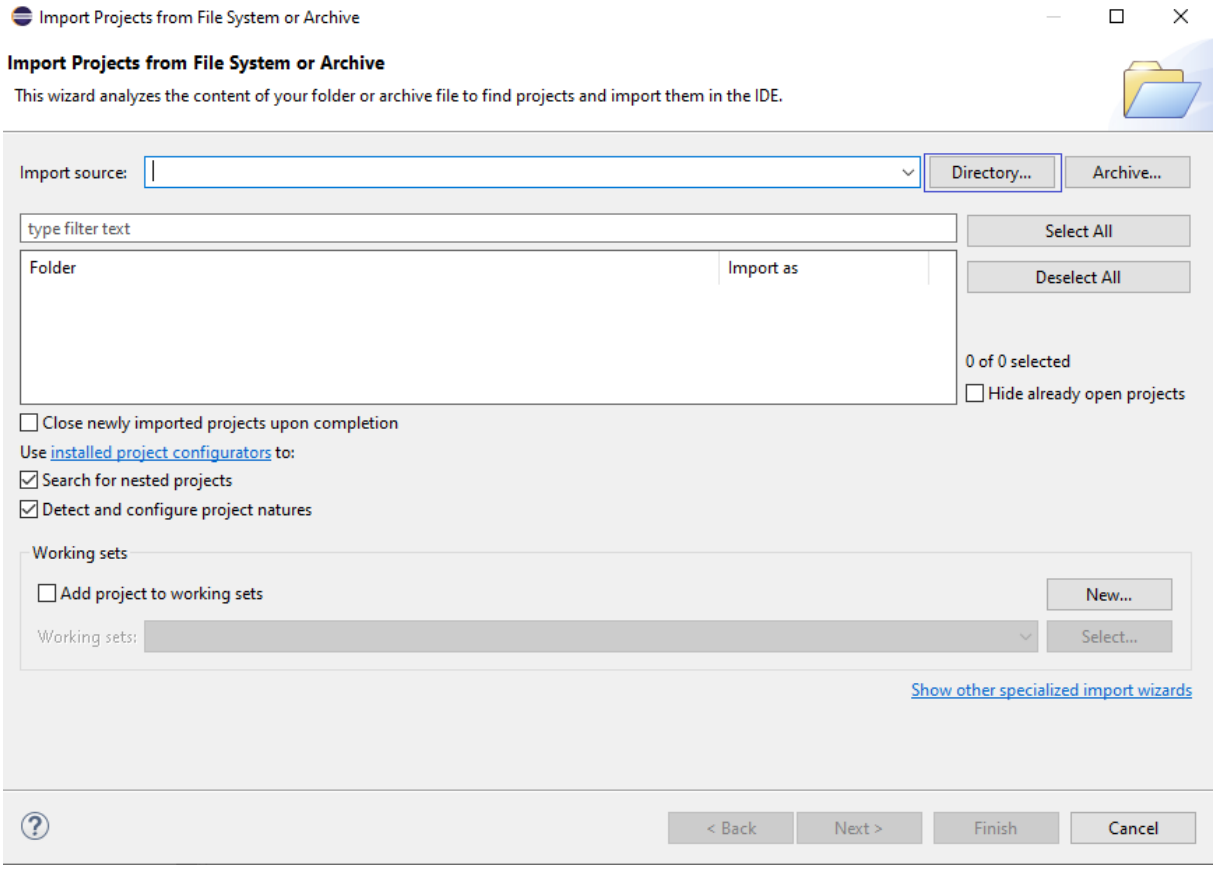
Vamos editar a nossa linguagem. O arquivo que contém a gramática, na qual defini-se

Figura 18 – Importar projeto para Eclipse IDE.



Fonte: Autor

Figura 19 – Selecionar pasta do projeto para abrir na Eclipse IDE.



Fonte: Autor

a linguagem, possui a extensão *.xtext*, e fica localizado no caminho *br.edu.utfpr.fb.OiArduBot* > *src* > *br.edu.utfpr.fb* > *OiArduBot.xtext*. Com um duplo clique no arquivo, abrimos o *OiArduBot.xtext*. Pode-se agora ver, na parte central da IDE, a gramática implementada em Xtext. Vamos editar algo simples, para entendermos o funcionamento desse *framework*.

Iremos implementar a função **piscarLed()**. Essa função irá piscar o LED que está conectado na porta informada, tendo como intervalo o valor passado para a função - esse valor será em *milissegundos*. Assim, temos em vista que nossa função irá receber dois parâmetros: porta e intervalo, ambos valores inteiros. Hora de programar. Com o arquivo *OiArduBot.xtext* aberto, vamos até o final do mesmo, aqui escreveremos a nova função. Com base na documentação do Xtext, implementaremos a função como está disposto na figura 20, logo abaixo.

Figura 20 – Função piscarLed() implementada em Xtext.

```
72 enum TipoBeep:
73     Simples | Duplo | Longo
74 ;
75
76 enum TipoPorta:
77     Digital | Analogica
78 ;
79
80 PiscarLed:
81     "piscarLed" '('  porta=INT ',' intervalo=INT ')'
82 ;
```

Fonte: Autor

Definimos nosso código da seguinte maneira: criamos uma nova regra denominada *PiscarLed*; em seguida, informamos que nossa regra deve conter o termo *piscarLed* no início, sucedido da abertura de um parênteses. Após, espera-se receber dois valores do tipo inteiro, separados por vírgula. Ao fim, temos o fechamento do parênteses. Note que os textos, em azul na figura, devem ficar dentro das aspas duplas ou simples.

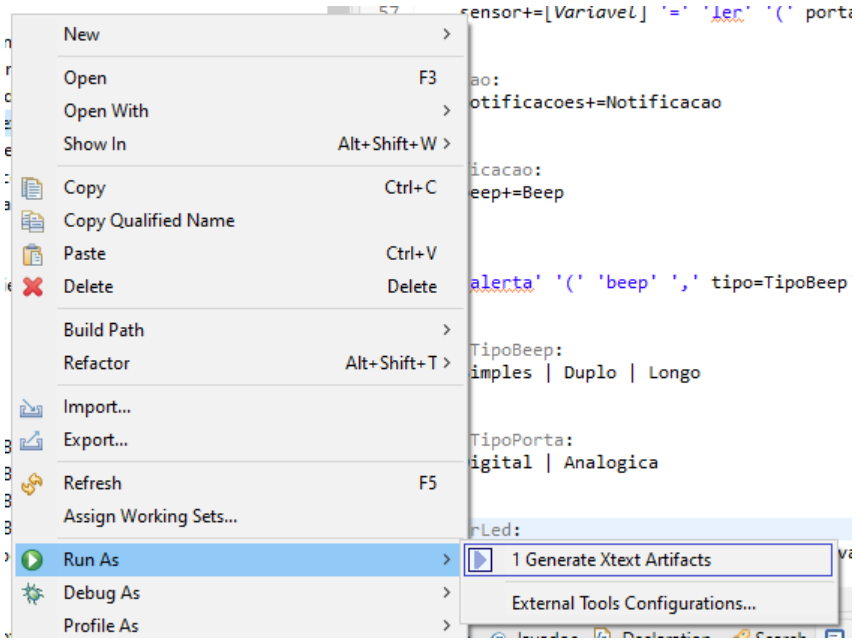
Após construirmos nossa função, a próxima etapa é gerar os artefatos. Os artefatos são parte dos arquivos essenciais para o funcionamento do programa, usados internamente. Assim que os artefatos são gerados, é como se um gerador de código em tempo real para nossa linguagem fosse implementado no projeto. Assim, podemos testar a sintaxe que acabamos de desenvolver. Para realizar essa tarefa, clicamos com o botão direito sobre o arquivo *OiArduBot.xtext*, vamos até *Run As* e selecionamos a opção *Generate Xtext Artifacts*, como ilustra a figura 21.

Em seguida, vamos sobre o pacote do projeto (*br.edu.utfpr.fb.OiArduBot*, clicamos com o botão direito, novamente em *Run As*, e deste vez vamos clicar na nova opção *1 Eclipse Application*, como representado pela figura 22.

Essa ação irá abrir uma nova instância do Eclipse. Aqui, criaremos um novo arquivo com a extensão da linguagem, que no nosso caso é *.hab*. É nesse arquivo que escreveremos nosso código em OiArduBot. Assim, como resultado do nosso trabalho, agora podemos implementar a função **piscarLed()**, como é demonstrado pela imagem a seguir.

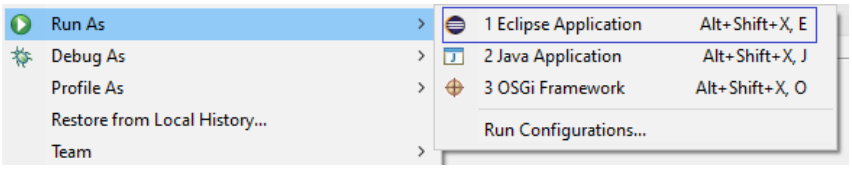
Para maiores detalhes, pode ser seguido o tutorial do site do Xtext <https://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html>.

Figura 21 – Gerar Artefatos do Xtext.



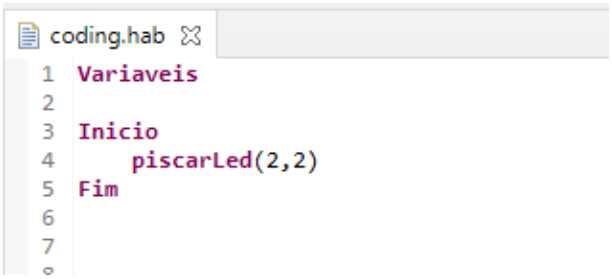
Fonte: Autor

Figura 22 – Gerar instância do Eclipse.



Fonte: Autor

Figura 23 – Implementando a função piscarLed().



Fonte: Autor

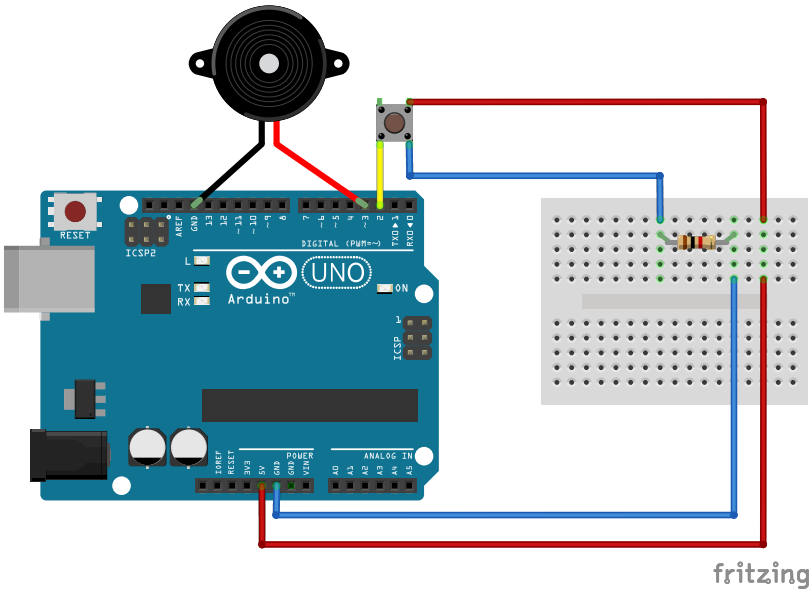
4.4 Comparativo com Outras Linguagens

A *OiArduBot* atualmente conta com um número limitado de funções, bem como os principais laços de repetição e condicionais. É necessário dizer que o escopo do projeto não irá atender a todas as funções existentes na linguagem de programação do Arduino, devido ao seu grande número de bibliotecas disponíveis. Entretanto, a linguagem implementa as principais

funções, quais foram julgadas relevantes para o trabalho e sua aplicação, mediante os trabalhos relacionados levantados e objetivos. Dentre essas, pode-se destacar desde funções mais básicas como ler o estado de um botão ou acender um LED (*Light emitter diode*), até funções que remetem a componentes e módulos encontrados com frequência em projetos com Arduíno, como sensor de temperatura, servo motor, entre outros. Assim, a linguagem irá se manter em níveis mais rasos de uma linguagem de programação de propósito geral - sintaxe e semântica, descartando, nesse primeiro momento, a possibilidade de desenvolvimento de funcionalidades mais avançadas como *bootloader*.

Para evidenciar a aplicabilidade da linguagem de programação proposta, bem como compará-la a outras linguagens, consideramos um projeto de robótica simples, com acionamento de som por um botão. A Figura 24 mostra a disposição dos componentes do projeto, que emite um som ligando o componente *buzzer* quando o botão é pressionado. Em seguida, mostramos como ficaria a programação para esse projeto com os três paradigmas: linguagem de blocos, linguagem C++ para Arduíno e a linguagem *OiArduBot*.

Figura 24 – Ilustração do caso de uso.



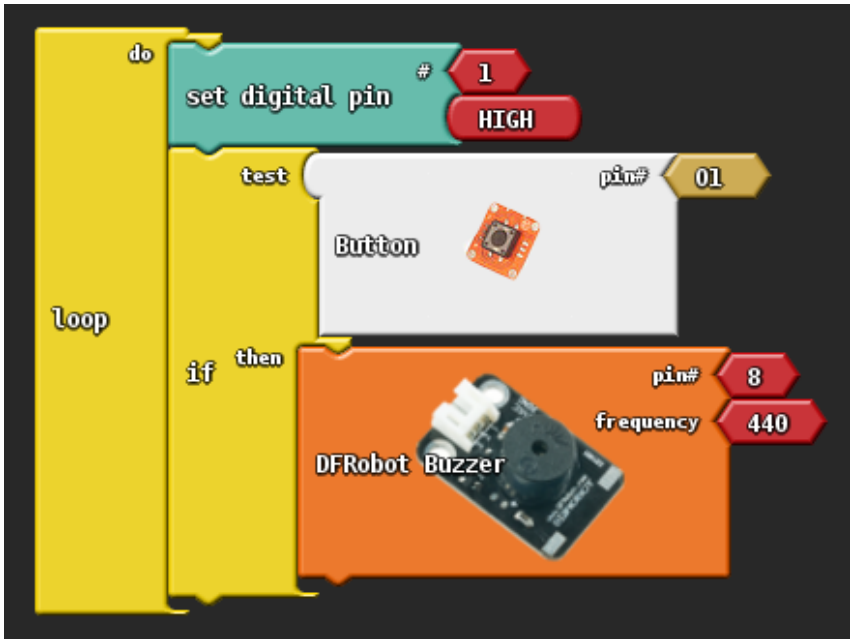
Fonte: Autor

4.5 Programação em Blocos

Para a programação em blocos, utilizamos a ferramenta Ardublock[4]. A Figura 25 ilustra a lógica, em blocos, para a implementação do estudo de caso, sendo necessários 10 blocos. Vemos que os blocos se encaixam, que é o princípio para construção do código nesse paradigma, o que auxilia em partes a programação.

[4] Website do Ardublock: <http://blog.ardublock.com/>

Figura 25 – Programação usando blocos.



Fonte: Autor

4.6 Programação na Linguagem C++ para Arduino

A Lista 4.2 mostra o código em C++ para programar o caso de uso, totalizando 17 linhas de código, escrita em língua inglesa e com dois métodos principais: *setup* e *loop*.

4.7 Programação na Linguagem OiArduBot

A Lista 4.3 mostra o código na linguagem de programação *OiArduBot*, com apenas 12 linhas de código, escrito em português e com uma estrutura de código semelhante aos algoritmos escritos em papel, utilizados por iniciantes em aulas de programação. Além de uma aparência mais sucinta, i.e., sem chaves, note que também são suprimidos alguns métodos necessários na programação em C++ para Arduíno (como setup e loop) (Lista 4.2).

```
const int pinBuz = 3;
const int pinSwi = 2;
int botao = 0;
void setup() {
  Serial.begin(9600);
  pinMode(pinBuz,OUTPUT);
  pinMode(pinSwi,INPUT);
}
void loop () {
  botao = digitalRead(pinSwi);
  if (botao == 1) {
    digitalWrite(pinBuz,1);
  } else {
    digitalWrite(pinBuz,0);
  }
  delay(10);
}
```

Lista 4.2 – Programação do caso de uso com C++/Arduino.

```
Variaveis
Numero campainha <- 3;
Numero botao <- 2;
Numero valorBotao <- 0;
Inicio
valorBotao <- lerDigital(botao);
se valorBotao  = 1 entao
    ligar(DIGITAL, campainha);
senao
    desligar(DIGITAL, campainha);
fimse
esperar(10);
Fim
```

Lista 4.3 – Programação do caso de uso com OiArduBot.

Comparando os códigos apresentados, pode-se notar que ocorre uma mudança abrupta de paradigma no processo atual. De fato há uma grande diferença entre o código em blocos e o código da linguagem convencional. Além da ruptura visual da abordagem, onde o aluno usava blocos gráficos para programar e agora precisa lidar com códigos textuais, a nova linguagem exige que o mesmo decore comandos em inglês e símbolos que são peça fundamental para funcionamento do programa. Levando em consideração a entrada da linguagem intermediária no processo de transição, é notória a simplificação do código (se comparado a linguagem convencional), concomitantemente à introdução de conceitos de programação da linguagem posterior.

4.7.0.1 Complexidade da Linguagem

Como medida de comparação e análise da complexidade entre as linguagens, elencamos a relação de palavras que foram empregadas no desenvolvimento do código do nosso caso de uso na linguagem convencional do Arduino e na nossa pseudolinguagem, OiArduBot. A inspeção trouxe como resultado as seguintes informações: **38 palavras** para o código convencional do Arduino e **29 palavras** para nossa linguagem. É importante salientar que contamos os parâmetros (valores) utilizados na atribuição à variáveis, funções e condicionais.

Com base nesses dados, podemos elencar alguns pontos que ficam evidentes. O número

de palavras empregadas para criar um código em OiArduBot é relativamente menor que na linguagem convencional, chegando à quase 25% essa disparidade. Outro ponto importante, que deve-se destacar, é a dificuldade trazida pelas palavras em outro idioma. Enquanto a linguagem convencional tem sua sintaxe em inglês, nossa linguagem é inteiramente em português brasileiro. Essa característica acaba por dificultar o desenvolvimento de projetos com o Arduíno, forçando o aluno a além de focar na aprendizagem da programação, lógica e sintaxe, ter de lidar com os comandos em uma língua diferente da sua materna.

O número menor de palavras na OiArduBot é relevante, pois a mesma ainda conta com algumas palavras que não existiam na linguagem convencional, se traduzida e comparada. Para ficar mais claro, um exemplo disso é o termo **Variaveis**. Há um total de 3 novas palavras encontradas no exemplo, sendo, além da citada, **Inicio** e **Fim**.

Uma comparação direta entre as linguagens pode mostrar ao leitor uma visão mais clara da evolução proposta. Assim, fica evidente que problemas o trabalho buscou resolver. Aqui, nos propomos a desenvolver uma pseudolinguagem que fosse intuitivamente mais simples e pode-se agir de igual modo.

5 CONCLUSÃO

Pensando em tornar a mudança de paradigmas de programação mais gradual, neste trabalho, propomos uma linguagem de programação intermediária, que chamamos de *OiArdu-Bot*.

A linguagem proposta, mantém as premissas de lógica de programação apresentadas na linguagem de blocos, agora apresentando-se de maneira escrita, como uma linguagem de programação convencional. Contudo, a escrita é mais próxima da linguagem natural e em português, buscando melhor entendimento por parte dos alunos. Mantendo princípios já trabalhados, seja na linguagem de blocos ou em outro paradigma, busca-se dar ao aluno uma ferramenta para aplicar seus conhecimentos e aprimorar sua lógica, sem ter uma textualização pesada - qual dependa de chaves e outros símbolos de maneira excessiva e, sem necessitar de conhecimentos adjacentes, como domínio de outra língua. Devido ao custo-benefício e questões de acesso, que possam impactar na difusão do projeto, nossa linguagem é voltada a robôs prototipados com Arduino.

Assim, com essa linguagem intermediária, pode-se tentar tornar o processo de ensino aprendizagem de programação mais gradual, buscando interligar os fundamentos de cada paradigma, ou seja, blocos e linguagem de programação, para que se tenha uma transição mais suave por meio de uma ferramenta que auxilie na construção do conhecimento.

5.1 Limitações

O trabalho, como qualquer outro, traz consigo algumas limitações. Com base em nossos estudos, investimos esforços em desenvolver uma linguagem voltada especificamente para Arduíno, o que acaba restringindo o seu uso apenas para essa plataforma. No entanto, com pequenas alterações no compilador, ela poderá ser utilizada para outras, mantendo seus princípios já destacados anteriormente, como sintaxe simples e em português.

A documentação da linguagem conta com algumas funções predefinidas e atualmente não dá suporte para desenvolvimento de funções ou procedimentos via código. Esse quesito pode em algum momento, restringir a criatividade do aluno e, dependendo da proporção, dificultar o desenvolvimento de algum projeto. Ainda, vale destacar, que a sintaxe para criação de funções já é válida, faltando apenas sua tradução pelo compilador. No entanto, reforçamos que a ideia é diminuir o escopo, para que a aprendizagem seja simplificada. Para elencarmos demais limitações nesse sentido, se faz necessário a aplicação da linguagem, por exemplo, para iniciantes em robótica. Assim, pode-se levantar os aspectos nos quais os alunos apresentaram dificuldades e demais quesitos que não são possíveis de mensurar antes de sua aplicação efetiva.

Por fim, a linguagem ainda não conta com um *bootloader* (*software* que permite a inicialização de um sistema). Sendo assim, é necessário que o aluno faça o envio do código para

o Arduino usando a IDE dele, ou até mesmo usando a feraramenta Tinkercad [1], apresentada no trabalho. Talvez, em trabalhos futuros, que podem gerar outros TCCs, isso venha a ser criado.

5.2 Trabalhos Futuros

O ponto principal que denotamos como trabalho futuro é o desenvolvimento de um editor executável para a linguagem. Com funcionalidades semelhantes as dispostas pelo editor web hoje disponível para DSLs textuais desenvolvidas com Xtext. O editor irá trazer novos horizontes para pseudolinguagem, tornando-a cada vez mais independente. Isso facilitará a difusão da nossa ferramenta, disponibilizando aos interessados um ambiente completo para desenvolvimento em *OiArduBot*.

Pretendemos ampliar a sintaxe da linguagem, adicionando mais funcionalidades. Essas funcionalidades visarão a utilização do Arduíno, porém mantendo nosso foco na robótica educacional, logo, focaremos em casos de uso para aplicação do mesmo como ferramenta de auxílio para ensino de programação e robótica. Concomitantemente, visamos a expansão da documentação da linguagem, proporcionando ao usuário uma base de consulta mais vasta.

Por estar atualmente aberto para possíveis engajamentos e melhorias, o projeto trás perspectivas de continuidade e crescimento. Sendo disponibilizado através da plataforma github [2], acreditamos que o trabalho possa originar outros trabalhos, sejam TCCs, artigos, ou até mesmo contribuições que venham agregar no desenvolvimento da OiArduBot.

5.3 Considerações Finais

O trabalho ocorreu em etapas que tinham como finalidade objetivos bem específicos, que facilitaram o desenvolvimento gradual do mesmo. Como parte inicial, foi realizada a modelagem da linguagem, a qual serviu e, também serve, como base para desenvolvimento contínuo da linguagem, sendo comum adotar esse processo no âmbito do desenvolvimento de software. Ainda, usamos como base para desenvolvimento do projeto a análise das tecnologias citadas, bem como o levantamento de referenciais teóricos. A *OiArduBot* apresenta grandes avanços, tendo disponível funções e condicionais, além de contar com um editor no qual podemos testar a linguagem.

Atualmente, tendo uma base sólida, a linguagem apresenta indícios (funcionamento adequado e atendendo as necessidades dos casos de uso) de ser uma abordagem eficiente para que a aplicação da robótica na educação seja facilitada.

Essa área, pouco desbravada ainda, é um campo de grande valia para que se expanda o aprendizado de programação, tornando-se porta de entrada para a robótica em escolas e universidades. Por fim, presumimos que o nosso projeto possa ser utilizado como base para

[1] Website do Tinkercad: <https://www.tinkercad.com>
[2] Website do github: <https://github.com/>

outros trabalhos de conclusão de curso, contribuindo ainda mais para o uso da *OiArduBot* na robótica educacional.

Referências

AGUIAR, D. S. de. **Robótica Educacional com Arduino como Ferramenta Didática para o Ensino de Física**. Sobral, CE, 2017. Citado na página 20.

ALI, S. et al. Constructionism, ethics, and creativity: Developing primary and middle school artificial intelligence education. In: **International Workshop on Education in Artificial Intelligence K-12 (EDUAI'19)**. [S.l.: s.n.], 2019. Citado na página 13.

AMSTERS, R.; SLAETS, P. Turtlebot 3 as a robotics education platform. In: SPRINGER. **International Conference on Robotics and Education RiE 2017**. [S.l.], 2019. p. 170–181. Citado na página 13.

ARDUINOUNO. 2018. Disponível em: <https://commons.wikimedia.org/wiki/File:ArduinoUno.svg>. Citado na página 23.

ARNOLD, G. V.; HENRIQUES, P. R.; FONSECA, J. C. Uma proposta de linguagem de programação para robótica. 2002. Citado na página 10.

ATAÍDE, J. F.; MESQUITA, N. A. d. S. O arborescer das tic na educação: da raiz aos ramos mais recentes. Brasil, 2014. Citado na página 9.

BENITTI, F. B. V. Exploring the educational potential of robotics in schools: A systematic review. **Computers & Education**, Elsevier, v. 58, n. 3, p. 978–988, 2012. Citado 2 vezes nas páginas 14 e 15.

BORELLI, H.; CARVALHO, S. T. d. **Uma linguagem de modelagem de domínio específico para linhas de produto de software dinâmicas**. Dissertação (Mestrado) — Universidade Federal de Goiás, 2016. Citado na página 22.

BOUDOURIDES, M. A. Constructivism and education: A shopper's guide. In: **International Conference on the Teaching of Mathematics, Samos, Greece**. [S.l.: s.n.], 1998. v. 3, n. 6. Citado na página 13.

BUARQUE, A. Desenvolvimento de software dirigido a modelos. 2009. Citado na página 24.

DANIELA, L.; LYTRAS, M. **Learning Strategies and Constructionism in Modern Education Settings**. [S.l.]: IGI Global, 2018. Citado na página 13.

DARGAINS, A.; SAMPAIO, F. F. Estudo exploratório sobre o uso da robótica educacional no ensino de programação introdutória. **Master's thesis, Universidade Federal do Rio de Janeiro-UFRJ**, 2015. Citado na página 11.

DEURSEN, A.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. **SIGPLAN Notices**, v. 35, p. 26–36, 01 2000. Citado na página 17.

Eclipse Foundation. **A linguagem da gramática**. 2020. Disponível em: <https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html>. Acesso em: 20 fevereiro 2020. Citado na página 22.

EGUCHI, A. Robotics as a learning tool for educational transformation. In: **Proceeding of 4th international workshop teaching robotics, teaching with robotics & 5th international conference robotics in education Padova (Italy)**. [S.l.: s.n.], 2014. Citado na página 14.

FEURZEIG, W. et al. Programming-languages as a conceptual framework for teaching mathematics. **ACM SIGCUE Outlook**, ACM New York, NY, USA, v. 4, n. 2, p. 13–17, 1970. Citado na página 13.

FREIRE, P. **Pedagogia da autonomia: saberes necessários à prática educativa**. [S.l.]: Paz e Terra, 2009. ISBN 978-85-7753-015-1. Citado na página 9.

GADELHA, F. et al. Práticas e experiências no ensino de engenharia dirigida por modelos. **iSys - Revista Brasileira de Sistemas de Informação**, v. 9, p. 106–135, 08 2016. Citado na página 24.

GOMES, A. J.; HENRIQUES, J.; MENDES, A. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. **EFT: Educação, Formação & Tecnologias**, Educom-Associação Portuguesa de Telemática Educativa, v. 1, n. 1, p. 93–103, 2008. Citado na página 11.

HAREL, I. E.; PAPERT, S. E. Constructionism. Ablex Publishing, 1991. Citado 2 vezes nas páginas 9 e 13.

HENSON, K. T. **Curriculum planning: Integrating multiculturalism, constructivism, and education reform**. [S.l.]: Waveland Press, 2015. Citado na página 13.

HONDA, L. C. R.; JUMES, F. Linguagem específica de domínio para programação de robôs. 2005. Citado na página 17.

JENKINS, T. On the difficulty of learning to program. In: **Loughborough University**. [S.l.: s.n.], 2002. Citado na página 11.

MANSO, A.; OLIVEIRA, L.; MARQUES, C. G. Ambiente de aprendizagem de algoritmos–portugol ide. In: **VI Conferência Internacional de TIC na Educação**. [S.l.: s.n.], 2009. p. 969–983. Citado 2 vezes nas páginas 10 e 16.

MCKERROW, P. J.; MCKERROW, P. **Introduction to robotics**. [S.l.]: Addison-Wesley Sydney, 1991. v. 3. Citado na página 19.

MENDES, A. **TIC – Muita gente está comentando, mas você sabe o que é?** 2008. Disponível em: <https://imasters.com.br/devsecops/tic-muita-gente-esta-comentando-mas-voce-sabe-o-que-e>. Citado na página 9.

MINSKY, M.; PAPERT, S. A. **Perceptrons: An introduction to computational geometry**. [S.l.]: MIT press, 2017. Citado na página 13.

MOURA, J. et al. Explorando o refinamento de uma dsl para versões baseadas em emf, eclipse sirius e xtext. In: **Anais da IV Escola Regional de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2020. p. 21–30. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/eres/article/view/13712>. Citado na página 25.

OVERVIEW. 2020. Disponível em: <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>. Citado na página 14.

PAPERT, S. **The children's machine: Rethinking school in the age of the computer.** [S.l.]: ERIC, 1993. Citado 2 vezes nas páginas 10 e 13.

PIAGET, J. **The construction of reality in the child**. [S.l.]: Routledge, 2013. v. 82. Citado 2 vezes nas páginas 10 e 13.

PINCIROLI, C.; BELTRAME, G. Buzz: An extensible programming language for heterogeneous swarm robotics. In: IEEE. **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2016. p. 3794–3800. Citado na página 10.

RIQUELMO, J. et al. Uma linguagem específica de domínio para a representação de modelos conceituais de bancos de dados relacionais. In: **Anais da III Escola Regional de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2019. p. 89–96. Disponível em: <https://sol.sbc.org.br/index.php/eres/article/view/8500>. Citado na página 17.

SEYMOUR. 1968. Disponível em: <https://proyectoidis.org/seymour-papert/>. Citado na página 14.

SILVA, A. H. A. et al. Usando a robótica educacional com scratch e arduino para melhor compreensão de ciências exatas. **Scientia Prima**, v. 6, n. 1, p. 147–159, 2020. Citado 3 vezes nas páginas 14, 15 e 20.

SPANGSBERG, T. H.; BRYNSKOV, M. The nature of computational thinking in computing education. **International Journal of Information and Education Technology**, v. 8, n. 10, p. 742–747, 2018. Citado na página 13.

THOMAZ, S. et al. Roboeduc: a pedagogical tool to support educational robotics. In: IEEE. **2009 39th IEEE Frontiers in Education Conference**. [S.l.], 2009. p. 1–6. Citado na página 10.

VALENTE, J. A. Informática na educação no brasil: Análise e contextualização histórica. 1999. Citado na página 9.

ZHONG, B.; XIA, L. A systematic review on exploring the potential of educational robotics in mathematics education. **International Journal of Science and Mathematics Education**, Springer, v. 18, n. 1, p. 79–101, 2020. Citado 2 vezes nas páginas 14 e 15.