UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniel Severo Estrázulas

Linguagem Específica de Domínio para Descrever Regras de Classificação de Candidatos ao Sistema de Cotas da Rede de Ensino Pública Federal

Recife
2020

Daniel Severo Estrázulas

Linguagem Específica de Domínio para Descrever Regras de Classificação de Candidatos ao Sistema de Cotas da Rede de Ensino Pública Federal

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

Área de Concentração: Engenharia da Computação

Orientador: Leopoldo Motta Teixeira

Recife
2020

Catalogação na fonte
Bibliotecária Mariana de Souza Alves CRB4-2105

E82l Estrázulas, Daniel Severo
Linguagem Específica de Domínio para Descrever Regras de Classificação de Candidatos ao Sistema de Cotas da Rede de Ensino Pública Federa/ Daniel Severo Estrázulas. – 2020.
139f.: il., fig., tab.

Orientador: Leopoldo Motta Teixeira.
Dissertação (Mestrado Profissional) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndices.

1. Engenharia da Computação. 2. Linguagem de Domínio Específico. 3. Meta Programming System. 4. Lei de Cotas 12.711/2012. I. Teixeira, Leopoldo Motta. (orientador) II. Título.

621.39 CDD (22. ed.) UFPE-CCEN 2020-179

Daniel Severo Estrázulas

Linguagem Específica de Domínio para Descrever Regras de Classificação de Candidatos ao Sistema de Cotas da Rede de Ensino Pública Federal

Dissertação apresentada ao Programa de Pós-Graduação Profissional em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 04 de setembro de 2020.

Aprovado em: 04/09/2020.

BANCA EXAMINADORA

Prof. Leopoldo Motta Teixeira
Centro de Informática / UFPE

Prof. Elder de Macedo Rodrigues
Universidade do Pampa

Prof. Márcio Lopes Cornélio
Centro de Informática / UFPE

AGRADECIMENTOS

Ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, em nome do Coordenador Professor **Ricardo Bastos Cavalcante Prudêncio**.

Ao Orientador Professor **Leopoldo Motta Teixeira**, pela paciência, por contribuir com os conhecimentos sobre o tema alvo da pesquisa, assim como as suas valiosas contribuições no desenvolvimento do trabalho.

Aos Professores **Elder de Macedo Rodrigues**, **Márcio Lopes Cornélio** e **Vanilson André de Arruda Buregio** pelas contribuições na Banca de Qualificação e de Defesa.

A minha mãe, **Julia Rosa Severo**, pelo apoio e pela motivação durante o meu processo estudantil e acadêmico.

A minha esposa, **Ana Paula Boff**, pela compreensão e pelo apoio durante as minhas ausências necessárias para o acompanhamento das aulas e para o desenvolvimento desse estudo.

Aos colegas **Eliandro Luiz Minski** e **Samuel Bristot Loli**, pela parceria durante as atividades do curso, assim como pela troca de conhecimentos e experiências de vida que vão contribuir de maneira imensurável para o meu futuro profissional e pessoal.

Ao **IFSC** pela oportunidade e liberação para as aulas, aos colegas da **Reitoria**, em especial os amigos da **DTIC**, pela troca de ideias, dicas, sugestões e pelo apoio nos compromissos e atividades assumidas em função da minha ausência durante as aulas.

A **todos os participantes** da presente pesquisa, em especial aos colegas do **Departamento de Ingresso (DEING)** da reitoria e dos campus, que contribuíram imensamente para os levantamentos e testes realizados por esse estudo.

A todos os **Professores**, **Técnicos Administrativos** e **Auxiliares** do **CIN**, em especial, a servidora **Joelma Souza de Menezes Franca** que sempre nos atendeu com presteza e prontidão em todos os assuntos e solicitações relativas ao curso.

A **todos os colegas de aula**, que contribuíram para os trabalhos em grupo, discussões, estudos e risadas durante as atividades realizadas dentro e fora do câmpus.

Aos membros da **comunidade MPS Jetbrains**, pelas dúvidas tiradas sobre os recursos do MPS por meio do canal oficial **slack (`jetbrains-mps.slack.com`)**, que foram de grande auxílio para o desenvolvimento da linguagem DSL Cotas.

RESUMO

Este trabalho apresenta uma pesquisa que tem como objetivo compreender, por meio da elaboração de uma linguagem específica de domínio, a viabilidade de melhoria na comunicação entre usuários de negócio e desenvolvedores, visando o aumento na produtividade da especificação de requisitos e da implementação de regras concernentes ao sistema de cotas da rede de ensino pública federal. A classificação de candidatos cotistas é garantida por meio da lei nº 12.711/2012, em conjunto com os decretos nº 7.824 e nº 9.034, os quais passaram por mudanças em suas diretrizes nos anos de 2012, 2016 e 2017. Adicionalmente a essas mudanças, podem ocorrer situações em que a interpretação de lei é alterada. Essas dificuldades podem resultar em atrasos na aderência à legislação, assim como falhas na comunicação entre desenvolvedores e especialistas de negócio. Portanto, a criação de uma linguagem que expresse regras de distribuição de vagas de maneira mais clara poderá auxiliar na geração de código de classificação com menor dependência de conhecimento técnico em programação, o que pode contribuir para a celeridade dos processos de ingresso em futuras alterações de regras de classificação. Como metodologia realizou-se uma pesquisa de natureza qualitativa com 20 usuários, de modo a verificar as dificuldades de compreensão e uso da linguagem desenvolvida na ferramenta Meta Programming System (MPS) da JetBrains, além do desenvolvimento de uma Application Programming Interface (API) como prova de conceito para a respectiva geração do serviço de classificação. Para tanto, esse trabalho foi baseado em um levantamento do histórico de mudanças no sistema de controle de versões do Instituto Federal de Santa Catarina (IFSC), no qual foram identificadas as principais alterações realizadas em função de versões de lei até o momento. Como resultado da presente pesquisa obteve-se o comparativo de dificuldades e preocupações sobre o uso da DSL entre 4 (quatro) grupos de usuários com diferentes características de formação e experiência. Após essa análise foram implementadas algumas melhorias na linguagem, com relação aos comentários e sugestões dos usuários. Ademais, com os testes da API, foi possível comparar os resultados da implementação da linguagem com os dados históricos de processamento de candidatos do IFSC em 16 processos seletivos. Por fim, conclui-se que a aplicação da DSL Cotas pode contribuir para a melhoria de comunicação e de entendimento entre os diferentes perfis de conhecimento dos envolvidos, além de propor um novo meio de aplicação prática-teoria de linguagens de domínio específicas. Ressalta-se a relevância social desse estudo, no que diz respeito às ações institucionais para atendimento de demandas que tratam de questões sobre a inclusão social como prioridade nos processos seletivos de ingresso.

Palavras-chaves: Linguagem de Domínio Específico. Meta Programming System. Lei de Cotas 12.711/2012. Regras para Classificação de Candidatos. Rede Federal de Ensino.

ABSTRACT

This work presents a research that aims to understand the feasibility of improving communication between business users and developers, through the elaboration of a domain specific language, intending to increase the productivity during the requirements specification and at the implementation of vacancy reservation law rules, required for federal public schools. The reservation categories are guaranteed by the law nº 12.711/2012 together with decrees nº 7.824 e nº 9.034, which were updated by changes in its guidelines in the years 2012, 2016 and 2017. In addition some changes may be the result of the law misinterpretation. These difficulties can result in delays in adhering to legislation, as well as failures in communication between developers and business experts. Therefore, the creation of a language that expresses the distribution rules in a clearer way may assist in the development of processing code with less dependence on technical knowledge in programming, which could contribute to speed up the process of learning and developing new changes on the reservation rules. This work addressed as a methodology, a qualitative research in which 20 users participated, in order to verify the difficulties using the DSL Cotas language, which was developed using the JetBrains Meta Programming System (MPS) tool. In addition, an API was also developed as proof of concept of the language model, with regard to generating valid source code for processing candidates lists. For this purpose, this study takes into account the history of changes in the IFSC version control system, in which the main changes in laws versions have been identified so far. As a result of this research, is given a comparative containing the difficulties and concerns about the DSL user experience, obtained between 4 (four) groups of users with distinct academic and experience backgrounds. After this analysis, some improvements in the language were implemented, regarding the users comments and suggestions. Furthermore, with the API tests, it was possible to compare the results of the language implementation with the historical processing data present in 16 selective processes. Finally, it is concluded that the DSL Cotas can contribute to the improvement of communication and understanding between the different users knowledges profiles, in addition to proposing a new means of practical-theoretical approach using domain specific languages. The social relevance of this study is emphasized, regarding institutional actions to meet demands that address social inclusion issues as a priority in the students selectives processes.

Key-words: Domain Specific Language. Meta Programming System. Vacancy Reservation Law 12.711/2012. Candidates Classification Rules. Federal Education Network.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AST	Árvore de Sintaxe Abstrata
BNF	Backus Naur Form
DEING	Departamento de Ingresso
DSL	Domain Specific Language
DTIC	Diretoria de Tecnologia da Informação e Comunicação
EMF	Eclipse Modeling Framework
ENEM	Exame Nacional do Ensino Médio
GPL	Linguagens de Propósito Geral
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	Integrated Development Environment
IFSC	Instituto Federal de Santa Catarina
JOVIAL	Jules Own Version of the International Algorithmic Language
MEC	Ministério da Educação
MPS	Meta Programming System
MVC	Model View Controller
PCD	Pessoas Com Deficiência
PPI	Candidatos autodeclarados Pretos, Pardos ou Indígenas
SDF	Syntax Definition Formalism
SERES	Secretaria de Regulação e Supervisão da Educação Superior
SISU	Sistema de Seleção Unificada
UML	Unified Modeling Language
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Este trabalho apresenta uma pesquisa realizada na área de engenharia de software com o intuito de padronizar as diretrizes presentes na legislação que definem regras de classificação de candidatos a cursos da rede de ensino pública federal. Desse modo, procura-se reduzir problemas durante as etapas de especificação e implementação do algoritmo de classificação de candidatos.

Para Ghosh (2011), problemas na comunicação entre desenvolvedores e especialistas de domínio são os principais motivos de falha no desenvolvimento e evolução de projetos de software. Especialistas entendem a terminologia do domínio e falam em um vocabulário que pode ser estranho para as equipes de desenvolvedores. Nesse sentido, é preciso identificar meios para redução da lacuna semântica entre especialistas e desenvolvedores, de modo que os especialistas possam estar envolvidos na verificação de regras de negócio ao longo do ciclo de vida do projeto.

A constante alteração de regras de negócio advém de mudanças de lei, forças de mercado ou novos objetivos empresariais, exigindo que essas possam ser expressas de modo a serem compreendidas e facilmente alteradas pelas organizações, a fim de melhorar o desempenho de negócios (CHRIS; DANIELA, 2002).

Segundo Berstel Silva (2013), a mudança de regras de negócio é inevitável, sendo cada vez mais frequente a adaptação às novas regulamentações e a necessidade de os sistemas estarem em conformidade com novas regras, que por décadas são implementadas em softwares de áreas industriais, financeiras, empresas de seguros, administração e várias outras.

O IFSC é uma instituição da rede de ensino pública federal presente no estado de Santa Catarina há mais de 100 anos, que hoje desenvolve e mantém diversos sistemas de informação, os quais estão sujeitos a alterações para conformidade à legislação federal. As aplicações institucionais têm como principais envolvidos os discentes, os professores e os técnicos administrativos, que participam da oferta de cursos nos mais diversos níveis, tais como cursos de qualificação profissional, educação de jovens e adultos, técnicos, superiores e pós-graduação.

Ao que concerne à necessidade de adaptação a novas regras de negócio, o autor da presente pesquisa trata sobre um dos sistemas de informação mais utilizados internamente, o sistema de ingresso de discentes na instituição, o qual é mantido desde 2007 pela Diretoria de Tecnologia da Informação e Comunicação (DTIC) e tem como objetivo disponibilizar ofertas de vagas em cursos por meio de processos seletivos. Atualmente, a base de dados do sistema conta com 788 processos seletivos e mais de 409 mil registros de candidatos.

A principal funcionalidade do sistema de ingresso do IFSC é a classificação de candidatos, a qual é construída e mantida conforme critérios da Lei nº 12.711/2012, que estabelece:

> Art. 1º As instituições federais de educação superior vinculadas ao Ministério da Educação reservarão, em cada concurso seletivo para ingresso nos cursos de graduação, por curso e turno, no mínimo 50% (cinquenta por cento)

> de suas vagas para estudantes que tenham cursado integralmente o ensino médio em escolas públicas.
>
> Art. 3º Em cada instituição federal de ensino superior, as vagas de que trata o art. 1º desta Lei serão preenchidas, por curso e turno, por autodeclarados pretos, pardos e indígenas e por pessoas com deficiência, nos termos da legislação, em proporção ao total de vagas no mínimo igual à proporção respectiva de pretos, pardos, indígenas e pessoas com deficiência na população da unidade da Federação onde está instalada a instituição, segundo o último censo da Fundação Instituto Brasileiro de Geografia e Estatística - IBGE (BRASIL, 2012).

A adequação à legislação é demanda proveniente do Ministério da Educação (MEC), o qual faz referência em seu site aos decretos nº 7.824/2012, nº 9.034/2017 (que regulamentam a lei nº12.711) e às portarias normativas nº 18/2012 e nº 9/2017, as quais estabelecem regras, conceitos básicos e fórmulas para preenchimento das modalidades de reservas de vagas do sistema de cotas.

Sempre que o MEC realiza alterações em um dos documentos de lei citados, é necessário também alterar os sistemas de informações das instituições de ensino. A exigência para adequação dos sistemas à lei traz o aumento da demanda de desenvolvimento, situação ilustrada no Capítulo 2, no qual é elencado o histórico do controle de versão para 3 (três) versões já implementadas no IFSC.

1.1 MOTIVAÇÃO

O estudo de linguagens que sejam capazes de expressar regras de maneira mais clara não é novidade na área da engenharia de sistemas. Bentley (1986) já demonstrava em sua pesquisa diferentes abordagens de construção de pequenas linguagens para facilitar a escrita de programas gráficos e interface gráfica.

Wexelblat L. (1981), apresenta o termo "linguagem de propósito especial", quando se refere à Jules Own Version of the International Algorithmic Language (JOVIAL), linguagem de alto nível destinada principalmente para auxiliar na programação de grandes sistemas complexos em tempo real.

Um exemplo de aplicação dessas linguagens pode ser visto na Figura 1, em que o compilador transforma comandos simples no formato textual em formato de digrama (Figura 2), abstraindo detalhes específicos das linguagens tradicionais de programação da época.

Na engenharia de software atual, as Linguagens de Domínio Específico ou DSLs, do inglês *Domain Specific Languages*, estão se tornando cada vez mais importantes e as novas ferramentas de criação dessas linguagens tem evoluído, reduzindo o esforço de desenvolvimento (VOELTER et al., 2013).

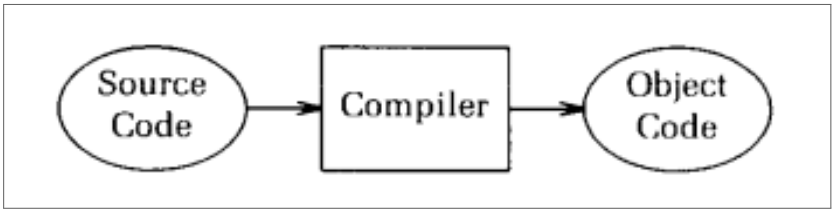
A equipe de desenvolvedores do IFSC possui uma alta demanda de desenvolvimento de sistemas e serviços internos, atualmente são mantidos mais de 10 sistemas em uso, alguns deles subdivididos em vários módulos (PRIMAO PACHECO, 2019). A DTIC centraliza e mantém

Figura 1 – Comandos na PIC Language

```
ellipse "Source" "Code"
arrow
box "Compiler"
arrow
ellipse "Object" "Code"
```

Fonte: Bentley (1986).

Figura 2 – Diagrama resultado após compilação



Fonte: Bentley (1986).

a maioria dos sistemas e serviços de TI, contando com apenas 12 analistas da tecnologia da informação, além de prestar suporte para todas as 23 unidades da instituição.

Como agravante, o sistema de ingresso foco deste trabalho, foi criado em meados dos anos 2000 por bolsistas que já não estão mais na instituição, e atualmente apenas 2 (dois) desenvolvedores são responsáveis pelas demandas de desenvolvimento e suporte. Por se tratar de um sistema legado criado em linguagem PHP sem qualquer preocupação com documentação ou qualidade de código, o custo de alterações mais complexas como no caso de regras de classificação acaba por atrasar a adequação aos novos requisitos de lei.

Até o momento, a presente equipe participou do desenvolvimento de 3 (três) versões do algoritmo de classificação, além de prestar manutenção corretiva em função de entendimentos equivocados nas regras implementadas. Os algoritmos envolvidos, assim como o seu histórico de versionamento são detalhados no Capítulo 2.

Esse trabalho tem como objetivo reduzir as lacunas de entendimento entre os especialistas na legislação e os desenvolvedores, por meio da criação de uma Domain Specific Language (DSL), mitigando os problemas de comunicação que acabam por atrasar o processo de desenvolvimento. A exemplo de aplicação de DSLs, destacam-se linguagens como a `LegalLanguage` (SOARES; MARTINS; SILVA, 2019), a `Ergo DSL` (ACCORD, 2019) e a `DSL Capgemini Pension` (KOLK; VOLTER, 2008). Essas linguagens estabelecem modelos para apoio em processos legislativos e regulatórios, sistematizando regras e jargões utilizados pelos usuários especialistas de domínio, possibilitando a execução de processamentos, validações e simulações, com notações claras e livres de termos técnicos presentes nas linguagens de desenvolvimento tradicionais.

Essas linguagens são apresentadas no Capítulo 7.

Nesse contexto, essa pesquisa apresenta uma Linguagem de Domínio Específico que permite a especificação de regras de classificação de candidatos, na qual usuários especialistas do sistema de ingresso possam estabelecer as categorias de cotas, com objetivo de reduzir o esforço do desenvolvimento e de entendimento das regras de negócio por parte de usuários e de desenvolvedores.

Nas Seções 1.2, 1.3 e 1.4, são apresentados, respectivamente, o problema de pesquisa e as hipóteses, os objetivos e, por fim, a organização do trabalho.

1.2 PROBLEMA DE PESQUISA

Essa pesquisa surge em decorrência das necessidades recentes para refatoração do código fonte do sistema de ingresso do IFSC. Essas necessidades foram resultadas em função de alterações nos documentos de lei e em função de divergências sobre o entendimento de distribuição de vagas entre desenvolvedores e especialistas de negócio. Essas questões acabam atrasando o processo de implementação do sistema para aderência à uma nova legislação ou à uma instrução normativa definida pelos órgãos de controle.

Do mesmo modo, essas situações trazem alguns desafios para a equipe de desenvolvedores e para os *stakeholders* que analisam a legislação e definem os requisitos do sistema de convocação de candidatos cotistas nos cursos do IFSC.

No que concerne ao problema de pesquisa destaca-se: Como utilizar uma linguagem de definição de regras do sistema de cotas de modo a melhorar a comunicação entre usuários especialistas e desenvolvedores e, assim, verificar a possibilidade de melhoria no processo de desenvolvimento das regras presentes na legislação?

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Compreender, por meio da elaboração de uma linguagem específica de domínio, a viabilidade de melhoria na comunicação entre usuários de negócio e desenvolvedores, visando o aumento na produtividade da especificação de requisitos e da implementação de regras concernentes ao sistema de cotas da rede de ensino pública federal.

1.3.2 Objetivos Específicos

a) Realizar levantamento do histórico de versões presentes no sistema de controle de versão do IFSC sobre regras de classificação de cotas;

b) Analisar e identificar as características em comum entre as versões do algoritmo implementadas para que seja possível elaborar a modelagem da DSL proposta;

c) Definir e implementar uma DSL para usuários especialistas nas regras do sistema de cotas;

d) Criar um cenário de avaliação da DSL, a fim de validar o seu entendimento por usuários de negócio e desenvolvedores;

e) Implementar uma API utilizando as regras definidas pelos usuários de negócio para geração do algoritmo de classificação pelo sistema de cotas.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho é organizado nos seguintes capítulos:

- Capítulo 1: Introdução - que apresenta a motivação, a contextualização, os objetivos e a organização do trabalho;
- Capítulo 2: Sistema de Ingresso e Versionamento - que elenca o histórico de mudanças no sistema de cotas do sistema de ingresso do IFSC, o que motivou o estudo inicial desta pesquisa;
- Capítulo 3: Fundamentação Teórica - o qual aborda a revisão da literatura sobre os temas ligados ao desenvolvimento de DSL na engenharia de software, as diferenças sobre Linguagens de Propósito Geral (GPL), as vantagens e desvantagens do uso DSL, assim como as ferramentas utilizadas para a sua construção;
- Capítulo 4: DSL de Cotas - que aborda uma DSL como meio de simplificar a especificação e desenvolvimento de regras de classificação de candidatos ao sistema de cotas da rede de ensino pública federal, assim como apresenta a API DSL Cotas, responsável por implementar o serviço de classificação e aprovação de candidatos;
- Capítulo 5: Metodologia - no qual é descrita as justificativas das tecnologias adotadas, a classificação e as etapas da pesquisa, o ambiente da pesquisa e os métodos de avaliação da DSL;
- Capítulo 6: Análise dos Resultados da Pesquisa - o qual contém a análise dos dados coletados por meio de um exercício prático de avaliação da DSL Cotas, de um questionário aplicado com os usuários e da avaliação da API DSL Cotas;
- Capítulo 7: Conclusões - na qual é realizada uma síntese dos principais resultados da pesquisa, bem como os trabalhos relacionados, o espoco negativo da pesquisa, as principais contribuições da DSL Cotas, os trabalhos futuros sugeridos e as considerações finais.

2 SISTEMA DE INGRESSO E VERSIONAMENTO

Neste capítulo são descritas as informações sobre as funcionalidades desenvolvidas no sistema de ingresso do IFSC com relação aos requisitos e algoritmos do sistema de cotas. Para esse fim, foi utilizado o histórico do controle de versão no que concerne ao quantitativo de arquivos, classes, funções e as diferentes versões desde o surgimento da demanda de cotas na legislação.

2.1 HISTÓRICO DE PROJETO

Criado em meados do ano de 2000, o sistema de ingresso do IFSC, tem por objetivo disponibilizar vagas de cursos para os discentes do IFSC. Esse sistema foi desenvolvido internamente na linguagem PHP, para automatizar os processos seletivos que eram realizados por meio de planilhas e ferramentas não integradas, as quais demandavam ao setor responsável muitas pessoas e muitos procedimentos operacionais repetitivos, gerando falhas no processo por erro humano.

O projeto não utiliza conceitos de orientação a objetos, em sua maioria os arquivos PHP ultrapassam duas mil linhas, sem divisão em camadas Model View Controller (MVC), com combinações das linguagens Javascript, HTML e PHP no mesmo arquivo. Quando era preciso criar ou adaptar alguma nova funcionalidade por falta de conhecimento técnico, os antigos desenvolvedores (bolsistas) faziam a cópia das funcionalidades para vários locais do sistema, sem pensar em reutilização de código.

Com o objetivo de elencar a situação atual do código fonte do sistema, neste trabalho são apresentados os quantitativos levantados a partir do sistema de controle de versão do IFSC. A Tabela 1 apresenta o levantamento geral sobre o total de arquivos, linhas de código, commits e desenvolvedores que já atuaram no projeto. Nas Seções seguintes são contextualizados os dados gerais sobre o algoritmo de classificação, assim como é descrito o levantamento feito nas 3 (três) versões do algoritmo de classificação que foram implementadas até o momento.

Tabela 1 – Dados gerais do controle de versão

Total de arquivos	1.357 arquivos (php, html, css, js)
Total de linhas de código	369.414 linhas
Total de commits	731
Total de desenvolvedores	6
Período da coleta	11/02/2015 - 17/04/2019

Fonte: Elaborada pelo autor (2019).

2.2 ALGORITMO DE CLASSIFICAÇÃO

Nesta Seção são detalhados os conceitos e as etapas do processo de classificação de candidatos às vagas do sistema de cotas, assim como o levantamento de cenários ilustrativos sobre a distribuição de vagas tendo como base as versões já implementadas no sistema de ingresso.

Para cada processo seletivo há uma lista de cursos, em que o setor que gerencia as vagas define-as com base no valor total inicial e indica se o processo seletivo vai utilizar a regra de classificação por cotas. Durante as inscrições são apresentados aos candidatos os campos necessários para indicar se vieram de escola pública, bem como informar a sua renda familiar e se autodeclararem pretos, pardos ou indígenas.

Após o término do período de inscrições, os candidatos são separados em categorias de concorrência de acordo com o preenchimento realizado no ato da inscrição. A seguir, os candidatos participam do processo seletivo de forma física como provas de vestibular ou processos eletrônicos de classificação, como por exemplo: sorteio, pontuação por preenchimento de formulários sócio-econômicos ou classificação por Exame Nacional do Ensino Médio (ENEM) ou Sistema de Seleção Unificada (SISU). Por fim é gerado um número de classificação que representa a ordem dos candidatos que disputam as vagas disponíveis.

O algoritmo de classificação utiliza como parâmetro de entrada a lista de candidatos, contendo o número de inscrição, a ordem de classificação geral, a data de nascimento (como critério de desempate), a quantidade de vagas total do curso, o percentual de vagas disponíveis para escola pública e o percentual de proporção do Instituto Brasileiro de Geografia e Estatística (IBGE).

Esses parâmetros variam conforme a Unidade Federada e são fornecidos pelo último censo demográfico, representando o percentual sobre o total de pretos, pardos e indígenas no estado em relação às demais categorias de cotas para estudantes da escola pública.

Com esses parâmetros de entrada, o algoritmo gera um quadro de vagas contendo quantas vagas estão reservadas para às respectivas categorias de cotas, e faz a seleção e aprovação de candidatos de acordo com a sua classificação e critérios de desempate. Por fim, em caso de sobra de vagas por falta de candidatos para uma determinada categoria de cota, o algoritmo faz nova busca por candidatos de outra categoria, de acordo com a ordem de prioridade estabelecida na portaria nº 18 de 2012 do MEC, a qual define:

> Art. 15. No caso de não preenchimento das vagas reservadas aos autodeclarados pretos, pardos e indígenas e às pessoas com deficiência, aquelas remanescentes serão preenchidas pelos estudantes que tenham cursado integralmente o ensino fundamental ou médio, conforme o caso, em escolas públicas, observadas as reservas realizadas em mesmo nível ou no imediatamente anterior, nos termos do art. 10 desta Portaria. (MEC, 2012)

No fim do processo de classificação é gerada uma lista de candidatos aprovados, incluindo a classificação geral e a classificação na respectiva categoria de cota, por fim esta lista é

enviada ao sistema acadêmico para que os candidatos possam realizar a matrícula e entregar a documentação necessária.

As alterações nos documentos de lei podem incluir novas categorias, novas formas de distribuição das vagas iniciais, mudanças de percentuais, formas de arredondamento, descrição dos tipos de cotas e mudança na ordem de prioridade em caso de sobra de vagas. Essas situações são exemplificadas nas Seções seguintes por meio dos casos e cenários nos quais houve maior impacto de refatoração de código do sistema de ingresso para adequação à legislação.

2.2.1 Versão 1 - Início do sistema de Cotas, Lei Nº 12.711 de 2012

Inicialmente, o sistema de ingresso foi construído sem considerar cotas, ou seja, a classificação era por pontuação ou nota de vestibular, não havia exigência de lei em função de reserva de vagas. As chamadas para as matrículas eram realizadas pela ordem geral de classificação e alguns critérios de desempate. Hoje apenas alguns tipos de processos seguem este molde, como por exemplo, cursos de curta duração ou cursos internos que não se aplicam à legislação.

Com o surgimento da lei nº 12.711 de 2012, vem também a demanda do governo para que as instituições reservem 50% das vagas para estudantes do ensino médio que estudaram exclusivamente em escolas públicas, e dentro desses 50% deveria ser reservado um percentual para estudantes cuja renda familiar fosse inferior a 1.5 salários mínimos per capita:

> Parágrafo único. No preenchimento das vagas de que trata o caput deste artigo, 50% (cinquenta por cento) deverão ser reservados aos estudantes oriundos de famílias com renda igual ou inferior a 1,5 salário-mínimo (um salário-mínimo e meio) per capita.
>
> Art. 3o Em cada instituição federal de ensino superior, as vagas de que trata o art. 1o desta Lei serão preenchidas, por curso e turno, por autodeclarados pretos, pardos e indígenas, em proporção no mínimo igual à de pretos, pardos e indígenas na população da unidade da Federação onde está instalada a instituição, segundo o último censo do Instituto Brasileiro de Geografia e Estatística (IBGE) (BRASIL, 2012).

Com esse objetivo, o algoritmo de classificação, que era uma consulta SQL, foi separado em 3 funções PHP: `calcula_vagasAcoesAfirmativas`, `aprova_Candidatos` e `retorna_OrdemdePreenchimentodeVagasNaoOcupadas`. A primeira função faz a geração do quadro de vagas; a segunda, a seleção dos candidatos para serem aprovados e atualizar a contagem de vagas em cada categoria e, a última, retorna a ordem de preenchimento em caso de sobra de vagas.

Tendo em vista que há uma divisão categórica das vagas entre os candidatos, foi preciso criar 5 (cinco) tipos de situações de classificação para cada combinação de cotas possível, definidas conforme a Tabela 2.

Tabela 2 – Lista de categorias de cotas da versão 1

CLAG	Ampla concorrência ou classificação geral (todos os candidatos concorrem)
EPRIPPI	Candidatos que estudaram em Escola Pública com Renda Inferior a 1.5 salários mínimos per capita e autodeclarados Pretos, Pardos ou Indígenas
EPRINPPI	Candidatos que estudaram em Escola Pública com Renda Inferior a 1.5 salários mínimos per capita NÃO são autodeclarados Pretos, Pardos ou Indígenas;
EPRSPPI	Candidatos que estudaram em Escola Pública com Renda Superior a 1.5 salários mínimos per capita e autodeclarados Pretos, Pardos ou Indígenas
EPRSNPPI	Candidatos que estudaram em Escola Pública com Renda Superior a 1.5 salários mínimos per capita NÃO são autodeclarados Pretos, Pardos ou Indígenas

Fonte: Sistema de controle de versão do IFSC (2015).

Dadas as situações de classificação possíveis, o primeiro passo do algoritmo é gerar um quadro de vagas para cada tipo de cota, tendo como base o percentual do IBGE e o total de vagas para o curso. O trecho de código é apresentado no Código Fonte 1.

Código Fonte 1 – Função que calcula o quadro de vagas

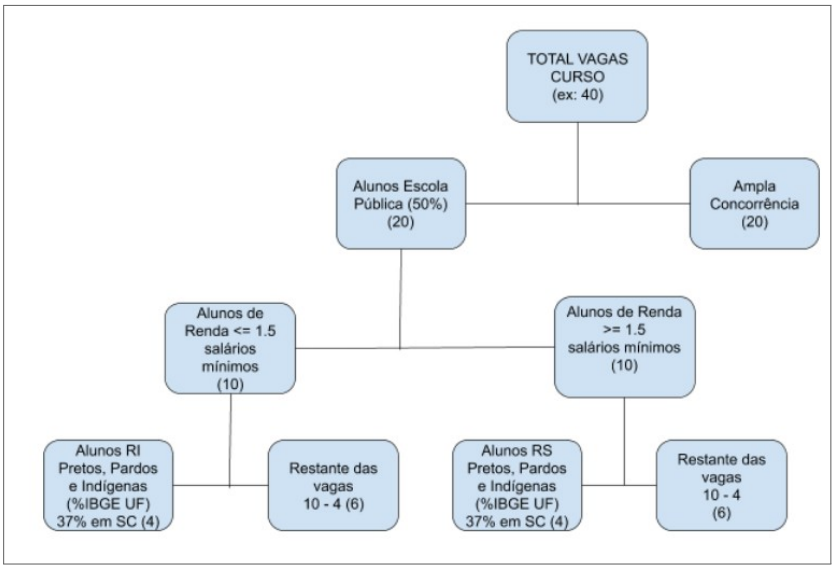
```
function calcula_vagasAcoesAfirmativas($totalVagas, $percentualNegros,
    $percentualEscolaPublica, $percentualPPIIBGE){
    $vagas = array();
    $multEscolaPublica = $percentualEscolaPublica/100;
    $multPPIIBGE = $percentualPPIIBGE/100;
    // vagas escola publica
    $vagas['AAEP'] = ceil($totalVagas * $multEscolaPublica);
    // restante para ampla concorrencia
    $vagas['CLAG'] = $totalVagas - $vagas['AAEP'];
    // vagas escola publica renda < 1.5 SM
    $vagas['AAEPRI'] = ceil($vagas['AAEP'] * 0.5);
    // restante para escola publica renda > 1.5 SM
    $vagas['AAEPRS'] = $vagas['AAEP'] - $vagas['AAEPRI'];
    // vagas escola publica renda < 1.5 SM - PPI
    $vagas['AAEPRIPPI'] = ceil($vagas['AAEPRI'] * $multPPIIBGE);
    // restante para escola publica renda < 1.5 SM - NPPI
    $vagas['AAEPRINPPI'] = $vagas['AAEPRI'] - $vagas['AAEPRIPPI'];
    // vagas escola publica renda > 1.5 SM - PPI
    $vagas['AAEPRSPPI'] = ceil($vagas['AAEPRS'] * $multPPIIBGE);
    // restante para escola publica renda > 1.5 SM - NPPI
    $vagas['AAEPRSNPPI'] = $vagas['AAEPRS'] - $vagas['AAEPRSPPI'];
    return $vagas;}
```

Basicamente, o que a função `calcula_vagasAcoesAfirmativas` faz é separar as vagas, conforme os seguintes passos:

1. Linha 1 - Dado o total de vagas do curso;
2. Linha 3 - Obter o percentual de escola pública;
3. Linha 4 - Obter o percentual de proporção de PPI do IBGE no estado de oferta;
4. Linha 6 - Multiplicar o total de vagas pelo percentual de Escola Pública (50%) e armazenar o valor em AAEP;
5. Linha 8 - Obter o total reservado para ampla concorrência (CLAG), diminuindo o total de vagas pelo reservado ao sistema de cotas AAEP;
6. Linha 12 - Dividir o total de vagas AAEP em 50% para candidatos de Renda Inferior a 1.5 (AAEPRI) e 50% para candidatos de Renda Superior a 1.5 (AAEPRS);
7. Linha 14 - Dentro do total de vagas AAEPRI, deve-se calcular a proporção reservada pelo IBGE para cotistas PPI (Preto, Pardo, Indígenas) que também possuam Renda Inferior a 1.5 salários mínimos e armazenar em AAEPRIPPI;
8. Linha 16 - O total de vagas de estudantes de Renda Inferior que NÃO são autodeclarados PPI (AAEPRINPPI) é obtido após diminuir o total de vagas para renda inferior AAEPRI - o total armazenado em (AAEPRIPPI);
9. Linha 18 - Dentro do total de vagas AAEPRS, deve-se calcular a proporção reservada pelo IBGE para cotistas PPI (Preto, Pardo, Indígenas) que também possuam Renda Superior a 1.5 salários mínimos e armazenar em AAEPRSPPI;
10. Linha 20 - O total de vagas de estudantes de Renda Superior que NÃO são autodeclarados PPI (AAEPRSNPPI) é obtido após diminuir o total de vagas para renda inferior AAEPSI - o total armazenado em (AAEPRSPPI).

Um exemplo do resultado da geração do quadro de vagas para um cenário de curso com 40 vagas, pode ser visto na Figura 3.

Figura 3 – Cenário de distribuição versão 1



Fonte: Elaborada pelo autor (2019).

Tendo como base o quadro de vagas, o algoritmo de classificação de candidatos utiliza a função aprova_Candidatos descrita no Código Fonte 2, fazendo a seleção dos candidatos para aprovação em cada categoria até atingir o total de vagas disponíveis.

Código Fonte 2 – Função de aprovação de candidatos

```
function aprova_Candidatos($idProcesso, $idCurso, $situacoesdeClassificacao, $vagas)
{
    if(!is_array($situacoesdeClassificacao))
    {
        $situacoesdeClassificacao = array($situacoesdeClassificacao);
    }

    foreach($situacoesdeClassificacao as $situacaodeClassificacao)
    {
        if($vagas == 0)
        {
            //Para quando completar as vagas
            break;
        }

        $strSQL = 'SELECT can.idCandidato AS idCandidato
                  FROM candidatos can
                  WHERE can.idProcesso=\'' . $idProcesso . '\'
                  AND can.curso=\'' . $idCurso . '\'
                  AND can.situacao=\'CLA\'';

        switch($situacaodeClassificacao)
        {
            case 'CLAG':
                break;
            case 'AAEPRIPPI':
                $strSQL .= ' AND can.cotaEscolaPublica=\'S\'
                AND can.cotaRendaInferior=\'S\'
```

```
29                  AND can.cotaPPI=\'S\' ';
                    break;
31              case 'AAEPRINPPI':
                    $strSQL .= ' AND can.cotaEscolaPublica=\'S\'
33                  AND can.cotaRendaInferior=\'S\'
                    AND can.cotaPPI=\'N\' ';
35                  break;
                case 'AAEPRSPPI':
37                  $strSQL .= ' AND can.cotaEscolaPublica=\'S\'
                    AND can.cotaRendaInferior=\'N\'
39                  AND can.cotaPPI=\'S\' ';
                    break;
41              case 'AAEPRSNPPI':
                    $strSQL .= ' AND can.cotaEscolaPublica=\'S\'
43                  AND can.cotaRendaInferior=\'N\'
                    AND can.cotaPPI=\'N\' ';
45                  break;
                default:
47                  return -1;
            }
49
            $strSQL .= ' ORDER BY can.classificacao ASC
51          LIMIT 0,' . $vagas;

53          $resultado = dbQuery($strSQL);

55          while($linha = mysql_fetch_array($resultado))
            {
57              $strSQL2 = 'UPDATE candidatos
                SET situacao=\'APV\',
59              situacaodeClassificacao=\'' . $situacaodeClassificacao . '\'
                WHERE idCandidato=' . $linha['idCandidato'];
61
                dbQuery($strSQL2);
63
                $vagas--;
65          }
        }
67
    return $vagas;
69  }
```

Por fim, por motivo de falta de candidatos, o algoritmo utiliza a função ordemdePreenchimentodeVagasNaoOcupadas para obter a prioridade por lei quando sobra vaga de uma determinada categoria de cota (Código Fonte 3).

Código Fonte 3 – Função de prioridade em caso de sobra de vagas

```
function retorna_OrdemdePreenchimentodeVagasNaoOcupadas($situacaodeClassificacao){
    $ordem = array();
    switch($situacaodeClassificacao){
        case 'AAEPRIPPI':
            $ordem = array('AAEPRINPPI', 'AAEPRSPPI', 'AAEPRSNPPI', 'CLAG');break;
        case 'AAEPRINPPI':
            $ordem = array('AAEPRIPPI', 'AAEPRSPPI', 'AAEPRSNPPI', 'CLAG');break;
        case 'AAEPRSPPI':
            $ordem = array('AAEPRSNPPI', 'AAEPRIPPI', 'AAEPRINPPI', 'CLAG'); break;
        case 'AAEPRSNPPI':
            $ordem = array('AAEPRSPPI', 'AAEPRIPPI', 'AAEPRINPPI', 'CLAG');break;
        default:
            break;
    }
    return $ordem;
}
```

Após análise no controle de versão pode-se identificar que as funções eram utilizadas em diferentes etapas e tipos de processo do sistema. No entanto, a equipe desenvolvedora não teve preocupação na organização do código e fez a cópia em vários pontos do sistema, trazendo ainda mais problemas de manutenção.

Os arquivos envolvidos, assim como os dados do controle de versão são listados nas Tabelas 3, 4, 5 e 6. Cada Tabela apresenta a quantidade de linhas de código de classificação e de aprovação de candidatos do sistema de ingresso, ademais esse levantamento apresenta a quantidade de commits relevantes registrados para questões relacionadas às funcionalidades do sistema de cotas, assim como o número de desenvolvedores envolvidos no processo de manutenção do código fonte. Para a realização desse levantamento foi utilizada a ferramenta phploc, essa ferramenta permite fazer a medição do tamanho e análise de estrutura para projetos e arquivos PHP (BERGMANN, 2020).

Tabela 3 – Versionamento do arquivo corrigir.php

ingresso/admin/admin/correcao/corrigir.php		
Total de linhas de código	1129	
Total de funções utilizadas	267 funções (incluídas e importadas)	
Total de funções / algoritmo de cotas	5 funções	
Número de linhas envolvidas / algoritmo de cotas	**processar_Correcao**	96 linhas de código
	calcula_vagasAcoesAfirmativas	33 linhas de código
	aprova_Candidatos	73 linhas de código
	retorna_OrdemdePreenchimentodeVagasNaoOcupadas	27 linhas de código
	alimenta_Classificacao	35 linhas de código
Commits relevantes	31 commits desde 11/02/2015	
Número de programadores envolvidos nos commits	2	
Total de linhas de código para a implementação de cotas	264 linhas	

Fonte: Elaborada pelo autor (2019).

Tabela 4 – Versionamento do arquivo informar_classificacao01_sorteio.php

ingresso/admin/admin/semprova/informar_classificacao01_sorteio.php	
Total de linhas de código	1360
Total de funções utilizadas	236 funções (incluídas e importadas)
Total de funções / algoritmo de cotas	5 funções
Commits relevantes	61 commits desde 11/02/2015
Número de programadores envolvidos nos commits	2

Fonte: Elaborada pelo autor (2019).

Tabela 5 – Versionamento do arquivo informar_classificacao01.php

ingresso/admin/admin/semprova/informar_classificacao01.php	
Total de linhas de código	1162
Total de funções utilizadas	88 funções (incluídas e importadas)
Total de funções / algoritmo de cotas	6 funções
Commits relevantes	14 commits desde 11/02/2015
Número de programadores envolvidos nos commits	2

Fonte: Elaborada pelo autor (2019).

Tabela 6 – Versionamento do arquivo semprova/classificacao01_sorteio.php

ingresso/admin/admin/semprova/informar_classificacao01_sorteio.php	
Total de linhas de código	1363
Total de funções utilizadas	119 funções (incluídas e importadas)
Total de funções / algoritmo de cotas	6 funções
Commits relevantes	24 commits desde 11/02/2015
Número de programadores envolvidos nos commits	3

Fonte: Elaborada pelo autor (2019).

Com base nesse levantamento pode-se observar uma quantidade considerável de código fonte envolvido para desenvolvimento de regras para classificação de candidatos. Como tentativa de dar celeridade no processo de desenvolvimento de correções e na criação de futuras versões, os problemas de código duplicado foram reduzidos por meio de refatoração nas versões posteriores do algoritmo de classificação. Na Subseção 2.2.2 são apresentados os dados sobre o controle de versão, após mudanças realizadas em função de novas demandas de alteração da legislação.

2.2.2 Versão 2 - Alteração para Lei Nº 13.409 de 2016

Em função da criação da Lei nº 13.409, que altera a lei original nº 12.711, foi feita uma re-estruturação do código para inclusão de novas categorias de cotas para Pessoas Com Deficiência (PCD), mantendo as outras categorias já existentes. Os trechos de lei afetados foram:

> Art. 3º Em cada instituição federal de ensino superior, as vagas de que trata o art. 1º desta Lei serão preenchidas, por curso e turno, por autodeclarados pretos, pardos e indígenas e por pessoas com deficiência, nos termos da legislação, em proporção ao total de vagas no mínimo igual à proporção respectiva de pretos, pardos, indígenas e pessoas com deficiência na população da unidade da Federação onde está instalada a instituição, segundo o último censo da Fundação Instituto Brasileiro de Geografia e Estatística - IBGE.
>
> Art. 5º Em cada instituição federal de ensino técnico de nível médio, as vagas de que trata o art. 4º desta Lei serão preenchidas, por curso e turno, por autodeclarados pretos, pardos e indígenas e por pessoas com deficiência, nos termos da legislação, em proporção ao total de vagas no mínimo igual à proporção respectiva de pretos, pardos, indígenas e pessoas com deficiência na população da unidade da Federação onde está instalada a instituição, segundo o último censo do IBGE (BRASIL, 2016).

A equipe de desenvolvimento original do sistema já não estava mais presente no setor e a falta de experiência nas regras de cotas por parte da equipe atual, acabou por dificultar e atrasar o processo de entendimento do código fonte para aplicação das mudanças no sistema, demandando também a alocação de mais desenvolvedores para divisão do trabalho de codificação dos novos tipos de cota.

Após a análise dos novos trechos de lei foi especificada uma nova versão do algoritmo de classificação, desenvolvida por meio de interpretação da área demandante e dos analistas de sistemas da DTIC, o que gerou uma nova implementação contendo 7 (sete) tipos de categorias de cotas possíveis, conforme Tabela 7:

Tabela 7 – Categorias de cotas na versão 2

<table>
<tr><td rowspan="6">Cotas (EP)
mínimo 50% das vagas</td><td rowspan="3">Renda Inferior do que 1,5 S.M.

RI 50% das cotas</td><td>PPI – Pretos Pardos e Índios*</td></tr>
<tr><td>PCD – Pessoas com Deficiência*</td></tr>
<tr><td>NPPID – Não incluídos nos casos acima**</td></tr>
<tr><td rowspan="3">Renda Superior a 1,5 S.M.

RS 50% das cotas</td><td>PPI – Pretos Pardos e Índios*</td></tr>
<tr><td>PCD – Pessoas com Deficiência*</td></tr>
<tr><td>NPPID – Não incluídos nos casos acima**</td></tr>
<tr><td colspan="3">Classificação Geral (CLAG)

vagas restantes além das cotas

máximo 50% das vagas</td></tr>
</table>

Fonte: Elaborada pelo autor (2019).

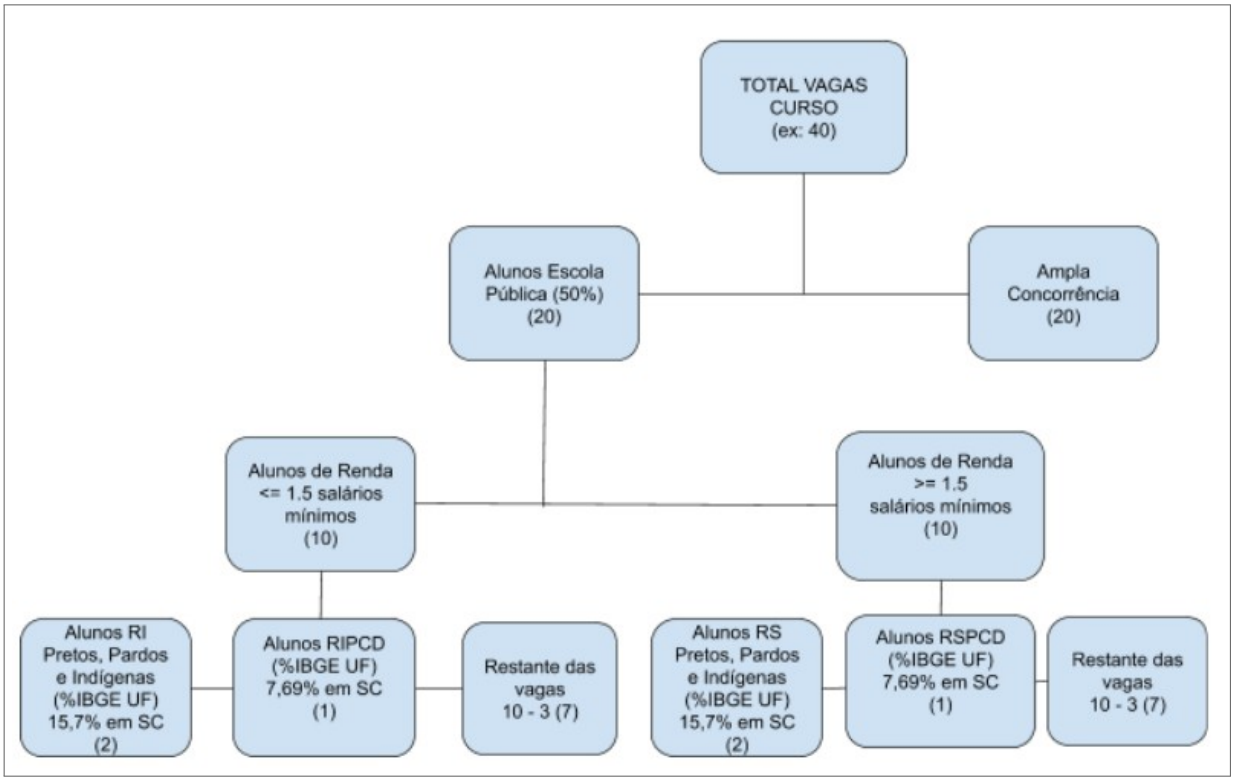
O desenvolvimento durou cerca de 3 (três) meses, com 2 (dois) desenvolvedores dedicados, os quais trabalharam em todos os arquivos descritos na primeira versão, assim como as telas envolvidas para atender a este novo entendimento e disponibilizar o Código Fonte 4:

Código Fonte 4 – Segunda versão do algoritmo

```
Vagas['EP'] = TV * PVEP
Vagas['CLAG'] = TV - Vagas['EP']
Vagas['RI'] = Vagas['EP'] * 0.5
Vagas['RIPPI'] = Vagas['RI'] * PIPPI
Vagas['RIPCD'] = Vagas['RI'] * PIPCD
Vagas['RINPPID'] = Vagas['RI'] -  (Vagas['RIPPI'] +Vagas['RIPCD'])
Vagas['RS'] = Vagas['EP'] -  Vagas['RI']
Vagas['RSPPI'] = Vagas['RS'] * PIPPI
Vagas['RSPCD'] = Vagas['RS'] * PIPCD
Vagas['RSNPPID'] = Vagas['RS'] -  (Vagas['RSPPI'] +Vagas['RSPCD'])
```

Esse algoritmo substitui o corpo da função `calcula_vagasAcoesAfirmativas`, detalhada no Código Fonte 1. Como resultado da geração do quadro de vagas na versão 2, a Figura 4 demonstra o novo cenário para um curso com 40 vagas.

Figura 4 – Cenário de distribuição versão 2



Fonte: Elaborada pelo autor (2019).

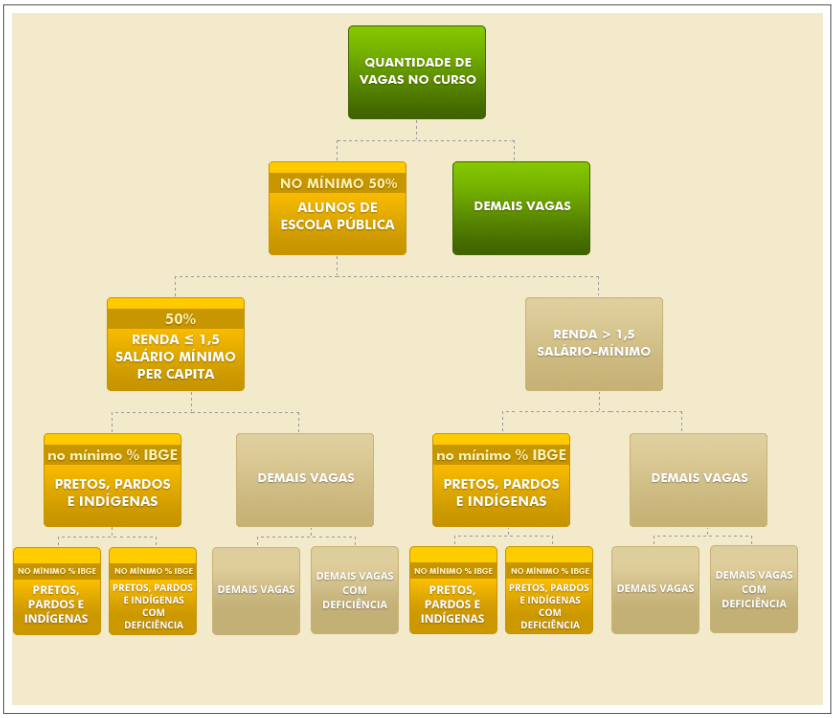
É importante ressaltar que o entendimento de lei repassado pela área demandante se deu sem definição formal do MEC, a análise foi procedida por meio de interpretação própria do IFSC, e também não havia sido definido como priorizar a sobra de vagas entre cotas, uma vez que a portaria nº 18/2012/MEC demorou a ser atualizada. Mesmo sem definição clara, o IFSC aplicou em seus processos a nova regra de cotas, incluindo a reserva para PCD.

Após alguns meses em funcionamento desta versão, o MEC em resposta oficial, em função das dúvidas das instituições, acabou por modificar novamente a distribuição de vagas, que agora seria em 9 (nove) categorias de cotas diferentes, e também foram atualizadas as regras de priorização em caso de sobra de vaga. Nesse sentido, na Subseção 2.2.3 são apresentados os dados de desenvolvimento da versão atualmente utilizada nos processos seletivos.

2.2.3 Versão 3 - Reimplementação para interpretação do MEC em 2017

Com objetivo de unificar o entendimento da distribuição de cotas, o MEC lançou em seu portal as alterações e detalhes sobre como deveria ser implementada a nova regra de ingresso em vestibulares e processos seletivos da rede federal. Na Figura 5, pode-se observar a nova divisão, que agora abrange Candidatos autodeclarados Pretos, Pardos ou Indígenas (PPI) que possuam alguma deficiência e PPI que não possuam deficiência, tanto para os estudantes de escolas públicas com renda inferior, como estudantes de renda superior a 1,5 salários mínimos per capita.

Figura 5 – Novo procedimento de aplicação do sistema de cotas



Fonte: MEC (2018).

Na versão anterior da implementação, o discente teria que optar por concorrer entre as opções: PPI, somente PCD ou nas demais vagas, causando problemas durante as inscrições de várias instituições de ensino, pois havia candidatos que eram tanto PPI como PCD e tinham direito a concorrer nas duas categorias de cota.

Tendo em vista a correção do entendimento feita pelo MEC, foi preciso criar 9 (nove) tipos de situações de classificação para cada combinação de cotas possível (Tabela 8).

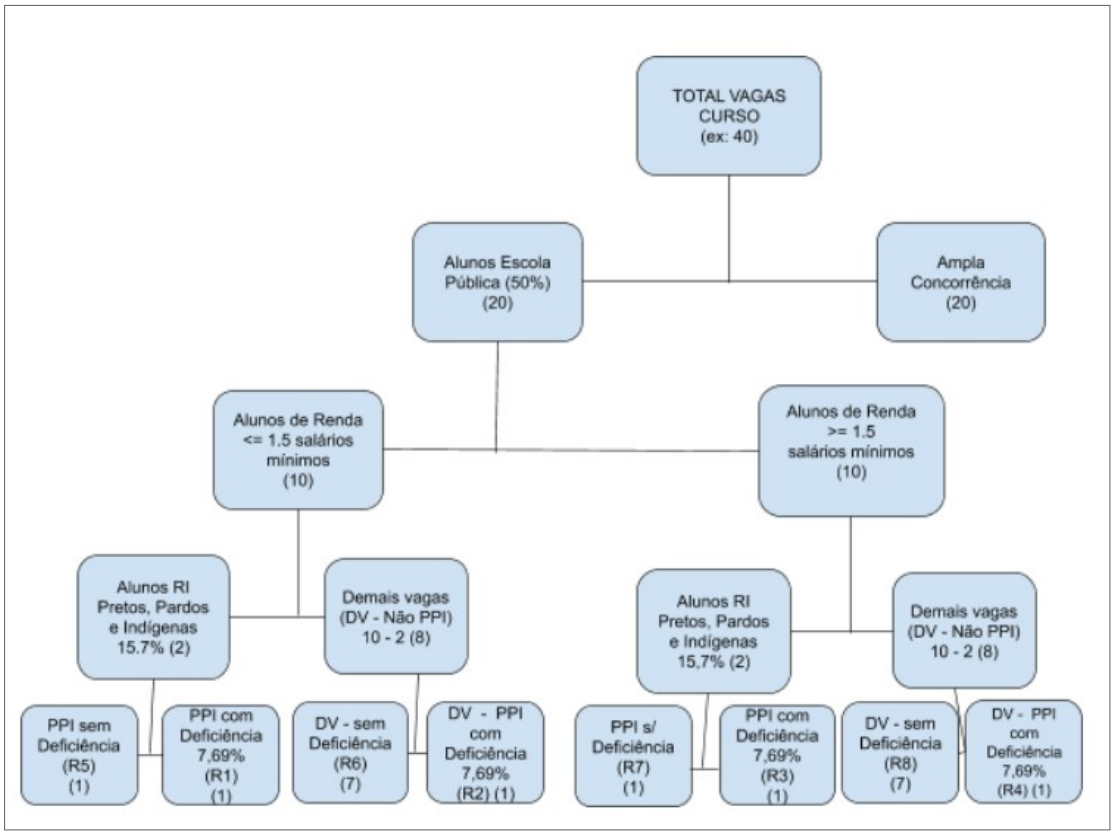
Tabela 8 – Lista de categorias de cotas da versão 3

CLAG	Ampla concorrência ou classificação geral
RIPPIPCDR1	Renda igual ou inferior a 1,5 (um vírgula cinco) salário-mínimo per capita autodeclarados pretos, pardos ou indígenas com deficiência (PcD PPI).
RINPPIPCDR2	Estudante de escolas públicas brasileiras com renda igual ou inferior a 1,5 (um vírgula cinco) salário-mínimo per capita não autodeclarados pretos, pardos ou indígenas com deficiência (PcD Não PPI)
RSPPIPCDR3	Renda superior a 1,5 (um vírgula cinco) salário-mínimo per capita autodeclarados pretos, pardos, indígenas com deficiência (PcD PPI)
RSNPPIPCDR4	Renda superior a 1,5 (um vírgula cinco) salário-mínimo per capita não autodeclarados pretos, pardos ou indígenas com deficiência (PcD Não PPI)
RIPPIR5	Renda inferior a 1,5 (um vírgula cinco) salário-mínimo per capita autodeclarados pretos, pardos ou indígenas (PPI)
RINPPIR6	Renda inferior a 1,5 (um vírgula cinco) salário-mínimo per capita não autodeclarados pretos, pardos ou indígenas (Não PPI)
RSPPIR7	Renda superior a 1,5 (um vírgula cinco) salário-mínimo per capita autodeclarados pretos, pardos ou indígenas (PPI)
RSNPPIR8	Renda superior a 1,5 (um vírgula cinco) salário-mínimo per capita não autodeclarados pretos, pardos ou indígenas (Não PPI)

Fonte: Sistema de controle de versão do IFSC (2017).

Tendo como base essas novas categorias, a Figura 6 detalha o terceiro cenário de classificação para um curso de 40 vagas.

Figura 6 – Cenário de distribuição versão 3



Fonte: Elaborada pelo autor (2019).

Diante da crescente demanda por alterações nas funções de classificação por cotas, a DTIC optou por refatorar o código de modo que as regras de distribuição não fossem implementadas de forma estática no código fonte, esta versão foi implementada com armazenamento da árvore de cotas em banco de dados, como uma tentativa de diminuir o impacto de novas mudanças por parte dos desenvolvedores.

Nesta versão houve uma redução de linhas de código considerável e a equipe centralizou rotinas responsáveis pelos cálculos em apenas um arquivo `calculo_cotas.php`. Nesse arquivo foram criadas as funções, `obtem_arvore_cotas` e `calcula_cotas` (Código Fonte 5), que fazem a consulta de configurações e percentuais necessários para o cálculo do quadro de vagas.

As novas funções foram importadas em todos os arquivos envolvidos nos processos de classificação de candidatos, e as funções antigas precisaram ser alteradas para buscar as configurações de cotas de maneira dinâmica, como pode ser visto no Código Fonte 6, que agora busca a ordem de prioridade de sobra de vagas da base de dados.

Código Fonte 5 – Nova funções quadro de vagas

```
function obtem_arvore_cotas($total_vagas, $modalidade=null, $vagasZeradas=null){
$vagas_cotas = array();

$resultado = obtem_info_cotas_pela_modalidade($modalidade);

$node = 0;

while ($item = mysql_fetch_array($resultado)){
    $cota = array();

$vagas_pai = get_vagas_pai($vagas_cotas,$item['parent'],$total_vagas);

    if ($node == 0 ){

        if ($item['arredondamento'] == 2){
            //sisu nodos folha, arredonda normal
$vagas_nodo_prioritario = round(($vagas_pai*$item['percentual'])/100);
        }else if ($item['arredondamento'] == 3)
        {
            //para sisu somente roundup sem alterar percentual
            $vagas_nodo_prioritario = ceil(($vagas_pai*$item['percentual'])/100);
        }
            $cota['vagas'] = $vagas_nodo_prioritario;
            $node++;
        }else{
            $cota['vagas'] = $vagas_pai - $vagas_nodo_prioritario;
            $node = 0;
        }
        $cota['parentid'] = $item['parent'];
        $cota['sigla'] = $item['sigla'];
        $cota['id_cota'] = $item['id_cota'];
        $cota['prioridade_transbordo'] = $item['prioridade_transbordo'];
        array_push($vagas_cotas, $cota);
    }

    return $vagas_cotas;
}

function calcula_cotas($total_vagas, $modalidade=null, $arvore_cotas = null,
    $tipoCota = null){
    $vagas = array();

    if($tipoCota != null && $tipoCota == "SEM"){
        return array("SEM" => $total_vagas);
    }

    if(in_array($modalidade, unserialize(C_MODALIDE_PCDPPI))){
        return calcula_cotas_mestrado($total_vagas, $modalidade);
    }

    if($arvore_cotas == null){
        $arvore_cotas = obtem_arvore_cotas($total_vagas,$modalidade);
    }

    foreach($arvore_cotas as $cota){
```

```
        /* as vagas efetivamente estao nas folhas */
        if (is_folha($arvore_cotas, $cota)){
            $vagas[$cota['sigla']] = $cota['vagas'];
        }
    }
    return $vagas;
}
```

Código Fonte 6 – Nova função de prioridade para sobra de vagas

```
function retorna_OrdemdePreenchimentodeVagasNaoOcupadas($situacaodeClassificacao)
{

    $sql = "select sigla FROM cotas WHERE prioridade_transbordo >0 order by
        prioridade_transbordo";
    $res = dbQuery($sql);

    $siglasOrgem = array();

    while($linha = mysql_fetch_array($res))
    {
        array_push($siglasOrgem, $linha['sigla']);
    }

    return $siglasOrgem;

}
```

Embora a inclusão das configurações de cotas no banco de dados tenha ajudado na redução de código fonte estático a ser modificado, ainda surgiram novas demandas de ajustes para determinados tipos de processos nos quais a área demandante precisou da intervenção da equipe de desenvolvimento. Estas situações são exemplificadas na Subseção 2.2.4.

2.2.4 Outras customizações realizadas no algoritmo

O MEC em sua implementação para classificação de candidatos do SISU para o primeiro semestre de 2018, acabou por adotar em seu sistema um entendimento distinto sobre questões de arredondamento de vagas em contra-mão do que era descrito na portaria normativa nº 18/2012/MEC:

> Art. 11 - Sempre que a aplicação dos percentuais para a apuração da reserva de vagas de que trata o art. 10 implicar resultados com decimais, será adotado, em cada etapa do cálculo, o número inteiro imediatamente superior (MEC, 2012).

A área demandante do IFSC em consulta com a Secretaria de Regulação e Supervisão da Educação Superior (SERES) recebeu o retorno de que tiveram que adotar regra diferenciada de arredondamento do exigido em lei, por questões de limitação de vagas, algumas categorias não poderiam ser atendidas na época, obrigando que o sistema de ingresso do IFSC fosse ajustado para a modalidade de processos do tipo SISU, principalmente para chamada de vagas posteriores à classificação gerada pelo SISU.

Nesse contexto, a equipe de desenvolvimento da DTIC precisou fazer correções no algoritmo de classificação para considerar o arredondamento diferenciado. Com isso, inseriu novas configurações em banco de dados para o cálculo condicionando à modalidade do processo, mantendo a compatibilidade com processos que seguem a forma de arredondamento prevista em lei.

A exemplo do desenvolvimento necessário para essa correção de arredondamento, o sistema de controle de versão indica como 8 (oito) arquivos PHP modificados, sendo 69 linhas adicionadas e 44 linhas removidas, além da necessidade de inclusão de 2 (dois) novos campos na tabela de configurações do sistema de cotas.

Essas foram as maiores alterações levantadas no sistema de controle de versões do IFSC, no entanto, pode-se observar na Figura 7, que foram criadas várias demandas na ferramenta *Trello* (utilizada para gerenciamento do desenvolvimento interno), assim como vários chamados de cunho corretivo.

Figura 7 – Demandas sobre o desenvolvimento de cotas

Fonte: Elaborada pelo autor (2019).

Na sequência, no Capítulo 3 estão presentes os principais conceitos encontrados na literatura que serviram de base para fundamentar o desenvolvimento dessa pesquisa.

3 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo são apresentados os principais conceitos necessários para entendimento do presente trabalho. Na Seção 3.1, são apresentados os conceitos sobre DSLs, tais como: diferenças sobre as linguagens convencionais de programação, vantagens e desafios, ferramentas de construção e por fim é apresentada a justificativa de escolha da ferramenta MPS para o desenvolvimento da DSL Cotas.

3.1 LINGUAGENS ESPECÍFICAS DE DOMÍNIO

Para Voelter (2018), a necessidade de abstrações personalizadas sobre um núcleo funcional originam as linguagens de domínio específicos, DSLs. Para Spinellis (2001) DSLs são, por definição, parte de um sistema maior e geralmente implementado para uso específico de domínio.

Mernik, Heering e Sloane (2005), afirmam que as DSLs fornecem construções sob medida para um domínio específico de aplicativo, fornecendo ganhos substanciais em expressividade e facilidade do uso quando comparado com as GPLs, o que corresponde em ganhos de produtividade e redução nos custos de manutenção.

As abordagens de desenvolvimento com base em DSLs possibilitam a especificação de modelos de domínio de forma mais rápida do que as abordagens que utilizam as linguagens de propósito geral. Assim pode haver o aumento da influência dos especialistas de domínio na formulação do modelo, possibilitando que esses apliquem os seus conhecimentos e experiências de domínio diretamente ao problema com apoio da DSL (HOFFMANN et al., 2019).

No que concerne às questões de comunicação, Ghosh (2011) descreve que uma DSL preenche a lacuna semântica entre usuários de negócio e desenvolvedores, incentivando uma melhor colaboração por meio de vocabulário compartilhado.

Segundo Mernik, Heering e Sloane (2005), são exemplos de linguagens de domínio específico: HTML, Latex, SQL, Excel, etc. As linguagens possuem domínios de aplicação diferenciados (Figura 8), e a sua aplicação pode reduzir a expertise necessária de programação para seus usuários .

As DSLs podem ser classificadas de acordo com a sua implementação, se construídas para serem incorporadas em uma GPL elas podem ser consideradas como DSLs internas, enquanto as externas são independentes da infraestrutura de uma linguagem de programação existente (VOELTER et al., 2013).

Figura 8 – Exemplos de DSL amplamente utilizadas

DSL	Application Domain
BNF	Syntax specification
Excel	Spreadsheets
HTML	Hypertext web pages
LaTeX	Typesetting
Make	Software building
MATLAB	Technical computing
SQL	Database queries
VHDL	Hardware design

Fonte: Mernik, Heering e Sloane (2005).

Fowler (2005) define DSLs externas e internas como:

> DSL externa é aquela que é escrita em uma linguagem separada da linguagem principal da aplicação, *Unix little languages* e arquivos de configuração de Extensible Markup Language (XML) são bons exemplos desse estilo de linguagem. Enquanto as DSLs internas são limitadas pela sintaxe e estrutura da linguagem base em que são criadas, recursos de linguagem como *closures*, *macros* são exemplos de construção de linguagens internas (FOWLER, 2005, s/p, tradução nossa).

Nesse contexto, o presente trabalho objetiva a criação de uma DSL externa, com sua própria sintaxe e semântica, de modo que seja possível especificar regras do sistema de cotas da rede de ensino pública federal, de maneira independente à linguagem de programação alvo do sistema.

3.1.1 Diferenças sobre linguagens convencionais

Segundo Voelter et al. (2013), o propósito das DSLs é de atender a um domínio específico, elas são construídas para resolver uma classe específica de problemas. De outro lado, as GPLs são voltadas aos desenvolvedores para resolver qualquer tipo de problema computável:

> As Linguagens de Programação de Uso Geral (GPLs) são um meio para programadores instruírem computadores.Todas podem ser usadas para implementar qualquer coisa computável com uma máquina de Turing. Isso também significa que qualquer coisa expressável com uma linguagem de programação completa de Turing também pode ser expressa em qualquer outra linguagem de programação completa de Turing. Nesse sentido, todas as linguagens de programação são intercambiáveis. (VOELTER et al., 2013, p.27, tradução nossa)

Para Ghosh (2011), projetar uma DSL não é uma tarefa tão assustadora quanto projetar uma linguagem de programação de uso geral. Pois tem foco limitado e é restrita apenas ao domínio que está sendo modelado.

Enquanto GPLs são flexíveis, as DSLs sacrificam a flexibilidade em benefício da produtividade e da concisão de programas relevantes em um domínio específico. As GPLs são utilizadas em domínios maiores e complexos, do outro lado são trabalhados problemas menores e bem

definidos (VOELTER et al., 2013). Algumas dessas diferenças podem ser observadas na Figura 9.

Figura 9 – DSL vs GPL

	GPLs	DSLs
Domain	large and complex	smaller and well-defined
Language size	large	small
Turing completeness	always	often not
User-defined abstractions	sophisticated	limited
Execution	via intermediate GPL	native
Lifespan	years to decades	months to years (driven by context)
Designed by	guru or committee	a few engineers and domain experts
User community	large, anonymous and widespread	small, accessible and local
Evolution	slow, often standardized	fast-paced
Deprecation/incompatible changes	almost impossible	feasible

Fonte: Voelter et al. (2013).

Existem situações em que, usuários sem habilidades de programação, precisam injetar regras no sistema, ou definir condições de modo que possam atender uma funcionalidade ou requisito necessário, porém, não é viável ensiná-los conceitos básicos de linguagens de programação. A configuração usando uma linguagem específica de domínio pode ser uma alternativa para resolver estas situações (NOVÁK, 2010).

No entanto, como em qualquer desenvolvimento de modelagem de software existem benefícios e desafios a serem levados em conta para a tomada de decisão sobre adoção DSLs no desenvolvimento de software. Na Subseção 3.1.2, são pontuados alguns desses desafios e vantagens encontrados na literatura.

3.1.2 Vantagens e desafios no desenvolvimento

Para Voelter et al. (2013), o uso de DSLs no desenvolvimento de soluções traz muitos benefícios, tais como:

a) Produtividade: no sentido de que pode-se substituir muito código fonte de GPLs, por algumas poucas linhas de código em DSLs;

b) Qualidade: quando se pensa na criação de um produto com menos *bugs*, resultado da redução do grau de liberdade (desnecessário) para os programadores, prevenindo a duplicação de código (se a DSL for projetada da maneira correta);

c) Validação e Verificação: pois as DSLs capturam suas respectivas preocupações de modo que não possuam tantos detalhes de implementação, os programas criados são semanticamente mais ricos que as GPLs, permitindo análises e a elaboração de mensagens de erro mais significativas sobre os conceitos do domínio;

d) Ferramenta de pensamento e comunicação: quando se tem uma maneira de expressar preocupações de domínio, em um idioma que está alinhado com o domínio, o pensamento se torna mais claro, porque o código que você escreve não está cheio de detalhes de implementação. Isso pode estreitar as diferenças de entendimento do domínio, que ocorrem em equipes com várias pessoas trabalhando na solução de um problema em comum;

e) Envolvimento de especialistas de domínio: quando os especialistas de negócio (não desenvolvedores) conseguem expressar mais facilmente suas ideias, o que pode levar a uma melhor integração entre eles e os desenvolvedores das linguagens;

f) Ferramentas produtivas: ao contrário de bibliotecas, *frameworks* e DSLs internas, as DSLs externas podem ser fornecidas com uma Integrated Development Environment (IDE) preparada para o seu idioma, o que pode resultar em uma experiência de usuário muito aprimorada;

g) Redução no *Overhead*: se tratando de DSL de geração de código, o gerador pode reduzir abstrações de domínio mais complexas e gerar código mais eficiente, assim como os compiladores das linguagens convencionais otimizam o *bytecode* gerado.

Mernik, Heering e Sloane (2005), Ghosh (2011) e Voelter et al. (2013) apresentam em seus textos, alguns desafios e desvantagens que precisam ser consideradas antes da elaboração de uma DSL:

a) A modelagem de DSL não é trivial, no sentido que exige muita experiência do domínio e expertise em desenvolvimento de linguagens, poucas pessoas possuem ambos;

b) Dependendo do tamanho da comunidade de usuários da DSL, o desenvolvimento de treinamentos, documentações, suporte e manutenção, pode se tornar um sério problema;

c) O custo inicial de desenvolvimento pode ser alto, apesar de poder ser compensado pelo tempo economizado com o aumento da produtividade nos estágios posteriores do ciclo de desenvolvimento;

d) A criação descontrolada de linguagens pode resultar em um design inchado, Voelter et al. (2013) apresenta o termo *DSL Hell*, no qual ao invés de pesquisar por uma DSL existente para determinado domínio e aprendê-la, o desenvolvedor acaba por fazer a construção de uma nova linguagem, que pode possivelmente sobrepor implementações já cobertas, mas ainda assim serem incompatíveis entre si;

e) A construção de DSLs requer mudança no pensamento cultural da organização, pois o método de desenvolvimento é bastante diferente dos métodos tradicionais da engenharia de software, e podem implicar em mudanças significantes em como a equipe de desenvolvedores e usuários trabalham.

No sentido de facilitar o design e a criação das linguagens específicas de domínio, atualmente são encontradas várias IDEs, algumas delas são listadas na Subseção 3.1.3. Elas têm como objetivo, agregar produtividade para os designers e desenvolvedores, assim como outras vantagens, compensando assim, os eventuais desafios descritos.

3.1.3 Ferramentas de construção

Para Voelter (2014), as ferramentas desempenham um papel importante no desenvolvimento de software, à medida em que a complexidade do software aumenta, da mesma forma há um aumento na importância em se construir e utilizar ferramentas de apoio.

Fowler (2005) apresenta o termo *Language Workbench*, como o ferramental necessário para construir um conjunto de linguagens específicas de domínio. Para ele, essas ferramentas possuem 3 (três) principais definições em comum para criação de DSLs:

a) Definem a sintaxe abstrata, que é o modelo da representação abstrata da linguagem;

b) Definem um editor, que permita manipulação da representação abstrata;

c) Definem um gerador, que descreve como deve ser traduzida a representação abstrata em uma representação executável.

O processo de desenvolvimento de linguagens pode ser facilitado por meio do uso de ferramentas ou de sistemas de criação de linguagens. Apesar de existirem vários, todo esse ferramental possui como propósito básico a descrição de linguagens, podendo variar os interpretadores, analisadores, verificadores de consistência e os ambientes de desenvolvimento (IDEs) disponibilizadas aos designers (MERNIK; HEERING; SLOANE, 2005).

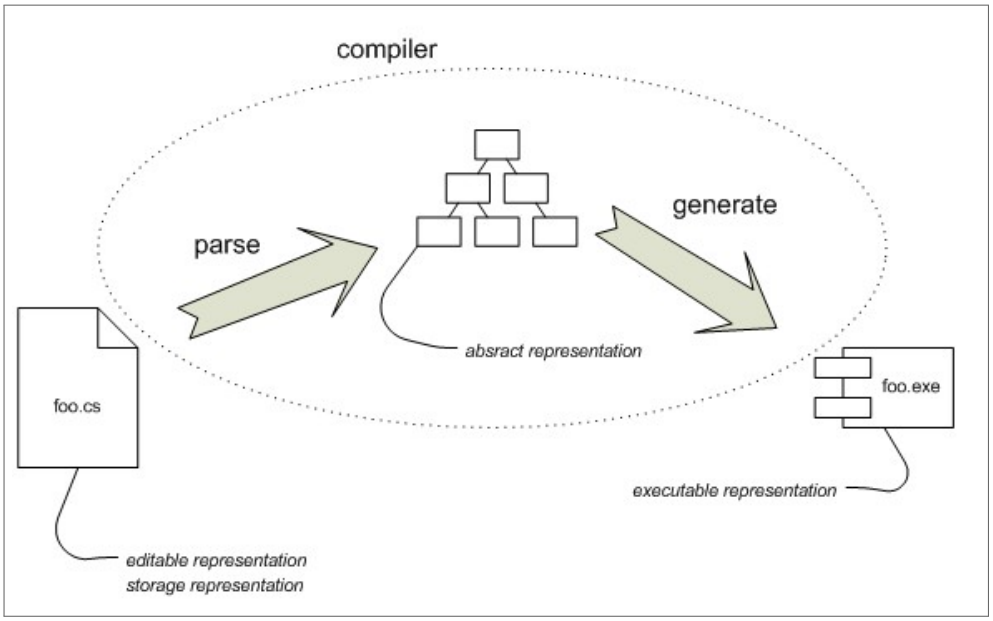
> A maioria dos ambientes de programação são baseados em edição livre de texto, na qual o desenvolvedor manipula o texto de modo a formar palavras e frases. Nesse sentido, é necessário um *parser* para verificar se o texto do programa (sintaxe concreta) possui a sintaxe correta e assim como criar uma Árvore de Sintaxe Abstrata (AST) para popular a estrutura de dados a partir da informação extraída da fonte de texto (VOELTER et al., 2013, p.179,tradução nossa).

Nos tradicionais compiladores e IDEs, os *parsers* precisam ser implementados manualmente, o que requer grande esforço de desenvolvimento. Para uma realidade de construção de uma linguagem tradicional, GPL, essa abordagem faz sentido, porém no contexto de DSLs, as ferramentas permitem gerar um *parser* a partir de uma gramática, não sendo necessário que o programador crie um *parser* customizado.

Voelter et al. (2013) categoriza as ferramentas modernas para construção de DSLs como *baseadas em parsing* e ferramentas de *abordagem projecional*. No primeiro caso, a partir da gramática é gerado um *parser*, e no segundo, a AST é construída diretamente por ações no editor, na qual é modificada diretamente enquanto o usuário edita o programa (abordagem bem conhecida para editores gráficos em geral, e o padrão MVC).

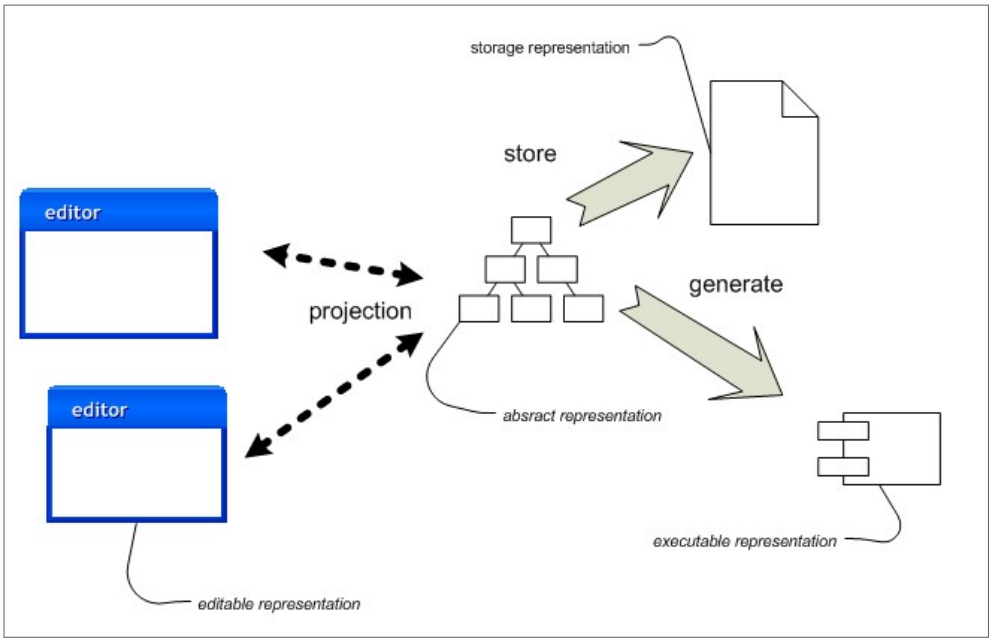
Para Fowler (2008), na abordagem de *parsing*, o compilador utiliza o código fonte para gerar uma representação executável do programa, transformando todo o texto em uma representação abstrata que é mantida apenas durante a compilação. (Figura 10).

Figura 10 – Abordagem de *parsing*



Fonte: Fowler (2008).

Figura 11 – Abordagem projecional



Fonte: Fowler (2008).

No caso da abordagem projecional, utiliza-se editores gráficos para manipular diretamente as definições do sistema, ou seja, edita-se a representação abstrata, de modo que o programador possa modificar as definições mantidas no modelo. Nesse caso, o executável é produzido por uma série de projeções e transformações a partir desta representação abstrata (Figura 11).

Na Subseção 3.1.4 são apresentados alguns exemplos de ferramentas de criação de DSLs.

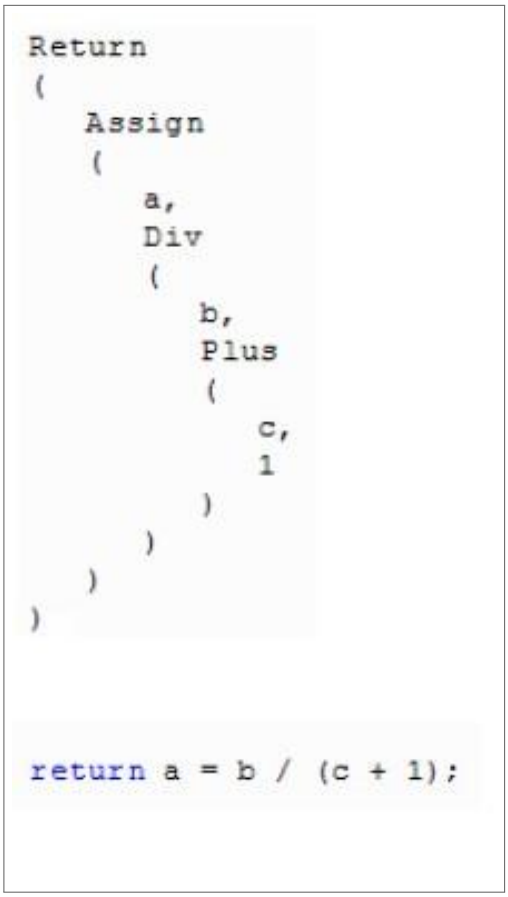
3.1.4 Exemplos de ferramentas

Fowler (2005), apresenta em seu texto, a *Intentional Software* como a precursora das ferramentas de criação de linguagens, ela foi desenvolvida por *Charles Simonyi* no setor de pesquisa *Microsoft Research*.

Essa ferramenta permite a criação e edição de código de domínio, o qual é criado por meio da definição de um esquema de domínio pelo *Domain Expert* com apoio dos desenvolvedores, para então ser construído um gerador de código que resulta na aplicação final (SIMONYI; CHRISTERSON; CLIFFORD, 2006).

A ferramenta utiliza uma representação de domínio em formato de árvore de intenções, que é apresentada ao usuário em diferentes formas de visualização e edição, a Figura 12 mostra a árvore de intenção para uma instrução matemática simples.

Figura 12 – Árvore de intenção no *Intentional Software*

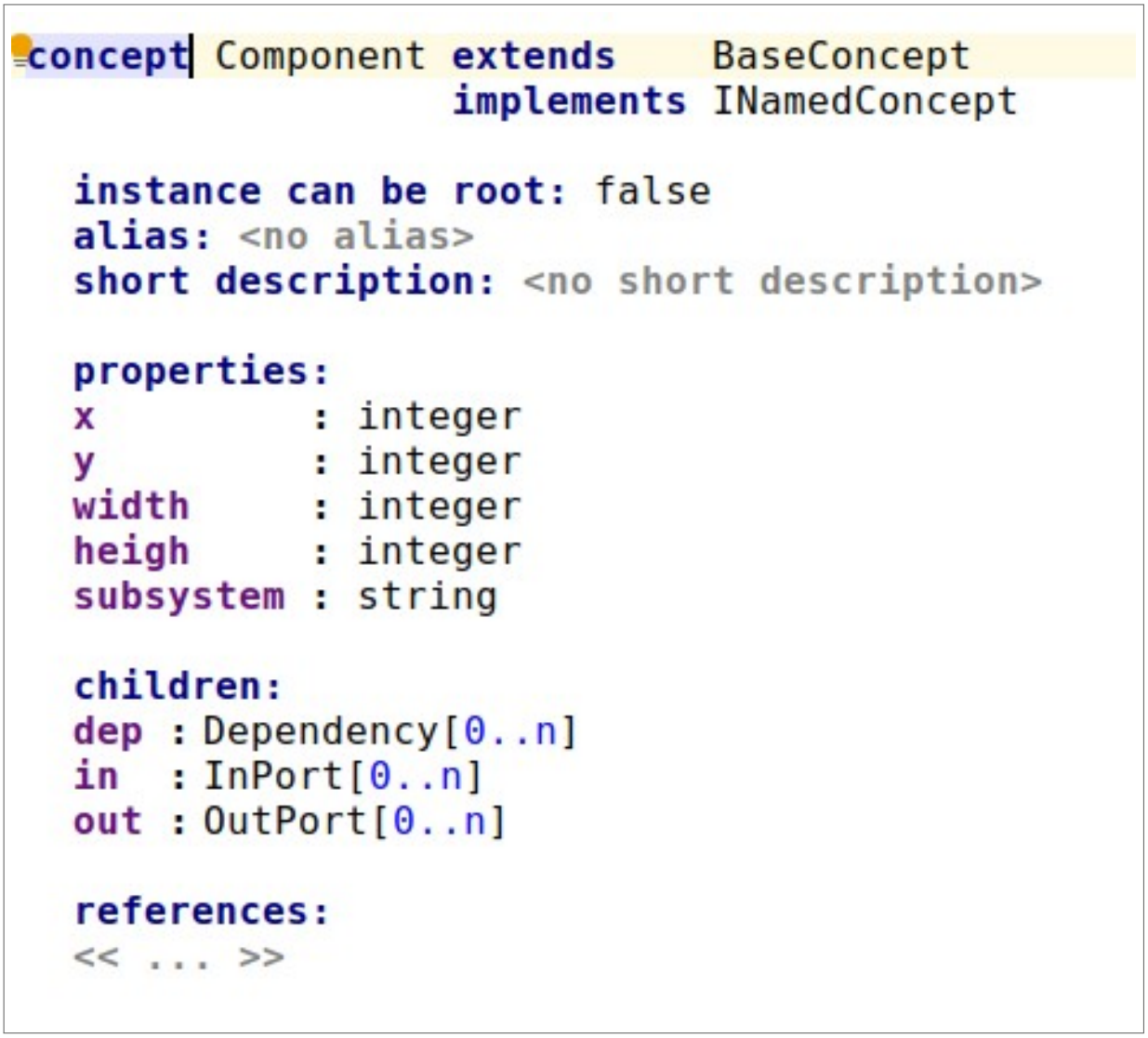


Fonte: Simonyi, Christerson e Clifford (2006).

Outra ferramenta de abordagem projecional é a MPS, de código aberto sob a licença Apache 2.0 e é desenvolvida pela *JetBrains*. Ela se baseia em definição de linguagens por meio de representação de texto estruturado, e possui uma série de recursos sofisticados de edição e ferramentas de navegação.

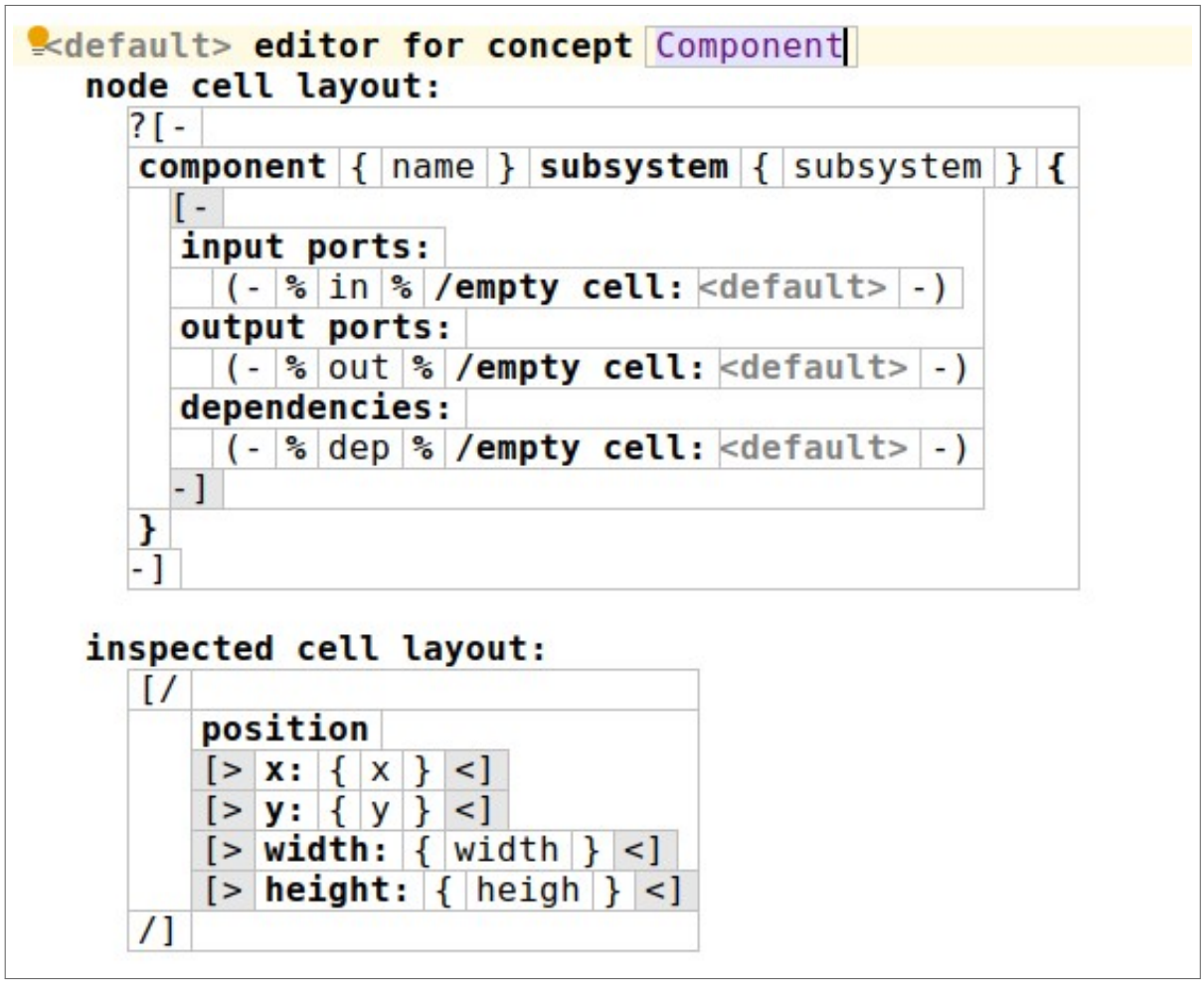
Segundo Voelter et al. (2013), a MPS suporta notações mistas (textuais, simbólicas, tabulares, gráficas) e uma ampla variedade de recursos de composição de idiomas. Ela define a linguagem por meio de *Concepts* que são elementos para criação da sintaxe abstrata (Figura 13), e esses conceitos são atrelados a um editor, o qual define as regras de projeção por meio de uma lista de células, que juntas definem a estrutura desejada para a sintaxe do conceito (Figura 14).

Figura 13 – MPS definição de Conceitos



Fonte: JetBrains (2018).

Figura 14 – MPS editor - sintaxe abstrata



Fonte: JetBrains (2018).

Como alternativa às ferramentas projecionais, existem ferramentas que geram o *parser* a partir da gramática, nesse caso, as regras da linguagem são expressadas em notação baseada em Backus Naur Form (BNF). As ferramentas Xtext e Spoofax são casos de ambientes nos quais é possível construir DSLs por meio desse tipo de abordagem, a seguir são listados alguns exemplos nessas ferramentas.

Eysholdt e Behrens (2010), descreve a ferramenta Xtext, como um *framework* que permite o rápido desenvolvimento de ferramental de suporte às linguagens textuais, podendo atender desde linguagens menores como DSLs, até linguagens completas GPLs. Ela utiliza o core do Eclipse Modeling Framework (EMF) para criar a AST a partir de uma especificação de gramática.

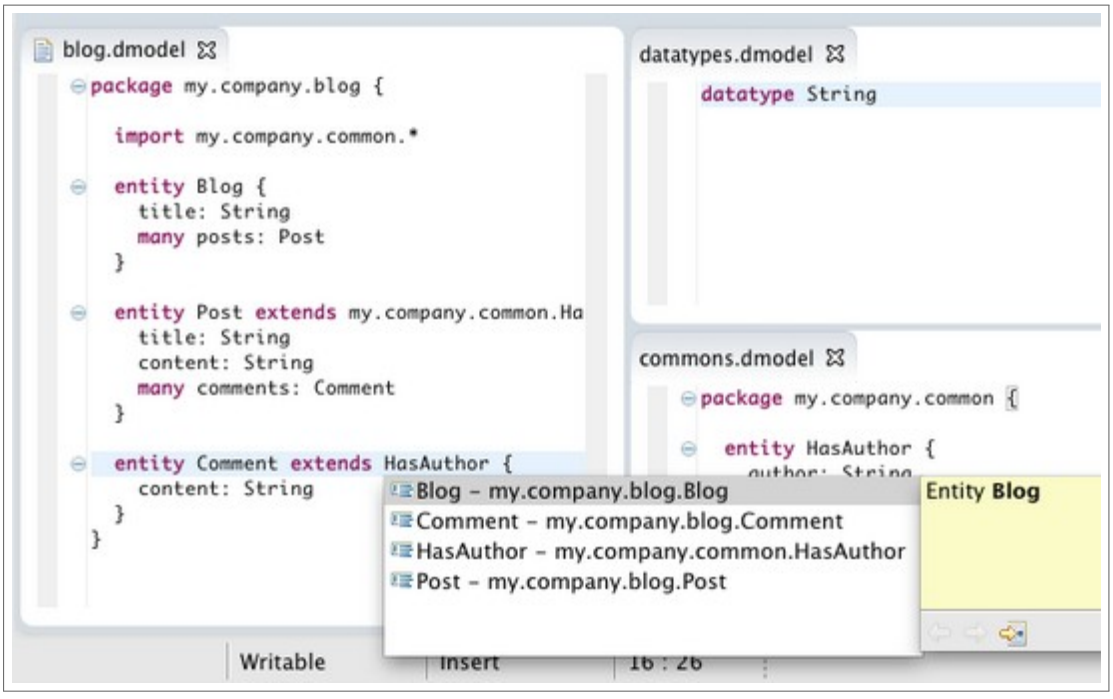
Na Figura 15, é possível verificar a definição da gramática de uma linguagem específica para modelagem de entidades, de forma similar ao que existe em alguns *frameworks* de programação, como Rails e Grails. O resultado é apresentado na Figura 16, na qual observa-se a utilização de recursos do Xtext no momento da codificação de um novo programa nessa linguagem.

Figura 15 – Exemplo de definição de gramática Xtext

```
grammar org.example.domainmodel.Domainmodel with
                                        org.eclipse.xtext.common.Terminals

generate domainmodel "http://www.example.org/domainmodel/Domainmodel"

Domainmodel:
    (elements+=AbstractElement)*;


QualifiedName:
    ID ('.' ID)*;

Type:
    DataType | Entity;

DataType:
    'datatype' name=ID;

Entity:
    'entity' name=ID ('extends' superType=[Entity|QualifiedName])? '{'
        (features+=Feature)*
    '}';

Feature:
    (many?='many')? name=ID ':' type=[Type|QualifiedName];
```

Fonte: XText (2019).

Figura 16 – Exemplo de uso da linguagem de entidades

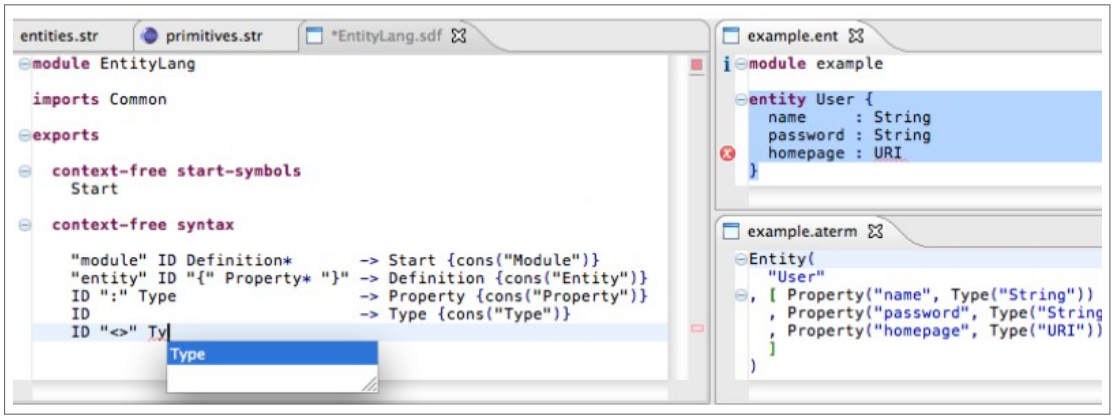


Fonte: XText (2019).

Por fim, a ferramenta *Spoofax* utiliza o formato Syntax Definition Formalism (SDF) para descrição da sintaxe da linguagem, sendo um ambiente integrado de especificação de linguagens, com suporte e *plugins* para a IDE Eclipse.

Um exemplo de definição de gramática em formato SDF pode ser observado na Figura 17. No canto esquerdo da imagem são definidas as regras da sintaxe, para uma linguagem de entidades similar a que foi utilizada no exemplo do Xtext.

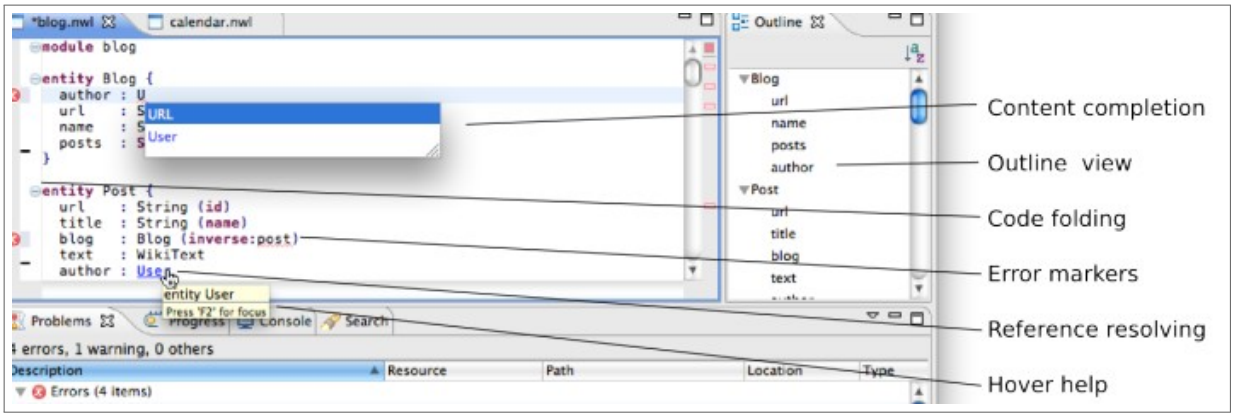
Figura 17 – Definição de gramática no Spoofax



Fonte: Kats e Visser (2010).

Segundo Kats e Visser (2010), os seus editores permitem verificar e destacar erros por linha, validar tipos de dados da linguagem, resolução de referências, recursos de ajuda conforme contexto e até completar ou sugerir opções de código para o usuário da linguagem (Figura 18).

Figura 18 – Recursos do editor Spoofax



Fonte: Kats e Visser (2010).

A fim de criar a linguagem de domínio desenvolvida pelo presente trabalho, foi necessário definir entre uma dessas ferramentas. Na Subseção 3.1.5 é apresentada a justificativa da escolha do MPS como ferramenta para implementação da linguagem de especificação de regras de classificação pelo sistema de cotas.

3.1.5 Justificativa de escolha da ferramenta MPS

Essa Subseção tem por objetivo detalhar os fatores considerados para a escolha da ferramenta MPS como meio de implementação da linguagem de domínio desenvolvida na presente pesquisa.

Anteriormente à escolha do MPS, o autor também considerou as ferramentas Xtext e Spoofax, nas quais foram realizados testes seguindo os guias: *Xtext 15 Minutes Tutorial* e *Spoofax Tutorial Questionnaire Language*. Após a execução dos tutoriais houve maior facilidade na utilização da ferramenta MPS, pois além do tutorial introdutório disponibilizou em seu portal um curso online gratuito complementar, o qual contribuiu para melhor entendimento dos recursos utilizados para desenvolvimento da pesquisa, tais como: múltiplas formas de projeções, textuais, tabulares e recursos de formulários.

Ademais, as ferramentas Xtext e Spoofax dependem de definição de uma gramática formal, enquanto o MPS permite que as definições da linguagem sejam associadas diretamente ao modelo. Segundo Volter (2011), isso traz alguns benefícios durante o desenvolvimento de linguagens:

a) Nas abordagens projecionais, não é necessário a definição de uma gramática ou *parser*, a edição modifica diretamente a estrutura da linguagem;

b) Notações podem ser combinadas e misturadas, como por exemplo, a combinação de editores em formato gráfico, textual ou simbólico, formalmente o suficiente para permitir o seu processamento ou a sua tradução automática e com bom suporte da IDE;

c) A modelagem é definida independentemente da sua notação concreta, sendo possível representar o mesmo modelo de diferentes maneiras, simplesmente fornecendo várias projeções.

Fowler (2005), descreve a empresa *JetBrains* como uma empresa com muita experiência no desenvolvimento de *workbenches* para programação, incluindo a credibilidade adquirida com o sucesso da ferramenta *IntelliJ* para desenvolvimento de Java. Nesse sentido, o MPS agrega várias capacidades que vieram do histórico de recursos disponíveis no *IntelliJ*.

Além de ser uma ferramenta robusta, desenvolvida por uma equipe experiente, o principal fator considerado para escolha da ferramenta MPS, foi a facilidade de encontrar documentação e vídeos exemplificativos sobre a IDE.

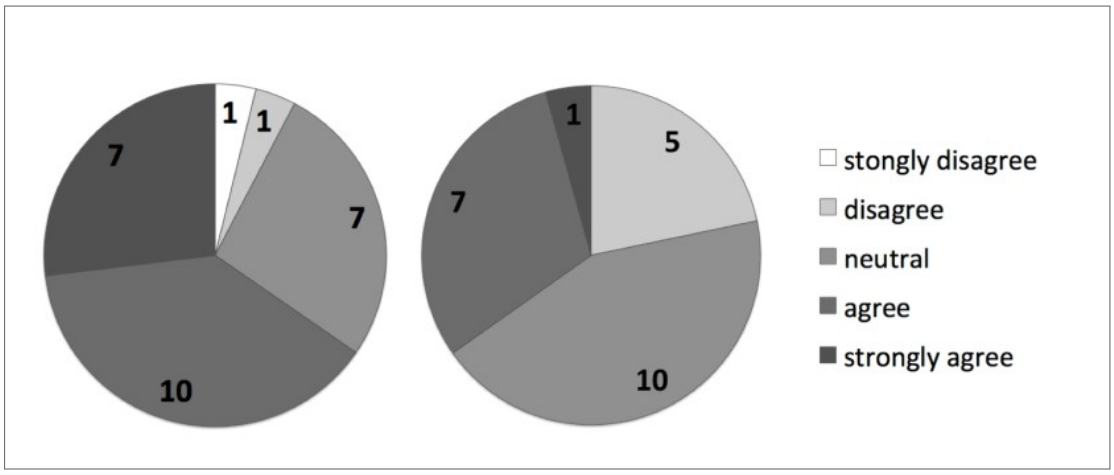
A JetBrains disponibiliza em seu site um guia rápido de 10 passos (*Fast Track to MPS - Ten steps to mastery*) para início do desenvolvimento de DSLs com o MPS, incluindo: *overview* sobre os principais recursos, procedimentos de instalação, tutoriais de implementação de alguns exemplos de DSLs, referência para documentação oficial e até um treinamento online introdutório, disponível de forma gratuita em ferramenta de ensino à distância (*JetBrains MPS Elementary Course*).

Tendo em vista que, a presente pesquisa procura reduzir a demanda para os desenvolvedores com relação às mudanças nas regras do sistema de cotas, é importante que a ferramenta ajude em questões de produtividade e possua boa usabilidade. Uma vez que as alterações podem surgir, seja em função de mudanças nos documentos de lei, ou até por questões de entendimento sobre determinado requisito de negócio - exemplos dessas situações foram apresentados anteriormente no Capítulo 2.

Nesse sentido, o texto de Voelter (2014), apresenta um levantamento sobre a usabilidade do MPS, em que 20 usuários do MPS foram questionados sobre: "Eu consigo trabalhar produtivamente com o MPS?" e "Foi fácil se acostumar com a programação no MPS?". Esta pesquisa, segundo ele, mostra que os primeiros resultados foram positivos.

O resultado dessa pesquisa pode ser visto na Figura 19. São utilizados 5 (cinco) níveis de escala de concordância às 2 (duas) perguntas. O gráfico da esquerda apresenta que a maioria concorda totalmente ou apenas concorda sobre a pergunta de trabalho com produtividade, enquanto no gráfico da direita, sobre a facilidade de se acostumar com a IDE, a concordância é geral (VOELTER, 2014).

Figura 19 – Pesquisa sobre usabilidade do MPS



Fonte: Voelter (2014).

Esses pontos favorecem o desenvolvimento e reutilização de linguagens, pois no MPS é possível criar linguagens novas estendendo definições de outras já existentes, sendo possível combinar conceitos que deveriam ser tratados em diferentes pontos de vista, segregando preocupações da solução, de acordo com as necessidades e interesses de cada domínio (VOLTER, 2011).

Do ponto de vista do autor da presente pesquisa, o MPS se mostra uma ferramenta moderna e madura, sua base de exemplos possui mais de 37 projetos que podem servir para facilitar o aprendizado e o desenvolvimento de linguagens. Ademais, é gratuita, com vasta documentação, e por se tratar de uma ferramenta projecional não há a necessidade de manter uma gramática formal ou de desenvolvimento de *parsers*.

Com foco nessa ferramenta e nos conceitos sobre linguagens específicas de domínio apresentados, nos próximos capítulos descreve-se a linguagem desenvolvida nessa pesquisa, assim como algumas vantagens que poderão contribuir na geração do código fonte que faz a classificação de candidatos ao sistema de cotas.

Na sequência, no Capítulo 4 será descrito o modelo e os elementos da DSL Cotas implementados por meio da ferramenta MPS.

4 A DSL COTAS

Neste Capítulo descreve-se a linguagem desenvolvida na ferramenta MPS que objetiva facilitar a definição das regras de distribuição de cotas e, dessa forma, permitir que os usuários do sistema de ingresso do IFSC possam alterar essas definições de forma independente à implementação do respectivo algoritmo de classificação.

Como foi descrito no Capítulo 2, desde a promulgação da lei de cotas em 2012, o sistema de ingresso passou por, pelo menos, 3 (três) versões diferentes de distribuição de vagas, ocasionando em várias refatorações de código desde então.

Desse modo, a presente pesquisa tem como foco: a definição de uma linguagem para descrever todos os 3 (três) cenários passados, com apoio de recursos de linguagens existentes no MPS tais como: validação de estrutura e de tipos de dados aceitos, preenchimento automático de código de acordo com o contexto, validação de inconsistências, recursos de dicas ao usuário da linguagem e geração de código fonte responsável pela classificação e aprovação de candidatos.

Com o apoio desses recursos, o usuário especialista no sistema de cotas, poderá fazer o uso controlado da especificação de regras, assim como, no caso de surgimento de um novo cenário de distribuição de vagas. Esse conjunto de regras alteradas podem ser transformadas em código fonte, possibilitando a sua importação no sistema de ingresso do IFSC, diminuindo a necessidade de refatoração em seu código.

Nesse sentido, as Seções 4.1 e 4.2 abordam as características de implementação da DSL a ser utilizada por especialistas de negócio, para definição da árvore de distribuição de cotas, tendo como base as especificações de lei. Enquanto na Seção 4.3, apresenta-se o detalhamento do desenvolvimento da API DSL Cotas responsável por classificar e aprovar os candidatos tendo como base essa linguagem.

4.1 A MODELAGEM DA DSL COTAS

Tendo como base a análise realizada no Capítulo 2, sobre as versões do sistema de cotas detalhadas nas Subseções 2.2.1, 2.2.2 e 2.2.3, podem ser pontuadas algumas características em comum identificadas entre cada uma dessas versões:

a) A divisão das vagas entre as diferentes categorias de cotas se deu em formato hierárquico, iniciando no total de vagas o qual foi sendo subdividido em percentuais reservados para suas subcategorias (ex: Estudantes da escola pública, Estudantes com renda baixa, Estudantes PCD, etc);

b) Esses percentuais são de conhecimento dos usuários especialistas no sistema de cotas, e podem variar de acordo com a documentação da legislação vigente, ou outras definições

que variam conforme a Unidade Federada que oferta o curso;

c) Em muitos casos, não é definido um valor de percentual fixo, de modo que uma categoria recebe o valor calculado restante de vagas da categoria pai. Por exemplo, o percentual de cotistas PCD é aplicado, e o que resta das vagas vai para candidatos da mesma categoria que não são PCD;

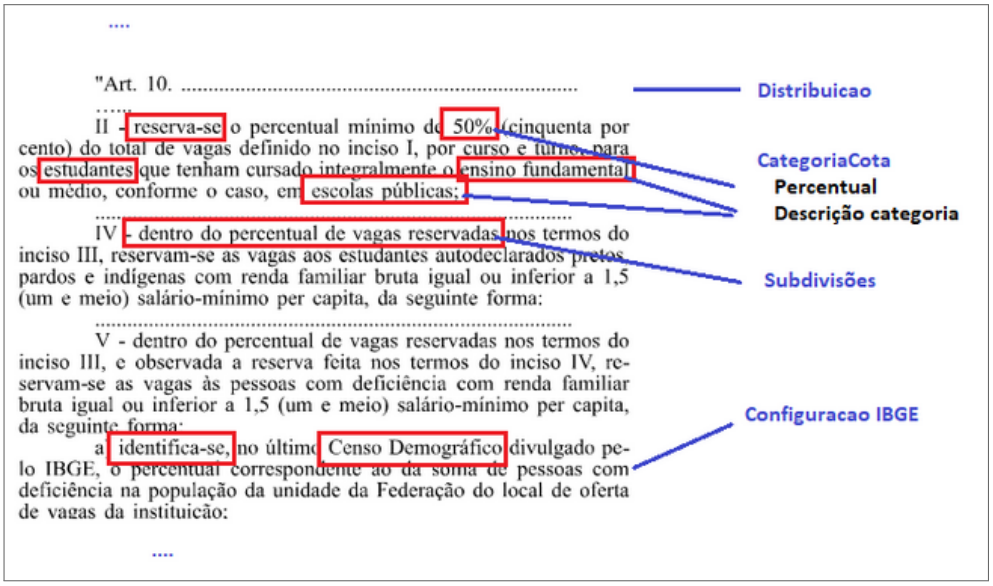
d) Todas as versões consideravam a ordem de prioridade entre as diferentes categorias de cotas passíveis de inscrição;

e) Questões de arredondamento das vagas (para cima ou para baixo), devem ser consideradas e podem variar de acordo com a versão de lei implementada.

Segundo Voelter et al. (2013), a modelagem de DSLs costuma ser realizada com abordagens *bottom-up* ou *top-down*, a primeira abordagem aplica-se para domínios em termos de software, enquanto na segunda, o modelo do domínio é aplicado ao conhecimento sobre o mundo real, fora do domínio de software. Nesse contexto, a modelagem da DSL Cotas utilizou a abordagem top-down, uma vez que seu design procura fornecer suporte para os usuários especialistas definirem os parâmetros da legislação de forma controlada, possibilitando a geração do código de processamento de classificação de candidatos sem que o usuário precise utilizar comandos complexos para a implementação dos respectivos algoritmos.

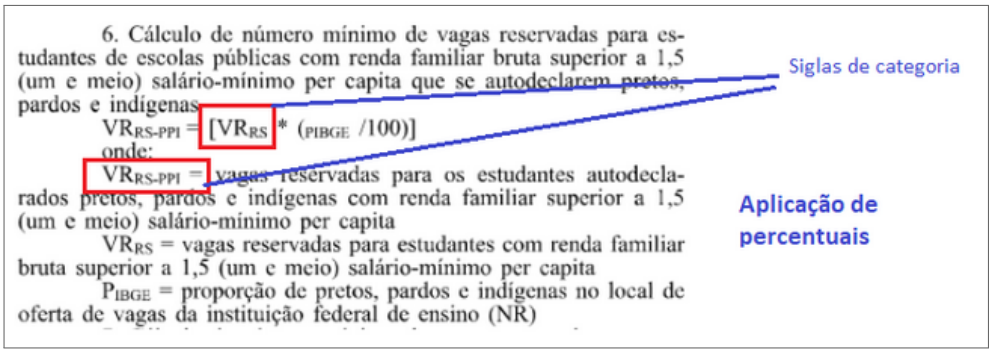
Para tanto, o modelo proposto foi baseado em artigos presentes na legislação que definem em formato textual, respectivamente: a distribuição de vagas, a forma de aplicação dos percentuais e a ordem de prioridade, como pode ser observado nas Figuras 20, 21 e 22.

Figura 20 – Elementos do modelo na portaria nº9/2017/MEC



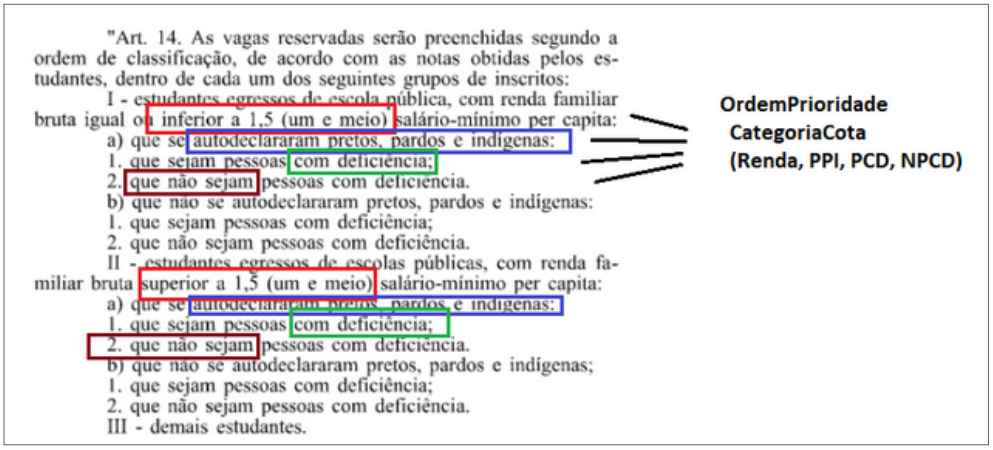
Fonte: Elaborada pelo autor (2020).

Figura 21 – Definições de fórmulas de reserva de vagas



Fonte: Elaborada pelo autor (2020).

Figura 22 – Ordem de prioridade da legislação



Fonte: Elaborada pelo autor (2020).

Essas Figuras exemplificam as propriedades e relacionamentos entre as categorias de cotas consideradas para a construção da linguagem proposta. Observa-se, por exemplo, que as informações de distribuição, das categorias de cotas, das subdivisões e das configurações (destacadas em "vermelho") na Figura 20 incorporam os elementos do modelo da DSL Cotas (destaques em "azul") e suas respectivas propriedades (destacadas em "preto"). Na Figura 21 é possível perceber no trecho de lei extraído a definição de siglas das categorias e exemplos de aplicação dos percentuais. Por fim, na Figura 22 apresentam-se as diferentes categorias que definem a ordem de prioridade, em formato sequencial definido pela legislação, em que as categorias de Renda estão destacadas em "vermelho", as categorias de PPI em "azul" e as PCD em "verde". Ao lado direito consta um exemplo de ordem no elemento do modelo, extraído do artigo de lei (destaque na cor "preta", nomeado `OrdemPrioridade`).

Assim, a DSL desenvolvida consistiu em criar um modelo que permita a configuração de todos os itens listados, como: identificador da versão de lei, lista de variáveis para configuração de percentuais, formas de aplicação de arredondamento, uma macro para aplicar a função de resto de vagas de uma categoria, campo descritivo para os diferentes tipos de categorias, estrutura para divisão em subcategorias e lista de ordem de prioridade para a sobra de vagas.

Na Seção 4.2, são apresentados os componentes desenvolvidos no MPS que são responsáveis pela construção da DSL Cotas.

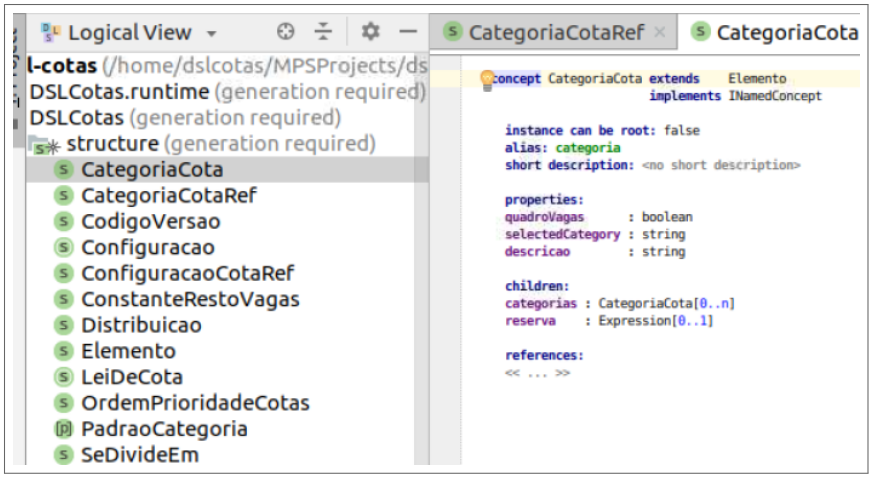
4.2 COMPONENTES DA DSL COTAS NO MPS

Nessa Seção são descritos os elementos de modelagem criados na ferramenta MPS tais como: os componentes de estrutura (`structure concepts`), a sintaxe dos editores projecionais (`editors`), as restrições de escopo (`constraints`), os comportamentos de conceitos (`behaviors`), o sistema de tipos (`typesystem`) e, por fim, os elementos de geração de texto (`textGen`).

4.2.1 *Componentes de estrutura*

Concepts ou conceitos no MPS servem para definir a estrutura base da linguagem, cada conceito pode conter propriedades, outros conceitos filhos (`childrens`) e referências para outros conceitos. Eles podem herdar ou implementar as características de outros conceitos. A Figura 23 apresenta a lista de conceitos criados para a DSL Cotas.

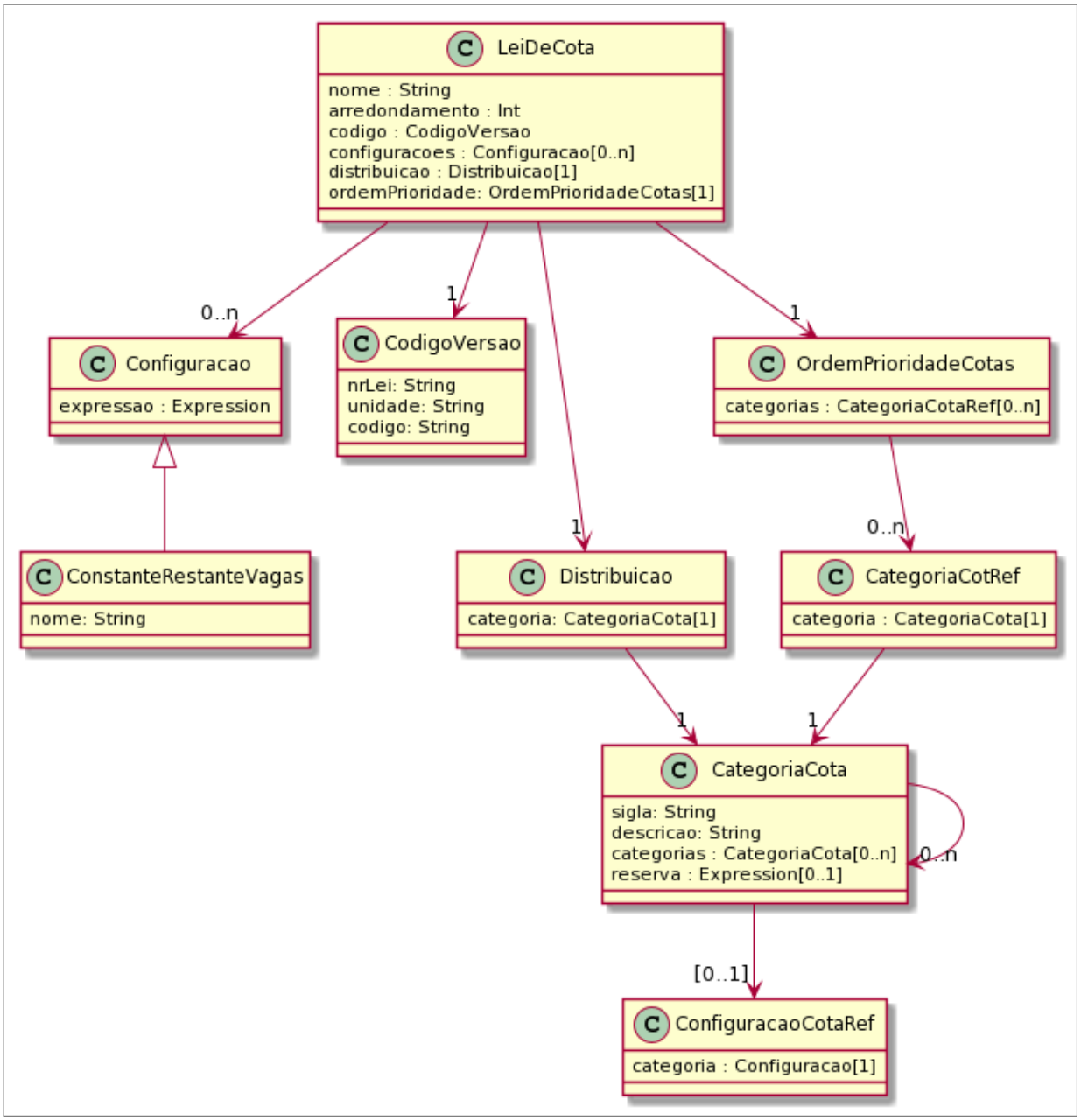
Figura 23 – Lista de Conceitos de Estrutura MPS



Fonte: Elaborada pelo autor (2020).

Esses conceitos definem a estrutura hierárquica da AST de modo análogo ao modelo orientado a objetos, portanto, a Figura 24 mostra a representação da modelagem dos conceitos em formato de diagrama de classes da Unified Modeling Language (UML).

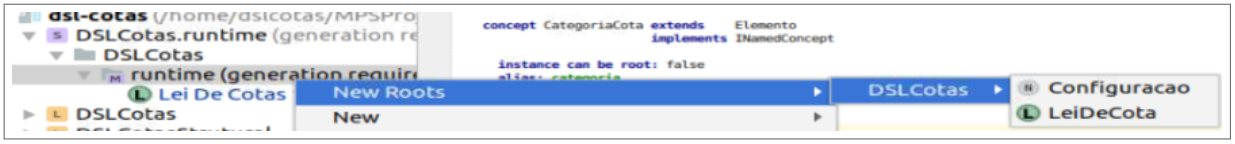
Figura 24 – Modelo de Conceitos no MPS



Fonte: Elaborada pelo autor (2020).

O conceito LeiDeCota é o elemento raiz que pode ser instanciado no MPS a fim de criar uma representação abstrata de uma nova lei de cotas na DSL. No MPS, os conceitos raiz podem ser criados em módulos Solutions, os quais utilizam uma ou mais linguagens e são os responsáveis por conter o código fonte do usuário final (Figura 25).

Figura 25 – Criação de elementos raiz em Solutions no MPS

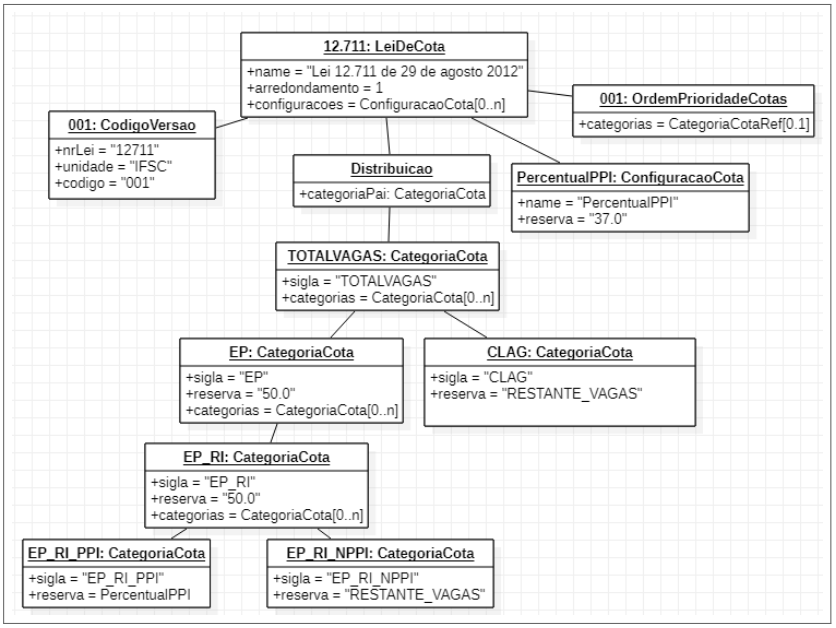


Fonte: Elaborada pelo autor (2020).

Por sua vez, uma LeiDeCota está associada aos elementos CodigoVersao, Lista de Configuracao, Distribuicao e OrdemPrioridadeCotas. O conceito CodigoVersao armazena os dados que identificam o número de lei, a sigla da instituição de ensino e um número sequencial para diferenciar as versões de implementação. Opcionalmente, a lei de cotas pode possuir uma lista de configurações que armazena os percentuais a serem reutilizados pelo usuário na Distribuicao, como por exemplo: PercentualPCD ou PercentualPPI.

A Distribuicao é o elemento que mantém a árvore de categorias, e os respectivos percentuais de reserva de vagas, iniciando com a CategoriaCota raiz, a qual contém uma sigla, uma descrição, uma Expression na qual será preenchida a reserva de vaga (percentuais fixos ou itens de Configuracao pré-definidos) e também uma lista de categorias filhas. Por fim, o elemento OrdemPrioridadeCotas faz referência às categorias criadas na Distribuicao. A Figura 26 apresenta um exemplo das associações entre os conceitos da DSL Cotas.

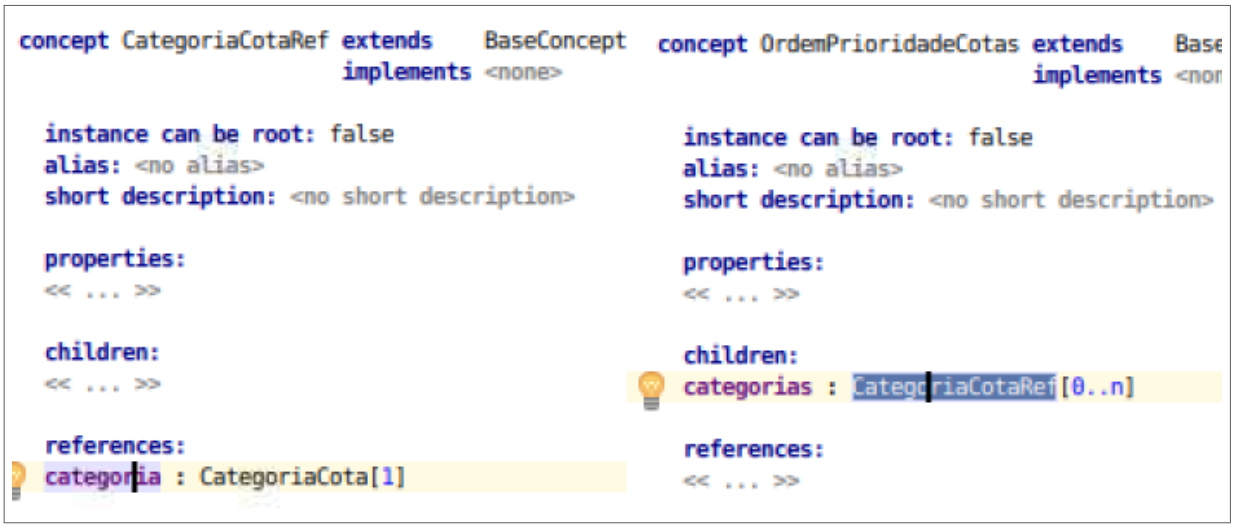
Figura 26 – Exemplo de associações entre os conceitos da DSL



Fonte: Elaborada pelo autor (2020).

Para possibilitar a relação entre as regras definidas, alguns componentes da DSL utilizam `references` para outros, permitindo que elementos já definidos possam ser acessados pelos comandos `control+espaço` no MPS. As `references` são restringidas pelo tipo do conceito alvo e pela cardinalidade. Por exemplo, o conceito `CategoriaCotaRef` possui uma referência para uma `CategoriaCota` e, por sua vez, o elemento `OrdemPrioridadeCotas` possui lista de `CategoriaCotaRef` para que seja possível indicar na linguagem a ordem de prioridade criada durante a distribuição de vagas (Figura 27).

Figura 27 – Definição de `References` no MPS



Fonte: Elaborada pelo autor (2020).

Na Subseção 4.2.2, são apresentados os editores criados para definição da sintaxe de cada conceito definido na modelagem.

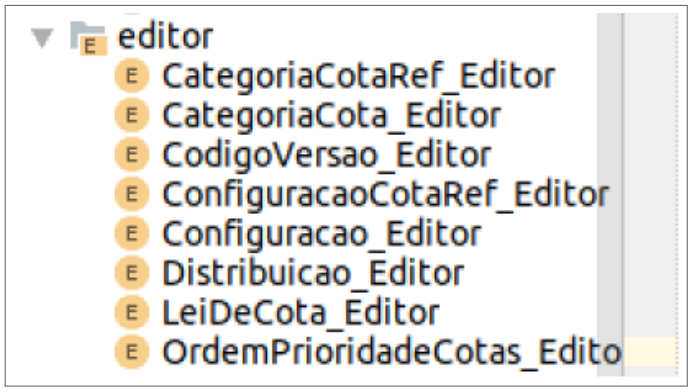
4.2.2 *Editores de conceitos*

O MPS oferece aos projetistas uma abordagem de definição de sintaxe abstrata por meio de editores construídos com a notação de células. O *designer* da linguagem combina as células do editor e as posiciona de maneira a refletir o *layout* final desejado da notação (JETBRAINS, 2019).

Por padrão, os elementos `Concepts` não possuem um editor associado, o editor padrão permite a edição direta da AST pelo usuário da DSL. No entanto, o editor padrão não é de simples entendimento para os usuários finais da linguagem, sendo necessário definir como aquele conceito deverá ser apresentado e editado pelo usuário final da DSL.

Nesse sentido, a Figura 28, apresenta a lista de editores criados para a DSL de cotas. As definições desses editores são apresentadas a seguir.

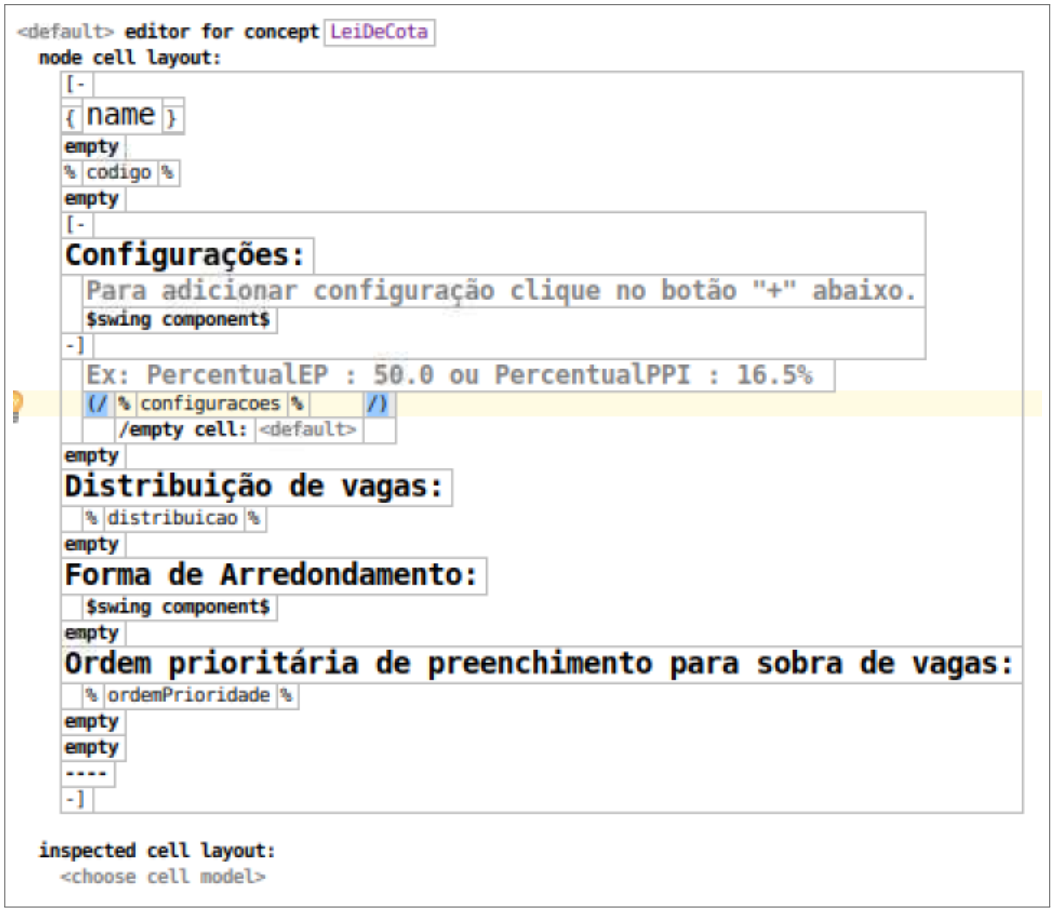
Figura 28 – Editores de conceitos da DSL Cotas



Fonte: Elaborada pelo autor (2020).

O editor `LeiDeCota_Editor` é responsável pela organização das informações relevantes para o modelo de cotas, na Figura 29, observa-se que ele é composto por uma coleção de células organizadas verticalmente, essa coleção é representada por `'[-'` e `'-]'`. Dentro de cada linha da coleção é possível utilizar propriedades básicas do conceito, como por exemplo, o campo `name`.

Figura 29 – Editor do conceito LeiDeCota



Fonte: Elaborada pelo autor (2020).

Também é permitido criar elementos estáticos para descrição de seções, como no caso do texto "Distribuição de Vagas" e referenciar os elementos `childrens` do conceito, por exemplo, `%configuracoes%`. No entanto, para que os elementos filhos sejam apresentados adequadamente para o usuário é necessário definir o respectivo editor.

Por se tratar de um editor de múltiplas projeções, é possível inserir nos editores do MPS alguns recursos gráficos disponíveis em sua linguagem base, Java, como por exemplo, elementos de `JTable` e `JButton` das bibliotecas gráficas `swing`. A Figura 30 demonstra um exemplo de como a célula `$swing_component$` é implementada para inserir o botão de adição de novas configurações na DSL.

Figura 30 – Exemplo de componente de projeção gráfica

Fonte: Elaborada pelo autor (2020).

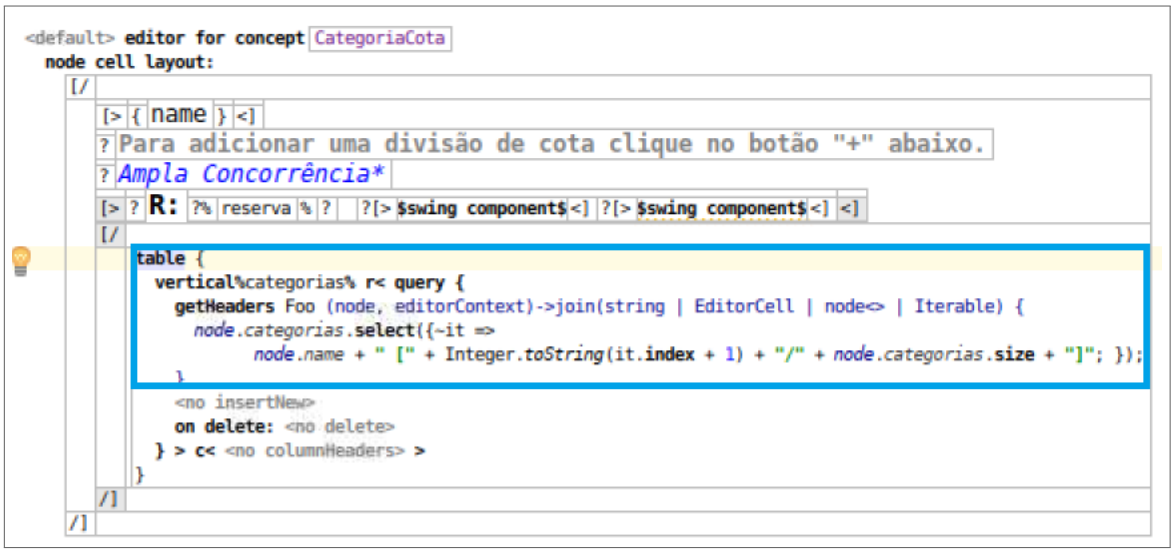
Com o objetivo de simplificar a edição da árvore de distribuição de vagas para o usuário final da DSL, foi utilizado o recurso `table` (`de.slisson.mps.tables`), que se trata de um plugin disponibilizado pelo pacote do `Mbeddr`.[1] Segundo Voelter e Lisson (2014), as tabelas podem ser usadas para representar coleções de dados estruturados que abordem problemas bidimensionais.

A organização em formato de tabelas permite uma melhor visualização para os usuários leigos durante a esquematização das regras da legislação. Desse modo, não se deixou de utilizar os benefícios de modelagem por meio de DSLs, porém agregou-se a tabela como um facilitador de entendimento. Ademais, o *plugin* utilizado possuía bons materiais de documentação e exemplos, permitindo criar a projeção gráfica de maneira simplificada.

[1] O Mbeddr é uma coleção de pacotes utilitários e extensões do MPS que permite a criação de muitos tipos diferentes de linguagens no MPS (ITEMIS, 2020).

Nesse contexto, na Figura 31 é possível observar o `CategoriaCota_Editor` em que as categorias filhas existentes são renderizadas com esse *plugin* por meio dos comandos `table{}` e `getHeaders`. Portanto, são iterados todos os elementos passados, de modo que as categorias sejam divididas por linhas indicando a sigla da categoria e o respectivo número de contagem, resultando no exemplo da Figura 32.

Figura 31 – Editor com o plugin `table` do `Mbeddr`



Fonte: Elaborada pelo autor (2020).

Figura 32 – Exemplo de resultado com uso do plugin `table`

Fonte: Elaborada pelo autor (2020).

Após a definição de editores para todos os conceitos do modelo da linguagem, foi necessário utilizar o recurso de `constraints` para criar restrições de escopo para acesso às referências entre os elementos da linguagem. Esse detalhamento é observado na Subseção 4.2.3.

4.2.3 *Restrições de escopo*

Na pesquisa ora apresentada utilizou-se o recurso de restrições de escopo, que se aplica nas restrições de acesso para referências de configurações e de categorias de cotas, dentro do contexto de um elemento de `LeiDeCota` criado pelo usuário da DSL. Possibilitando, por exemplo, que o MPS preencha o menu de *code-complete* somente valores válidos para aquela versão da lei de cotas e não mostre opções existentes em outras versões já definidas. Em relação a esse aspecto do MPS pode-se completar que:

> O aspecto de estrutura da linguagem pode ser insuficiente para expressar restrições avançadas para a DSL. O aspecto `constraints` do MPS fornece uma maneira de definir essas restrições adicionais. Referências dependem da definição de restrições de escopo, já que por padrão quando nenhum escopo é definido para uma referência, essas podem ser utilizadas em qualquer parte da modelagem da linguagem (JETBRAINS, 2019, s/p, tradução nossa).

Desse modo, nos conceitos de `CategoriaCotaRef` e `ConfiguracaoCotaRef` utilizou-se o conceito de definição de escopo herdado ou *inherited scopes* (Figura 33). Conforme JetBrains (2019, s/p, tradução nossa), "esse mecanismo delega a resolução do escopo aos ancestrais, que implementam o ScopeProvider".

Figura 33 – Definição de gerenciador de escopo

```
concepts constraints ConfiguracaoCotaRef {
  can be child <none>

  can be parent <none>

  can be ancestor <none>

  <<property constraints>>

  link {configuracao}
    referent set handler <none>
    scope inherited for Configuracao
    <no presentation (deprecated)>


  default scope <no default scope>
}
```

Fonte: Elaborada pelo autor (2020).

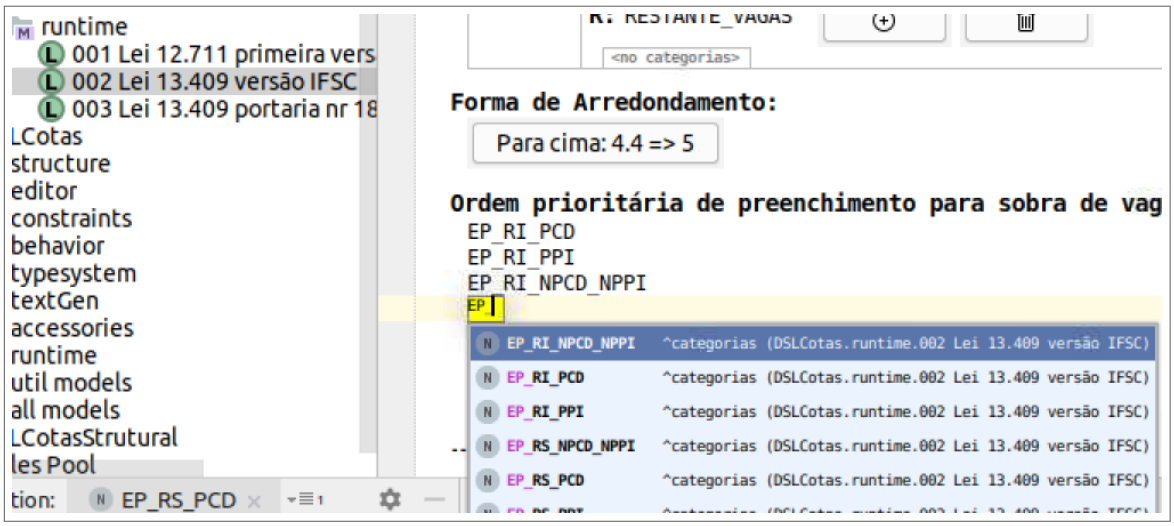
Nesse sentido, o MPS inicia a procura por provedores de escopos nos ancestrais de `Configuracao` até encontrar algum conceito que sobrescreva a implementação do método `getScope` da classe `ScopeProvider` do MPS.

No caso da DSL Cotas, o provedor de escopo foi o elemento `LeiDeCota` (Código Fonte 7), no qual todas as categorias da distribuição e as configurações de cota são adicionadas em uma lista (Linha 13) e retornadas por meio da criação de um `ListScope` (Linha 16), o qual é retornado sempre que acionado durante os comandos de `code-complete` pelo usuário da linguagem (Figura 34).

Código Fonte 7 – Implementação do ScopeProvider

```
1  concept behavior LeiDeCota {
     ...
3    public Scope getScope(concept<> kind, node<> child)
     overrides ScopeProvider.getScope {
5
     nlist<> categorias = new nlist<>;
7
     node<CategoriaCota> pai = this.distribuicao.categoria;
9
     adicionaCategoriasRecursivo(pai, categorias);
11
     nlist<Configuracao> configs = this.configuracoes;
13   categorias.addAll(configs);
     ...
15
     return new ListScope(categorias) {
17     public string getName(node<> child) {
         return child:INamedConcept.name;
19   ...
```

Figura 34 – *Code-complete* para referências da `LeiDeCota`



Fonte: Elaborada pelo autor (2020).

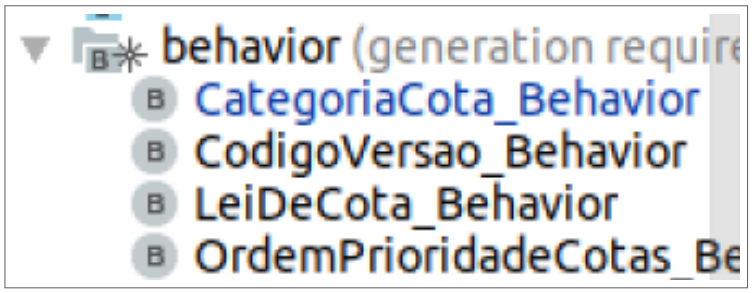
Na Subseção 4.2.4 é apresentado o aspecto de definição de comportamentos de conceitos (*behaviors*) da DSL Cotas.

4.2.4 *Comportamento dos elementos de conceito*

Durante a manipulação AST no MPS, operações comuns são frequentemente extraídas para métodos utilitários, a fim de simplificar a tarefa e de reutilizar as funcionalidades. O aspecto *behavior* possibilita a criação de métodos de instância, métodos estáticos e construtores de instância dos conceitos (JETBRAINS, 2019).

Na presente pesquisa, esse aspecto foi utilizado principalmente para simplificar implementações de verificações de inconsistências, fazer validações, implementar operações de *quick fixes* para o usuário da linguagem e também para apoio em questões relacionadas à extração das regras em formato JSON pelo recurso de `textGen`. A Figura 35 apresenta a lista dos `behaviors` presentes no código fonte da DSL Cotas.

Figura 35 – `Behaviors` da `DSL Cotas`



Fonte: Elaborada pelo autor (2020).

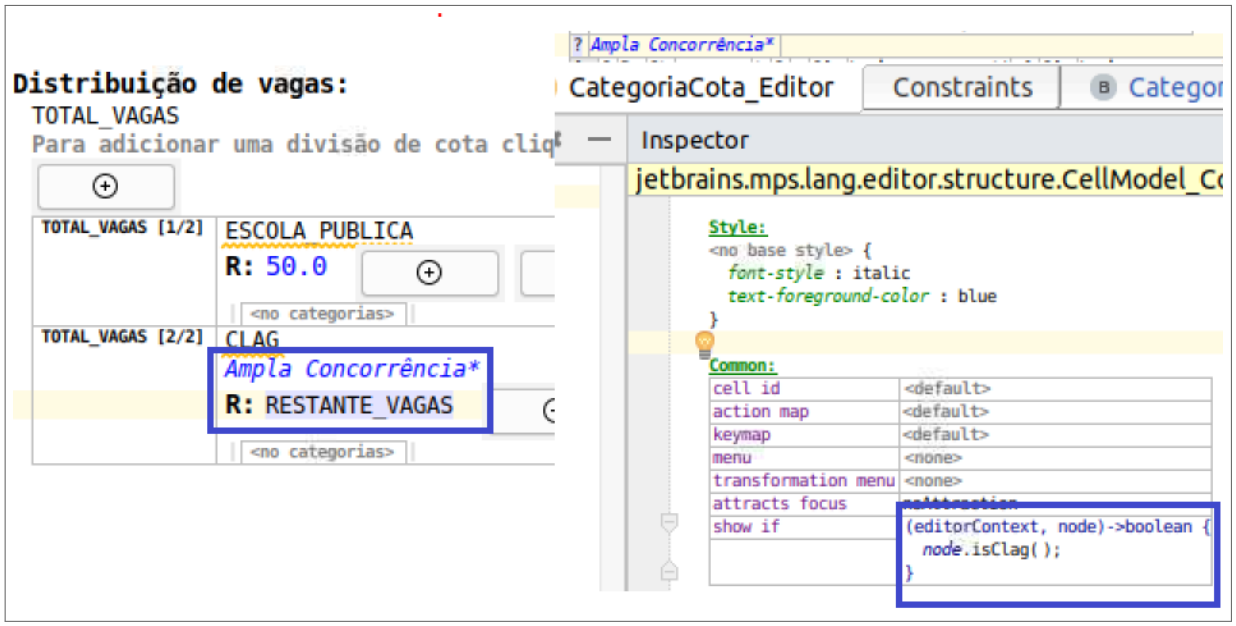
Um exemplo de concept `behavior` é apresentado na Figura 36, para o conceito `CategoriaCota`, em que o método `isClag` é responsável por verificar se a categoria corrente é a correspondente à ampla concorrência, possibilitando distingui-la e informar ao usuário o respectivo ramo na distribuição em que ela deve ser definida (Figura 37).

Figura 36 – Exemplo de *concept behavior* para `CategoriaCota`

```
concept behavior CategoriaCota {

  constructor {
    <no statements>
  }

  public boolean isClag( ) {

    if (!this.parent.parent.isInstanceOf(CategoriaCota) && this.parent.isInstanceOf(Categoriat
      list<node<CategoriaCota>> lista = this.parent:CategoriaCota.categorias;
      if (lista.last.name != null && this.name != null && lista.last.name.equals(this.name)) {
    }
    return false;
  }
```

Fonte: Elaborada pelo autor (2020).

Figura 37 – Uso do método `isClag` para identificar nível de Ampla Concorrência



Fonte: Elaborada pelo autor (2020).

Em continuação ao objetivo de descrever os recursos que fornecem apoio aos usuários da linguagem, a Subseção 4.2.5 apresenta o sistema de tipos criado para a DSL Cotas, o qual possibilita a definição controlada das regras de cotas para o usuário final.

4.2.5 *Sistema de tipos*

Segundo Campagne (2014), o aspecto *typesystem* do MPS é um componente da linguagem que torna possível calcular tipos da AST e também serve para definir e reportar erros de semântica para o usuário da linguagem. Nesse sentido, são descritos os seguintes conceitos utilizados na DSL, conforme documentação do MPS JetBrains (2019):

a) `Checking rules`: São regras de verificação que podem realizar uma busca no modelo à procura de padrões de erros conhecidos no código e relatá-los ao usuário, também são conhecidas como análise estática de código. O MPS diferencia os problemas conforme a severidade: *erros*, *warning* e *infos*, respectivamente, por meio de destaques nas cores vermelha, amarela e cinza;

b) `Quick Fix`: São responsáveis por implementar correções automáticas, fornecendo uma função de transformação do modelo, eliminando o problema relatado para o usuário. Por exemplo, um quick fix poderia ser criado para eliminar automaticamente trechos de código ou declarações duplicadas;

c) `Inference Rules`: Uma regra de inferência para um determinado *concept* é responsável principalmente por computar um tipo para instâncias desse conceito. O MPS fornece uma série de métodos de inferência, tais como: expressões de *typeof*, declarações *when*

concrete, *equations* e *inequations*, etc. Esses recursos podem ser utilizados, por exemplo, para verificar se o tipo de uma variável local é igual ao tipo da sua declaração.

A Tabela 9, apresenta a lista de checking rules implementadas na DSL Cotas, enquanto a Tabela 10, elenca os elementos de quick fixes implementados como recursos de apoio ao usuário.

Tabela 9 – *Checking rules* da DSL Cotas

categoria_resestante_vagas	Busca identificar se o último ramo de uma divisão de vagas é preenchida pelo usuário com a constante RESTANTE_VAGAS.
categoria_unica	Impede siglas de categorias de cotas duplicadas.
codigo_versao_lei	Valida padrões de preenchimento dos dados de identificação da lei de cotas.
configuracao_simples	Impede que o usuário preencha expressões complexas em uma definição do conceito Configuracao.
divisao_pelo_menos_dois_ramos	Garante que uma divisão de categoria possua pelo menos 2 (duas) categorias, exceto o ramo raiz.
filha_sem_reserva	Verifica se todas as categorias possuem o campo de percentual de reserva preenchido.
formato_sigla_cota	Faz o *matching* da String de siglas de cota, permitindo apenas caracteres alfanuméricos maiúsculos.
distribuicao_nome	Garante que o ramo raiz TOTAL_VAGAS não tenha o nome alterado pelo usuário da DSL.
ordem_prioridade_duplicada	Valida se o usuário informou uma categoria de cota duplicada no conceito OrdemPrioridadeCotas.
ordem_prioridade_catfilhas	Gera um *warning* para o usuário quando uma categoria não foi informada na seção OrdemPrioridadeCotas.
warning_nm_inic_cota	Sugere um nome de preenchimento para a sigla de cota que inicie com o prefixo da cota anterior, por exemplo: a cota de estudantes de renda inferior (sigla RI), que fica dentro da categoria de escola pública (sigla EP), tem como sugestão EP_RI, para facilitar a identificação.

Fonte: Elaborada pelo autor (2020).

Tabela 10 – *Quick fixes da DSL Cotas*

`preenche_totalvagas_name`	Associado ao erro gerado pela *checking rule* `distribuicao_nome`, faz o preenchimento automático do nome padrão no ramo raiz de distribuição de vagas.
`remover_categoria_duplicada`	Associado ao erro gerado pela *checking rule* `categoria_unica`, sugere a remoção automática do ramo de distribuição com a duplicidade.
`reserva_vagas_ultima_da_lista`	Associado ao erro gerado pela *checking rule* `categoria_resestante_vagas`, sugere a correção automática da distribuição de vagas para preenchimento adequado da constante `RESTANTE_VAGAS`.
`sugere_sigla_nome`	Associado ao *warning* gerado pela *checking rule* `warning_nm_inic_cota`, sugerindo a correção automática para padronização do nome das siglas de distribuição.

Fonte: Elaborada pelo autor (2020).

Com relação aos recursos de regras de inferência (*inference rules*), foram definidas as regras: `typeof_CategoriaCota` e `typeof_Configuracao`, criadas para verificar se os valores dos percentuais são preenchidos somente com expressões ou referências do tipo `double`.

As Figuras 38, 39 e 40, demonstram exemplos de como foram implementadas, respectivamente, as *checking rules*, as *quick fixes* e as *inference rules*.

Figura 38 – Exemplo de *checking rule*

```
checking rule divisao_pelo_menos_dois_ramos {
  applicable for concept = CategoriaCota as categoriaCota
  overrides false

  do {
    if (categoriaCota.categorias.size == 1 && categoriaCota.parent.isInstanceOf(Categor
      error "Uma divisão de cotas deve ter pelo menos 2 categorias!" -> categoriaCota;
    }
  }
}
```

Fonte: Elaborada pelo autor (2020).

Figura 39 – Exemplo de *quick fix*

```
quick fix sugere sigla nome

arguments:
node<CategoriaCota>  node

fields:
<< ... >>

description(node)->string {
  "Corrigir nome da categoria para começar com a anterior.";
}

execute(node)->void {
  if (node.name != null && !node.name.isEmpty( )) {
    node.name = node.parent:CategoriaCota.name + "_" + node.name;
  } else {
    node.name = node.parent:CategoriaCota.name;
  }
}
```

Fonte: Elaborada pelo autor (2020).

Figura 40 – Exemplo de *inference rule*

```
inference rule typeof_CategoriaCota {
  applicable for concept = CategoriaCota as categoriaCota
  applicable always
  overrides false

  do {
    typeof(categoriaCota.reserva) :==: <double>;
  }
}
```

Fonte: Elaborada pelo autor (2020).

A *checking rule* `divisao_pelo_menos_dois_ramos` é aplicada ao conceito `CategoriaCota` de modo a identificar se o usuário definiu pelo menos 2 (dois) ramos para a sua subdivisão, gerando um erro que é associado ao respectivo ramo da AST com problema (Figura 38). Uma *checking rule* pode ser associada a um *quick fix* com objetivo de sugerir correções para o usuário da DSL, a exemplo o *quick fix* `sugere_sigla_nome` faz a correção para que as siglas sejam identificadas conforme a sigla da categoria anterior, por exemplo "EP_"RI como sub-categoria da conta EP (Figura 39). Por fim, a *inference rule* `typeof_CategoriaCota` garante que o campo de reserva seja preenchido com valores e configurações definidas com o tipo `double` (Figura 40).

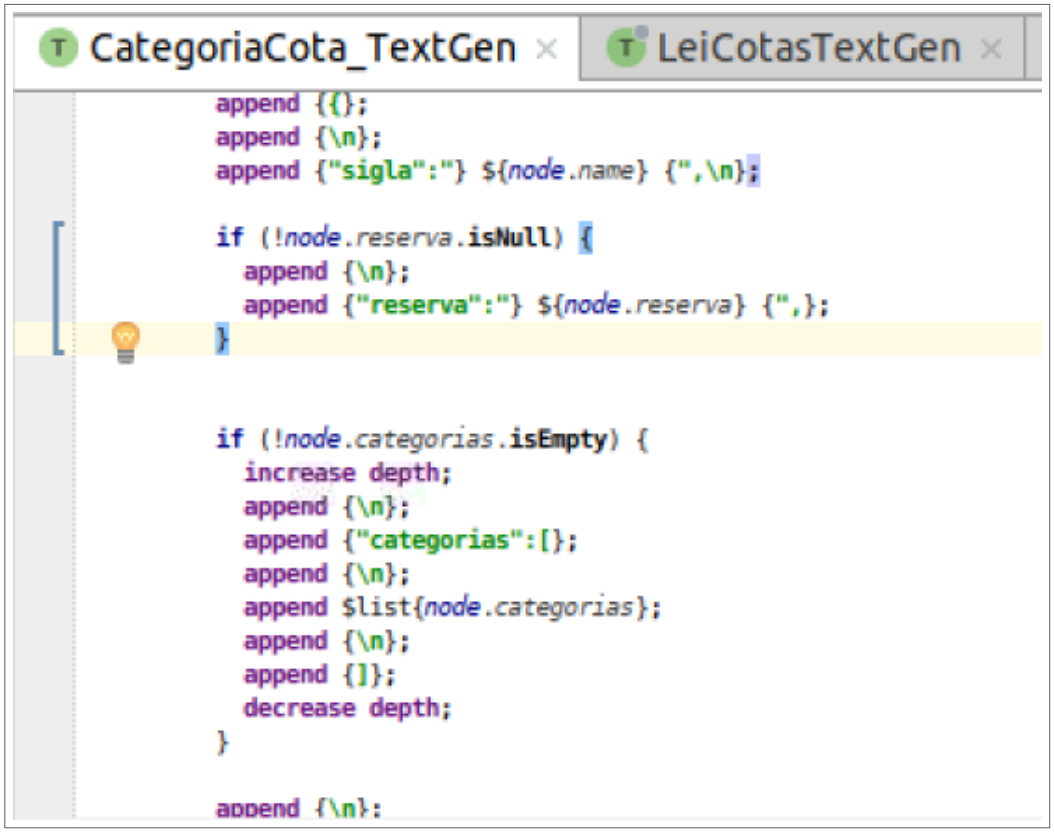
4.2.6 *Gerador textGen*

Na presente pesquisa optou-se por implementar o código fonte de classificação de candidatos em formato de API externa ao MPS, por meio de leitura das definições da DSL Cotas e, posterior, extração em formato JSON. Isso foi possível mediante ao uso do aspecto `TextGen` do MPS.

Esse aspecto define como o modelo da linguagem pode ser transformado em formato textual. Esse recurso é útil sempre que o modelo definido no MPS precise ser convertido diretamente para formato textual (JETBRAINS, 2019).

Para tanto, esse recurso requer que para cada conceito do modelo seja criado um *text gen component*, no qual devem ser utilizadas operações do tipo *append*, para que as informações sejam extraídas no formato de texto desejado. Um exemplo da geração do conceito `CategoriaCota` pode ser observado na Figura 41.

Figura 41 – Geração do conceito `CategoriaCota` utilizando `TextGen`



Fonte: Elaborada pelo autor (2020).

Esses componentes permitiram a construção da estrutura necessária para modelagem das regras do sistema de cotas. Em continuidade ao presente estudo, a Seção 4.3 detalha a API construída com o propósito de validar os objetivos que concernem à geração do código fonte de classificação e aprovação de candidatos.

4.3 A API DSL COTAS

Em relação ao desafio abordado nos objetivos específicos dessa pesquisa, no que diz respeito à evolução entre as versões e à geração de código fonte de classificação e aprovação de candidatos, foi criada uma API para prova de conceito sobre o uso dos padrões das regras definidas na DSL Cotas. Esta por sua vez implementa os algoritmos responsáveis por calcular o quadro de vagas, aprovar candidatos e definir a ordem de prioridade conforme as regras definidas pelo usuário.

Para tanto, a tecnologia escolhida foi o projeto SpringBoot, o qual segundo Walls (2016), oferece um novo paradigma de desenvolvimento de aplicações com o Spring Framework, possibilitando desenvolver aplicativos com mais agilidade, focando em atender as necessidades de funcionalidade com o mínimo de configurações que for necessário.

No que concerne à escolha para criação de serviços web, o setor de desenvolvimento do IFSC possui dois sistemas que envolvem processos seletivos, o primeiro é o sistema legado em PHP, tratado no Capítulo 2 e o segundo é o novo Sistema Integrado de Gestão (SIG) que também contém um módulo responsável pela criação de processos seletivos e foi desenvolvido em Java. Portanto, a camada de serviços foi empregada com o intuito de permitir a utilização em sistemas distintos de modo independente de linguagem alvo para geração de código fonte.

Nas próximas Subseções são detalhados os recursos do Spring que foram utilizados na criação das funcionalidades de classificação e aprovação de candidatos.

4.3.1 Componentes da API

Inicialmente, foi gerado um projeto SpringBoot por meio do site start.spring.io, no qual foram marcadas as opções de módulos necessários para o desenvolvimento da pesquisa, tais como:

a) Spring Boot Starter Web: Principal dependência do Spring, que fornece a camada de desenvolvimento utilizada para a construção de aplicações e de serviços web com o spring-web, incluindo um tomcat pré configurado e a biblioteca jackson, para manipulação de (JSON ou XML);

b) Spring Data JPA: Utilizado para simplificar a criação, seleção e manipulação das entidades de banco de dados criadas para processar o algoritmo de classificação e aprovação de candidatos;

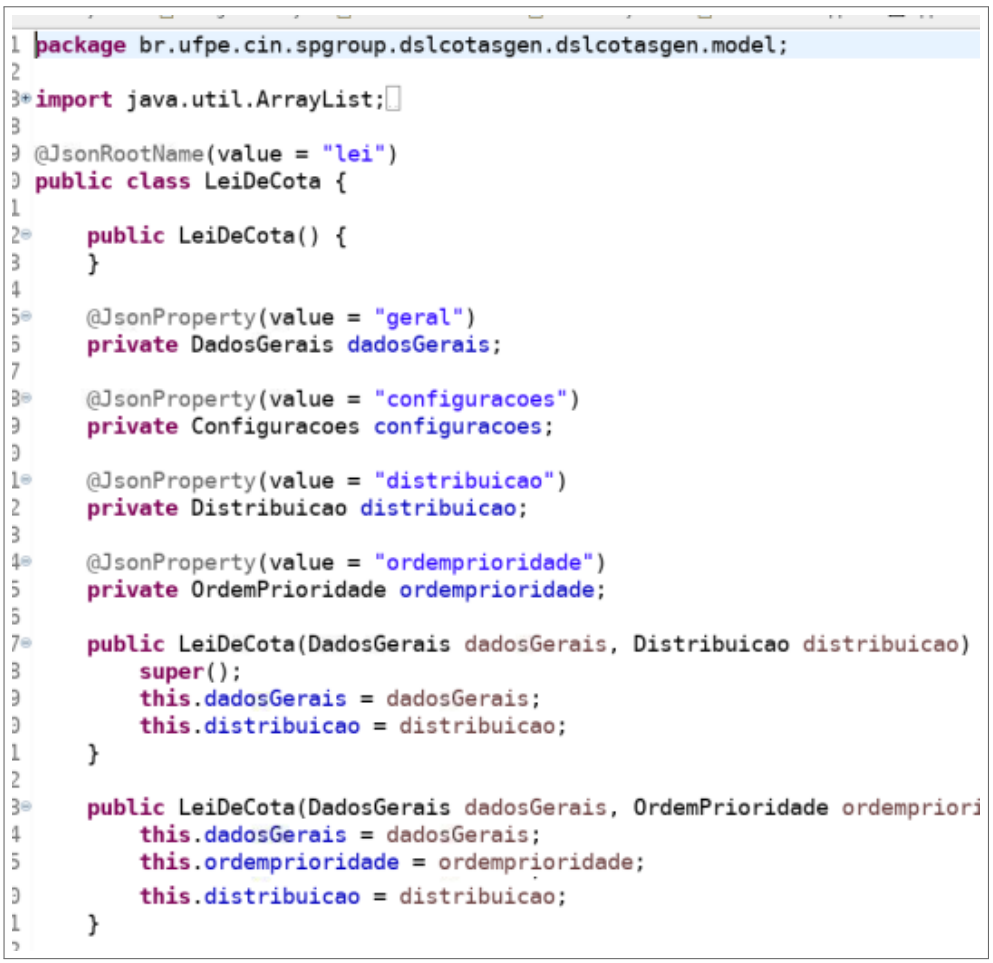
c) JDBC API: Para fornecer acesso ao banco de dados MYSQL do sistema de ingresso, utilizado para validação e possibilitar a comparação de resultados do histórico de candidatos dos processos do IFSC com o resultado processado pela API DSL Cotas;

d) Spring Reactive Web: Módulo do Spring utilizado para testar as requisições para os endpoints implementados com base na DSL, por meio do WebClient em conjunto com

o JUnit.

Após a inicialização do projeto foram criadas as entidades de modelo necessárias para fazer o mapeamento do arquivo JSON gerado pela DSL Cotas, para isso, foram utilizadas as anotações Json, disponíveis pela biblioteca jackson. Um exemplo do mapeamento é apresentado pela Figura 42.

Figura 42 – Anotações com a biblioteca jackson



```
package br.ufpe.cin.spgroup.dslcotasgen.dslcotasgen.model;

import java.util.ArrayList;

@JsonRootName(value = "lei")
public class LeiDeCota {

    public LeiDeCota() {
    }

    @JsonProperty(value = "geral")
    private DadosGerais dadosGerais;

    @JsonProperty(value = "configuracoes")
    private Configuracoes configuracoes;

    @JsonProperty(value = "distribuicao")
    private Distribuicao distribuicao;

    @JsonProperty(value = "ordemprioridade")
    private OrdemPrioridade ordemprioridade;

    public LeiDeCota(DadosGerais dadosGerais, Distribuicao distribuicao)
        super();
        this.dadosGerais = dadosGerais;
        this.distribuicao = distribuicao;
    }

    public LeiDeCota(DadosGerais dadosGerais, OrdemPrioridade ordempriori
        this.dadosGerais = dadosGerais;
        this.ordemprioridade = ordemprioridade;
        this.distribuicao = distribuicao;
    }
```

Fonte: Elaborada pelo autor (2020).

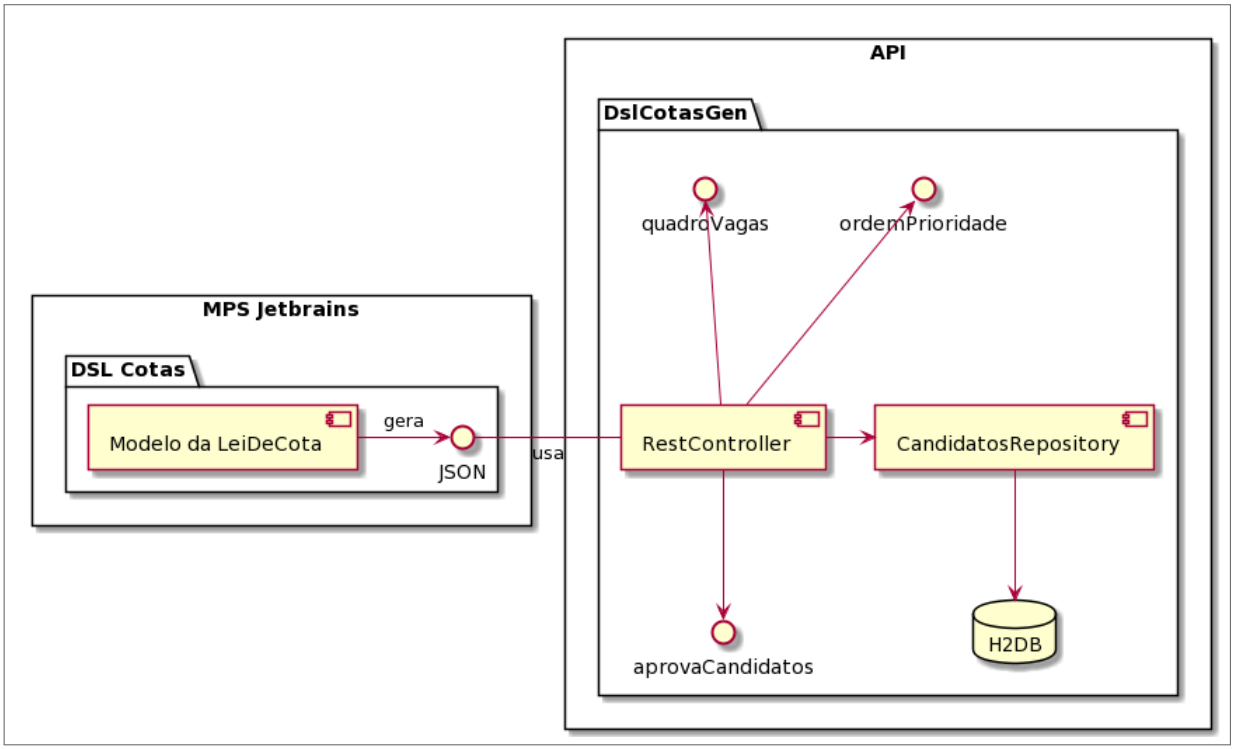
Desse modo, o modelo de regras gerado pela DSL Cotas é convertido para o objeto instância da classe LeiDeCota, que possui toda a estrutura de dados utilizada para armazenar e percorrer as regras definidas pelo usuário.

A API possui um controlador REST para cálculo de cotas, o qual fornece acesso às requisições HTTP que têm a função de: gerar o quadro de vagas, retornar a ordem de prioridade e aprovar uma lista de candidatos. Esse controlador por sua vez, utiliza os recursos do Spring Data JPA para conexão em um banco de dados H2, o qual é utilizado como meio de processar e aprovar a lista de candidatos passada.

O relacionamento entre os principais componentes da API e a DSL Cotas pode ser observado na Figura 43. Adicionalmente, o Código Fonte 8 mostra as assinaturas das interfaces

presentes no controlador REST.

Figura 43 – Componentes da API DSL Cotas



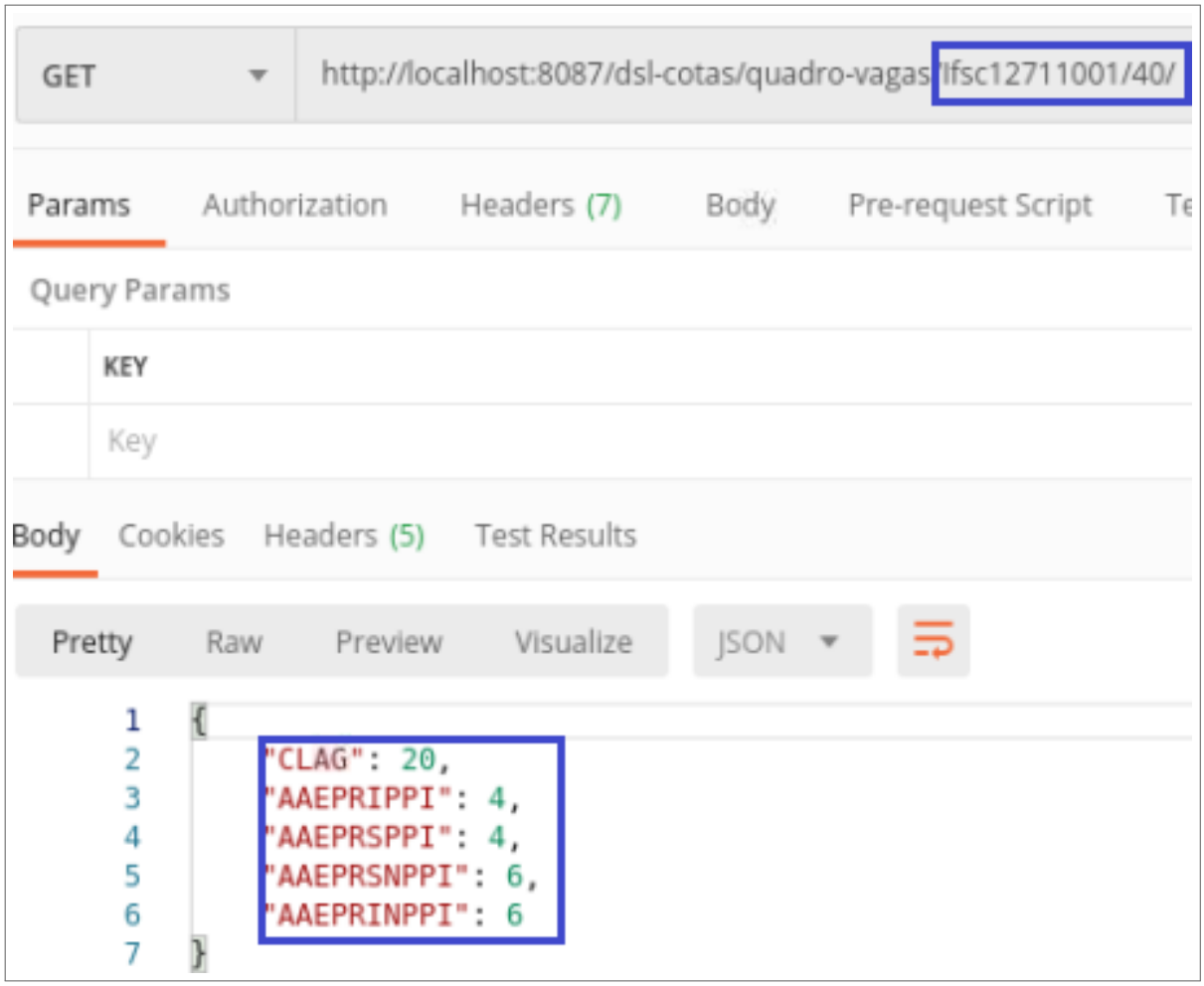
Fonte: Elaborada pelo autor (2020).

Código Fonte 8 – Endpoints no REST Controller

```
@RestController
public class CalculoCotasController {
...
    @GetMapping("/dsl-cotas/quadro-vagas/{versao}/{quantidade}")
    public Map<String, Integer> getQuadroVagas(@PathVariable String versao,
        @PathVariable Integer quantidade) {
    ...
    }
    @PostMapping("/dsl-cotas/aprova-candidatos/{versao}/{quantidade}")
    public List<Candidato> aprovaCandidatos(@PathVariable String versao,
        @PathVariable Integer quantidade,
        @RequestBody List<Candidato> candidatos)  {
       ...
     }
    @GetMapping("/dsl-cotas/ordem-prioridade/{versao}/")
    public Map<String, String> retornaOrdemPrioridade(@PathVariable String versao) {
        ...
```

O método quadro-vagas (Código Fonte 8, Linha 5), recebe como parâmetro a quantidade de vagas e o identificador da versão de lei, retornando uma lista de categorias com as respectivas siglas e quantidade de vagas (Figura 44).

Figura 44 – Retorno do método quadro-vagas

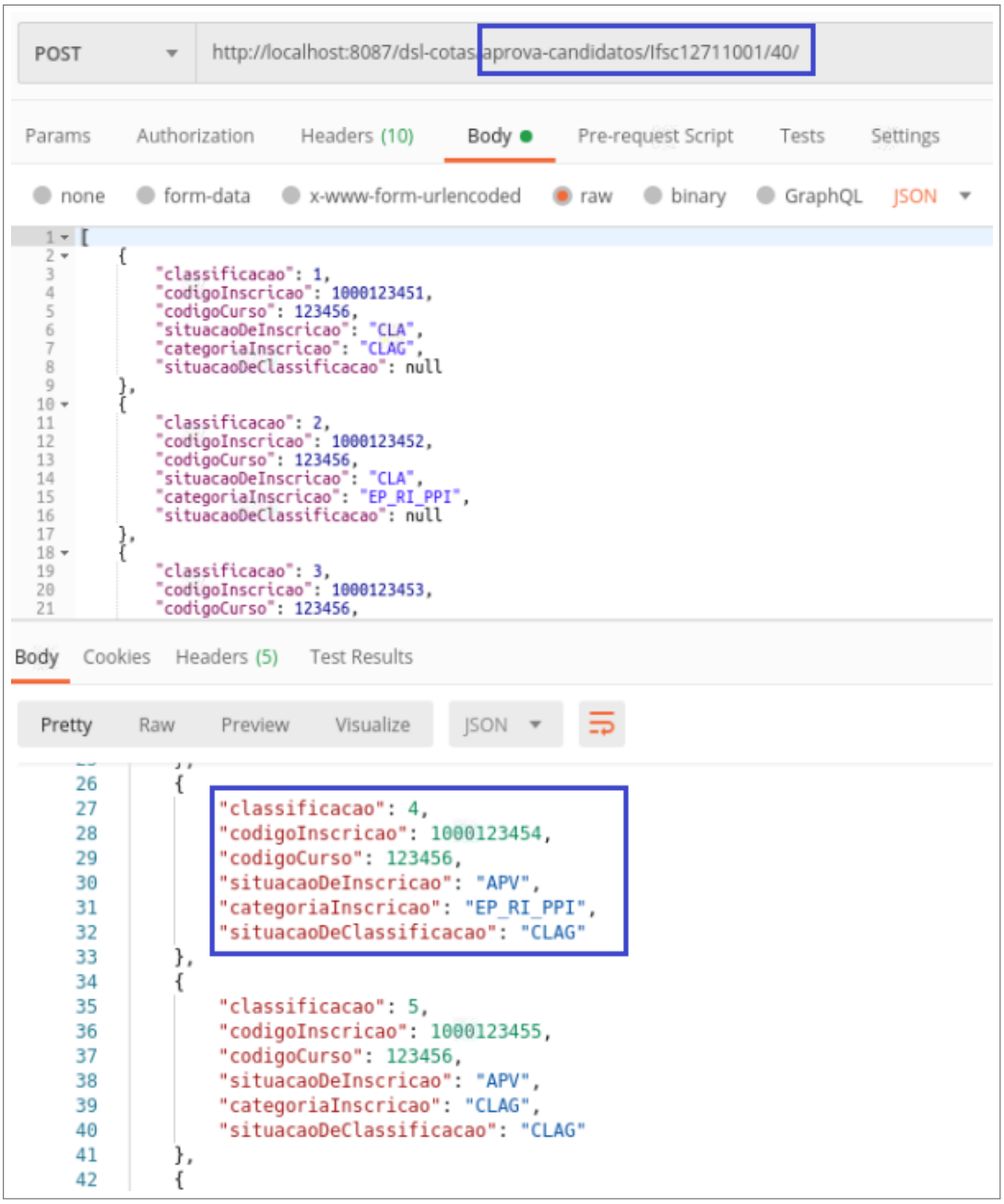


Fonte: Elaborada pelo autor (2020).

A aprovação é realizada no método aprova-candidatos (Código Fonte 8, Linha 8), que recebe adicionalmente o parâmetro da lista de candidatos que deve ser processada. O seu retorno se dá por meio da mesma lista de candidatos, no entanto, já com a identificação da categoria de aprovação, conforme a versão da lei de cotas desejada (Figura 45).

A lista de candidatos é composta pela ordem de classificação, o código de inscrição, o código do curso a que o candidato concorre, a situação de inscrição, categoria de concorrência selecionada na inscrição e a situação de classificação. O campo `situacaoDeInscricao` inicia com a situação classificado (CLA), para que após o processamento somente os candidatos aprovados sejam marcados como aprovados (APV). A categoria de classificação conforme o sistema de cotas é retornada no campo `situacaoDeClassificacao`.

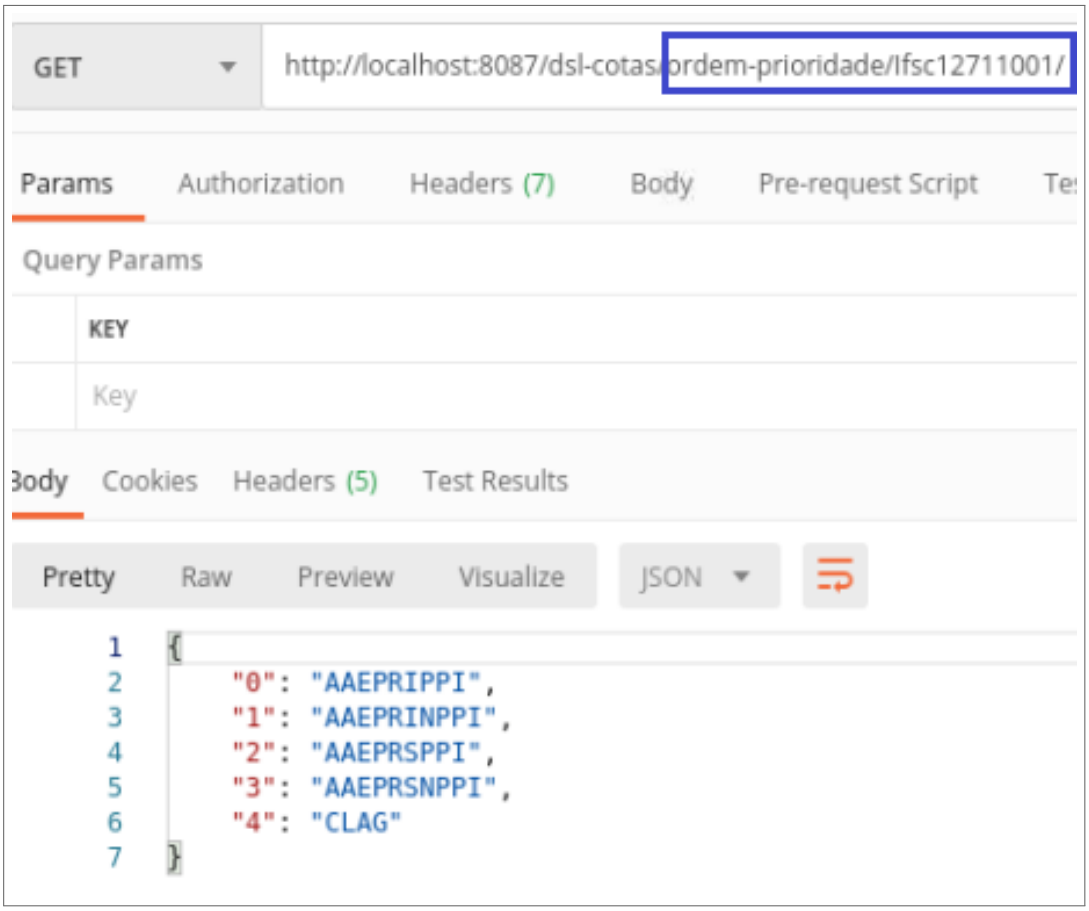
Figura 45 – Retorno do método aprova-candidatos



Fonte: Elaborada pelo autor (2020).

A interface de ordem de prioridade é atribuída ao método `ordem-prioridade` (Código Fonte 8, Linha 13), esse possui apenas o parâmetro identificador da lei e devolve a lista de siglas conforme a definição de prioridade feita pelo usuário na DSL Cotas (Figura 46).

Figura 46 – Retorno do método ordem-prioridade



Fonte: Elaborada pelo autor (2020).

No Capítulo seguinte serão apresentados os procedimentos metodológicos utilizados na pesquisa.

5 METODOLOGIA

Este Capítulo apresenta a metodologia utilizada para delinear o presente estudo. Desse modo, a seção foi organizada em subcapítulos que tratam, respectivamente, da classificação da pesquisa, das etapas realizadas para execução do processo de implementação e de análise da DSL de Cotas, bem como do ambiente construído pelo autor para avaliação dos objetos de pesquisa.

5.1 CLASSIFICAÇÃO E ETAPAS DA PESQUISA

A elaboração do presente estudo caracterizou-se como uma pesquisa aplicada quanto à sua natureza. Segundo Silva (2005), a pesquisa aplicada visa gerar conhecimentos para aplicação prática e voltados à solução de problemas específicos e interesses locais.

Do ponto de vista da abordagem do problema a pesquisa classifica-se como qualitativa. Para Silva (2005), a pesquisa qualitativa preocupa-se com o processo de investigação, sendo que há uma relação dinâmica entre o objeto a ser estudado e o sujeito, estabelecendo-se um vínculo indispensável entre o mundo objetivo e a subjetividade do sujeito. Somada a essa questão, destaca-se que a interpretação dos fenômenos e a atribuição de significados pelo pesquisador é o instrumento-chave da pesquisa qualitativa.

Em relação aos objetivos, a pesquisa é exploratória, pois buscou:

> proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a constituir hipóteses[...] seu planejamento é, portanto, bastante flexível, de modo que possibilite a consideração dos mais variados aspectos relativos ao fato estudado (GIL, 2002, p. 41).

No que concerne aos procedimentos técnicos utilizou-se para o presente estudo: a pesquisa bibliográfica, a documental e o estudo de caso (GIL, 2002). Esses procedimentos técnicos referem-se à:

a) Pesquisa bibliográfica: realizada a partir de buscas aos documentos dos principais autores sobre DSL: Fowler (2005, 2008) e Voelter (2011, 2013, 2014, 2018). A consulta aos referidos autores possibilitou a identificação de outros materiais sobre a temática, bem como outros pesquisadores utilizados no decorrer do texto. Ademais, utilizou-se a legislação sobre as leis de cotas no sistema de ensino público federal, que foram descritas no Capítulo 2;

b) Pesquisa documental: utilizada durante a análise do histórico do controle de versão do IFSC, mediante à busca manual de *commits* e tarefas na ferramenta `Trello` que continham as palavras-chave "cotas, sistema de cotas e Departamento de Ingresso (DEING)" e a posterior identificação dos arquivos relevantes na classificação de candidatos. Nessa

análise foram encontrados arquivos contendo as implementações das funcionalidades apresentadas no Capítulo 2, no qual foram detalhadas as linhas de código e as funções envolvidas em 3 (três) versões do sistema de ingresso. Adicionalmente, utilizou-se um *e-mail* (APÊNDICE A), contendo o histórico de demandas sobre o assunto, o qual foi fornecido pelo responsável no setor de ingresso (DEING);

c) Estudo de caso: Para Gil (2002), o estudo de caso tem o propósito de explorar situações da vida real no contexto do objeto estudado, buscando-se analisar e formular hipóteses sobre a sua aplicação. Por esses motivos foi utilizado nessa pesquisa articulando-se ao estudo de usabilidade da DSL, baseado em Nielsen (2012). Para esse autor, quando o público alvo de usuários é variado, normalmente, os testes de usabilidade são realizados em grupos formados de 3 (três) a 4 (quatro) integrantes. Desse modo, considerando os objetivos da presente pesquisa, foram convidadas 20 pessoas com diferentes perfis de experiência, as quais responderam a um exercício proposto para desenvolvimento na DSL, bem como a um questionário após a realização dos testes.

O processo do estudo de caso foi conduzido com base no trabalho de Runeson e Höst (2009), o qual define 5 (cinco) passos principais para serem considerados:

a) Definir o estudo de caso: objetivos são definidos e o estudo de caso é planejado;

b) Preparar a coleta de dados: procedimentos de coleta de dados são definidos;

c) Evidenciar a coleta de dados: execução e registro da coleta de dados;

d) Analisar os dados coletados;

e) Relatar os resultados.

Considerando a pergunta central da pesquisa, "Como utilizar uma linguagem de definição de regras do sistema de cotas de modo a melhorar a comunicação entre usuários especialistas e desenvolvedores e, assim, verificar a possibilidade de melhoria no processo de desenvolvimento das regras presentes na legislação?", foi possível definir as seguintes questões do estudo de caso:

a) É possível a criação de uma linguagem específica de domínio que padronize as definições em comum nas regras de distribuição de vagas?

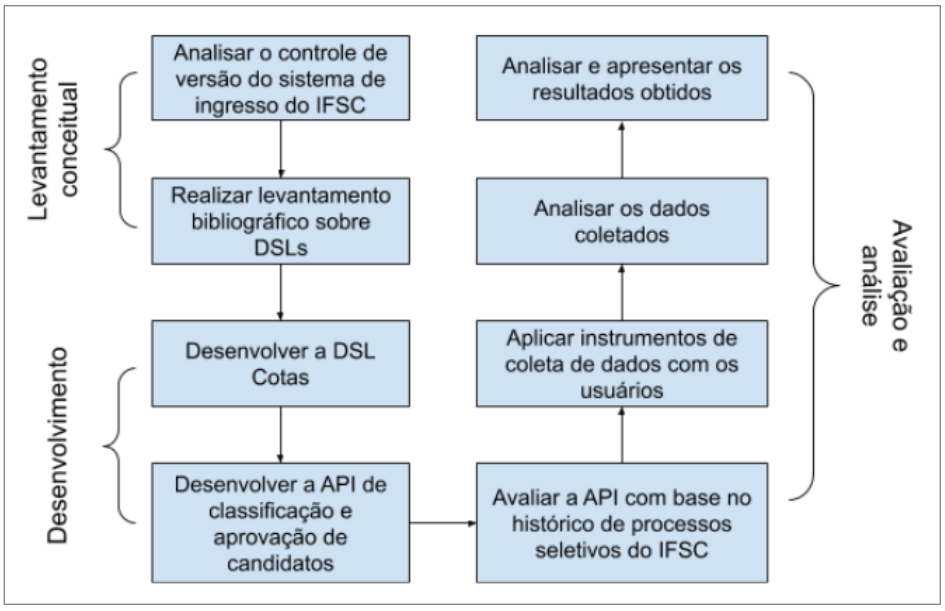
b) A utilização da linguagem proposta pode ajudar na produtividade durante o desenvolvimento de alterações na legislação, reduzindo a quantidade de linhas de código a serem implementadas manualmente pelos desenvolvedores?

c) A definição de regras do sistema de cotas pode ser facilitada com apoio de uma IDE que faça validações e forneça recursos específicos para a linguagem proposta?

Para a etapa de preparação da coleta foram apresentados os materiais de apoio elaborados (vídeo explicativo e manual de uso da DSL), e na sequência, o exercício na DSL (APÊNDICE B) e o questionário sobre a experiência de uso com os usuários (APÊNDICE C). Esses materiais foram utilizados como instrumentos de coleta de dados com o objetivo de fazer uma avaliação preliminar de uso da DSL Cotas, assim como, de identificar o perfil dos usuários e as principais dificuldades encontradas por eles. O detalhamento do exercício e do questionário serão retomados nas próximas Seções. As evidências da coleta, a análise dos dados coletados e o relato de resultados serão descritos respectivamente nos Capítulos 6 e 7.

A fim de situar o leitor acerca das etapas desenvolvidas na pesquisa, apresenta-se a Figura 47. Nessa Figura, as duas primeiras etapas são responsáveis pelo levantamento conceitual, no qual realizou-se a análise do controle de versão do sistema de ingresso do IFSC e o levantamento dos principais estudos e pesquisas sobre DSLs, respectivamente. As etapas seguintes referem-se ao desenvolvimento da DSL Cotas e da API de classificação e aprovação de candidatos. Na sequência, os procedimentos de avaliação e de análise foram compostos pelas etapas de avaliação da API com base no histórico de processos seletivos do IFSC e de aplicação dos instrumentos de coleta com os usuários da DSL Cotas. Com base nesses dados foi possível analisar a viabilidade de melhoria de comunicação entre os usuários de negócio e desenvolvedores a partir da DSL Cotas e, por fim, apresentar os resultados obtidos.

Figura 47 – Etapas da pesquisa



Fonte: Elaborada pelo autor (2020).

Desse modo, a Seção 5.2 descreve a metodologia de avaliação da DSL, abordando o perfil dos usuários, o detalhamento do exercício, o questionário aplicado e os critérios utilizados para avaliação da API.

5.2 AMBIENTE DA PESQUISA

Esta Seção apresenta o ambiente elaborado para verificar a validade da DSL de Cotas em relação aos objetivos da presente pesquisa. Desse modo, são descritos os critérios utilizados para definir os grupos de usuários convidados, os critérios de avaliação da API implementada para classificar e aprovar os candidatos, e também apresentar o questionário aplicado com os participantes.

5.2.1 Metodologia de avaliação da DSL

Segundo Moran (2018), estudos qualitativos de usabilidade tentam compreender o pensamento e as dificuldades vivenciadas pelos indivíduos, geralmente este tipo de estudo apresenta aos usuários atividades abertas que têm o potencial de expor problemas na interface do sistema.

Moran (2018) também cita alguns princípios básicos para elaboração de tarefas para qualquer tipo de teste com usuário, tais como:

a) Atente-se ao que seus usuários precisam fazer com o seu produto para inspirar suas tarefas;

b) Evite fornecer dicas para a sua tarefa. Não descreva os passos exatos que o usuário precisa fazer, deixe que eles descubram por si mesmos;

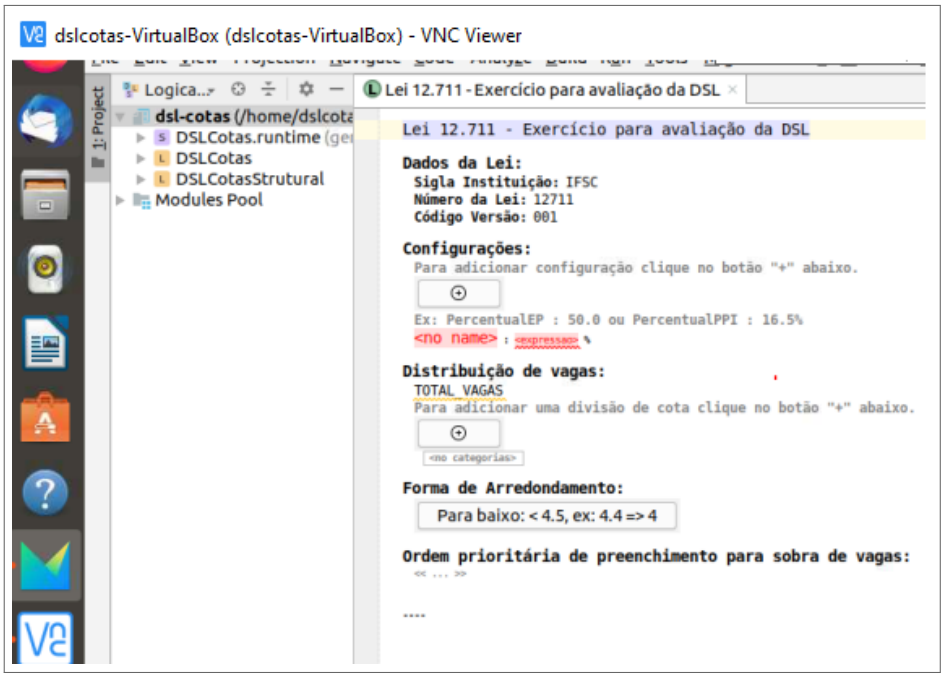
c) Sempre faça um teste piloto para as tarefas, isso é essencial para evitar obter acidentalmente dados incorretos ou ruins.

Considerando essas orientações foi realizado um teste piloto da DSL em Abril de 2020. Para tanto, foi instalado o software *TeamViewer* para acesso remoto a uma máquina virtual *Linux* contendo a ferramenta MPS, no qual a DSL foi disponibilizada para teste. Todavia, observou-se que a ferramenta de acesso remoto dificultava o uso da linguagem, no sentido de apresentar lentidão durante a execução dos comandos. Outro problema percebido foi a falta de instruções sobre o uso da linguagem, bem como foram encontradas dificuldades de visualização de alguns elementos, tendo em vista que a fonte fornecida era pequena.

A partir dessa avaliação foram feitas as correções necessárias, por meio da troca da ferramenta de acesso remoto para o software *VNCViewer* e a elaboração de um documento para subsidiar os usuários. Desse modo, foi elaborado um manual de utilização da DSL (APÊNDICE C), no qual foram apresentados os objetivos e os elementos da linguagem, os principais comandos para edição das regras de negócio e o link para o vídeo explicativo também elaborado pelo autor, no qual se exibiu um exemplo de uso das principais funcionalidades. O ambiente para avaliação da DSL pode ser observado na Figura 48.

No que concerne à seleção dos usuários para os testes, baseando-se em Nielsen (2012), entre os meses de Abril e Maio de 2020 buscou-se 20 pessoas com perfis diferentes que foram agrupadas de acordo com as seguintes categorias: 1) não desenvolvedores e especialistas na

Figura 48 – Ambiente de avaliação no MPS

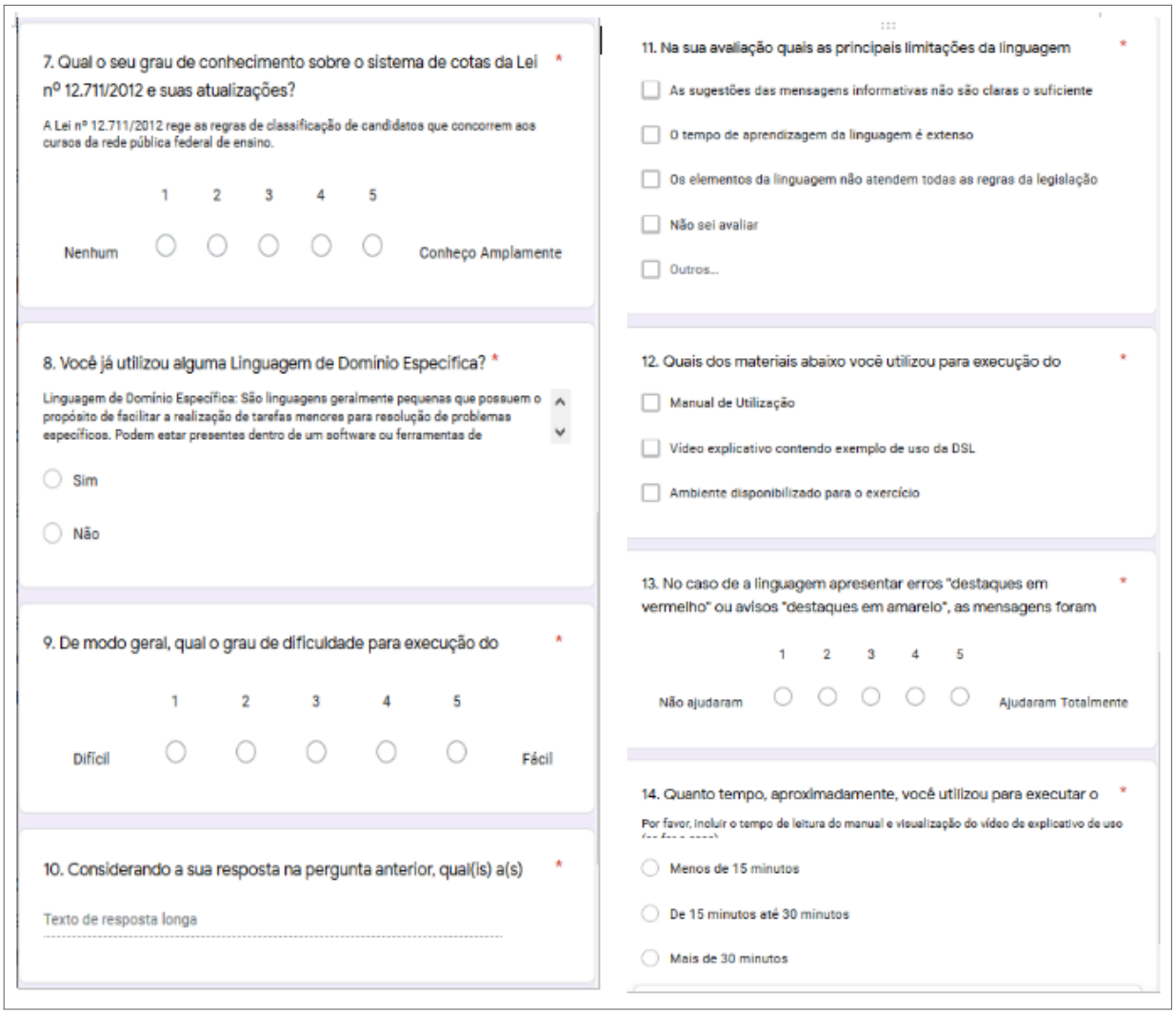


Fonte: Elaborada pelo autor (2020).

legislação de cotas; 2) desenvolvedores e não especialistas na legislação de cotas; 3) desenvolvedores e especialistas na legislação de cotas; 4) não desenvolvedores e não especialistas na legislação de cotas. Ao longo da análise dos dados estas foram identificadas no texto, respectivamente, como: NDEV-ESP; DEV-NESP; DEV-ESP; NDEV-NESP.

A coleta de dados iniciou-se por meio de envio de e-mail (APÊNDICE D) para cada participante, no qual constavam o manual de utilização da DSL, a instrução para acesso remoto, o link para o vídeo explicativo e o exercício aberto para descrição da primeira versão de lei nº 12.711 na DSL Cotas (Capítulo 2, Seção 2.2.1). Também foi sugerido para que os participantes contabilizassem o tempo utilizado para desenvolvimento do exercício. A instrução final do e-mail continha o link de acesso ao questionário de avaliação (Figura 49). O detalhamento completo do questionário, assim como as perguntas avaliadas estão presentes no Capítulo 6 desse documento.

Figura 49 – Questionário aplicado



Fonte: Elaborada pelo autor (2020).

Na sequência, a Subseção 5.2.2 descreve os procedimentos metodológicos criados de modo a possibilitar a avaliação da API.

5.2.2 Metodologia de avaliação da API

Tendo como base a DSL elaborada, uma API foi desenvolvida para disponibilizar *endpoints* responsáveis pelos métodos de classificação e de aprovação de candidatos. Para tanto, as regras definidas na linguagem foram exportadas em formato de arquivos JSON, os quais foram utilizados no framework *SpringBoot* [1] para geração das operações API.

Como fator relevante para a escolha dessa tecnologia, destaca-se a necessidade de aplicação interna no IFSC em função da equipe possuir conhecimento na linguagem de desenvolvimento

[1] Segundo Webb et al. (2018), o Spring Boot é um projeto da Spring que visa simplificar a criação de aplicações Java com base no Spring *Framework*, para que se tenha uma aplicação inicial sem a necessidade de muitas configurações.

Java. O SpringBoot já é utilizado em outros projetos do setor de desenvolvimento de sistemas, o que pode favorecer a aceitação e a implantação por parte da equipe.

Desse modo, foi possível executar a classificação dos candidatos e fazer a comparação com os resultados de 13494 candidatos. No total foram selecionados, aleatoriamente, 16 processos na base do sistema de ingresso do IFSC de 2013 a 2020, contendo as diferentes versões de lei nas quais foram realizadas classificações pelas regras de cotas, conforme apresentado na Tabela 11. Para documentar as divergências entre os resultados da API e do histórico presente no banco de dados do sistema utilizou-se o recurso de *issues* disponível no sistema de controle de versão *github* (Figura 50). Para cada curso com diferenças nas situações de classificações foi descrita a versão de lei aplicada na classificação, a quantidade de vagas, o problema apresentado, o motivo encontrado e uma possível solução.

Tabela 11 – Processos utilizados para avaliação da API

2013/1 - Cursos Técnicos Integrados
2013/1 - Cursos Técnicos Concomitantes
2013/1 - Cursos Técnicos Subsequentes
2013/2 - Curso Técnico Subsequente - Palhoça - Edital 05/2013/2
2014/1 - Cursos Técnicos Integrados - Edital 01/DEING/2014/1
2014/1 - PROEJA/Técnico - Edital 06/DEING/2014/1
2015/2 - Cursos de Graduação - Edital 04/DEING/2015-2
2016/1 - Cursos Técnicos - Sorteio - Edital 05/DEING/2016/1
2016/1 - Cursos Técnicos Integrados - Exame de Classificação - Edital 01/DEING/2016/1
2017/1 - Cursos Técnicos - Integrados e Concomitantes - Edital 01/DEING/2017/1
2018/1 - Cursos Técnicos - Edital 07/DEING/2018/1 - Subsequentes e Concomitantes
2018/1 - Graduação - SiSU - Edital 13/DEING/2018/1
2018/2 - Técnicos Itajaí - Sorteio - Edital 02/DEING/2018/2
2019/1 - Técnicos Integrados - Exame de Classificação - Edital 08/DEING/2019/1
2019/1 - Cursos Técnicos - Sorteio - Edital 11/DEING/2019/1 - Integrados
2020/1 - Cursos Técnicos - Sorteio - Edital 15/DEING/2020/1 - Subsequentes

Fonte: Base de dados do IFSC (2020).

Figura 50 – Issues da API no GitHub

Fonte: Elaborada pelo autor (2020).

No Capítulo a seguir, é elencada a análise dos resultados levantados durante a coleta de dados com os usuários, assim como a análise de testes realizados com os endpoints da API. Também são apresentadas algumas mudanças na DSL Cotas que foram provenientes das sugestões dos usuários que participaram do estudo.

6 ANÁLISE DOS RESULTADOS DA PESQUISA

Segundo Poltronieri et al. (2018), há muito esforço para criar e usar DSLs como recurso de facilitar a construção do sistema, aumentar a produtividade e facilitar a manutenção. No entanto, as DSLs tratam do problema de domínio (usuários especialistas) e não somente do domínio de solução (desenvolvedores/engenheiros de software), isso significa que nem sempre há aceitação e entendimento entre esses usuários.

Nesse sentido, este Capítulo apresenta a análise dos dados obtidos em relação aos 2 (dois) instrumentos de coleta de dados (exercício de avaliação da DSL Cotas e o questionário aplicado com usuários), bem como, os resultados com os testes da API DSL Cotas. Esses instrumentos têm como foco verificar se a DSL Cotas atinge seus objetivos em relação à perspectiva dos diferentes grupos de usuários.

Para melhor sistematizar a organização dos dados optou-se por apresentar a análise por grupos de usuários, considerando os diferentes perfis de experiência, os resultados foram agrupados de acordo com os 4 (quatro) grupos: DEV-NESP; DEV-ESP; NDEV-ESP e NDEV-NESP, cujos perfis são definidos na Seção 6.1.

Na Seção 6.2 apresenta-se o exercício aplicado com os usuários na DSL Cotas, assim como os resultados obtidos com a aplicação do questionário após a realização do exercício e algumas mudanças desenvolvidas na DSL Cotas com base nas sugestões dos participantes. Por fim, a Seção 6.3 detalha os resultados dos testes com a API.

6.1 IDENTIFICAÇÃO DOS PERFIS DE USUÁRIOS

A Figura 51 apresenta o perfil de cada usuário que participou da avaliação de uso da DSL Cotas. Os critérios para inserção em cada grupo são descritos a seguir:

a) `DEV-ESP`: usuários que possuem conhecimento sobre a lei nº 12.711 e suas atualizações, são desenvolvedores e já atuaram em sistemas ou processos seletivos que utilizam o sistema de cotas;

b) `DEV-NESP`: usuários com perfil de desenvolvedor sem conhecimento ou com conhecimento restrito sobre a legislação;

c) `NDEV-ESP`: usuários com conhecimento na legislação, não desenvolvedores. Em sua maioria usuários que trabalham em setores que organizam processos seletivos relacionados ao sistema de cotas;

d) `NDEV-NESP`: usuários não conhecedores da legislação e não desenvolvedores, de modo que pudesse ser identificado se as pessoas com esse perfil teriam dificuldades com a utilização da linguagem.

Figura 51 – Perfil dos usuários participantes

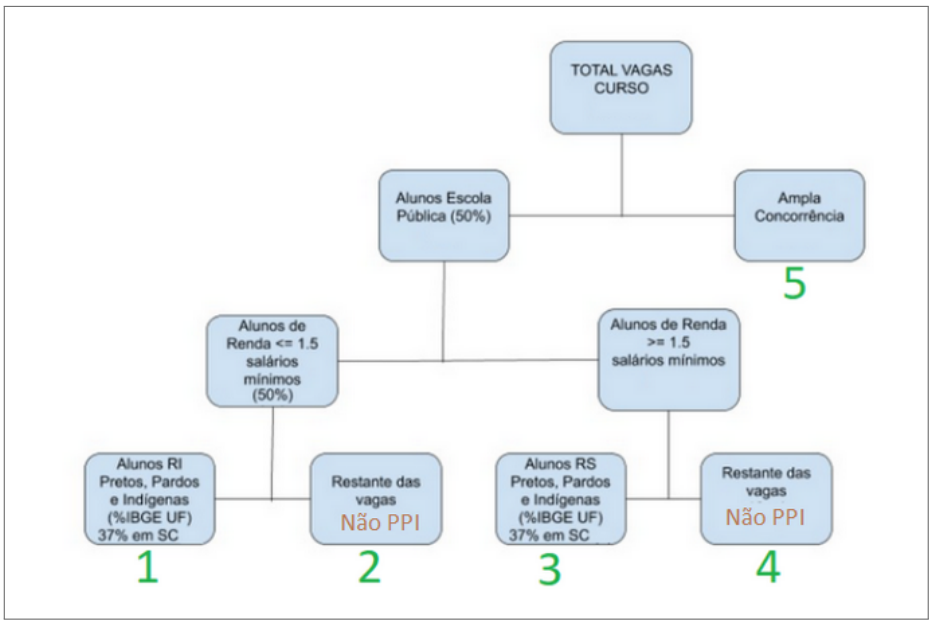
Grupo	ID	Desenvolvedor	Conhecimento na legislação	Idade	Escolaridade	Formação acadêmica	Cargo/Setor	Tempo de Experiência Profissional	Atuou em sistemas de cotas
DEV-ESP	3	Sim	4	> 10 anos	ME	Ciências Da Computação	TI	> 10 anos	Sim
	12	Sim	4	5-10 anos	ME	Ciência Da Computação	TI	5-10 anos	Sim
	16	Sim	5	3-5 anos	ES	Tecnólogo Em Gestão De Ti	Deing	3-5 anos	Sim
	20	Sim	3	> 10 anos	ME	Sistemas De Informação	TI	> 10 anos	Sim
DEV-NESP	5	Sim	2	> 10 anos	ES	Ciência Da Computação	TI	> 10 anos	Sim
	11	Sim	1	> 10 anos	ME	Sistema De Informação	TI	> 10 anos	Sim
	14	Sim	1	5-10 anos	ES	Tecnologo Rede De Computadores.	TI	5-10 anos	Não
NDEV-ESP	1	Não	4	3-5 anos	ES	Sistemas De Informação	TI	3-5 anos	Não
	4	Não	5	> 10 anos	ES	Agronomia	Deing	> 10 anos	Não
	7	Não	5	> 10 anos	ME	Administração	Deing	> 10 anos	Sim
	10	Não	5	3-5 anos	ES	Gestão Da Tecnologia Da Informação	TI	3-5 anos	Sim
	13	Não	5	> 10 anos	ES	Geografia	Deing	> 10 anos	Sim
	15	Não	4	3-5 anos	ME	Gestão Pública	Registro Acadêmico	3-5 anos	Sim
	17	Não	5	5-10 anos	ES	Administração	Deing	5-10 anos	Sim
	18	Não	4	0-2 anos	ME	Sistemas De Informação	TI	0-2 anos	Não
NDEV-NESP	2	Não	3	0-2 anos	ES	Pedagogia	Estágiaria	0-2 anos	Não
	6	Não	3	> 10 anos	ES	Licenciatura Em Matemática	Professor Ensino Fundamental/Médio	> 10 anos	Não
	8	Não	4	> 10 anos	ME	Gestão Em Ti	TI	> 10 anos	Não
	9	Não	2	> 10 anos	ES	Direito	Digitadora	> 10 anos	Não
	19	Não	3	5-10 anos	ME	Pedagogia	Núcleo De Acessibilidade Educacional	5-10 anos	Não

Fonte: Elaborada pelo autor (2020).

6.2 RESULTADOS OBTIDOS COM O EXERCÍCIO E COM O QUESTIONÁRIO

Cada usuário participante recebeu por e-mail um manual contendo as principais instruções para uso da DSL Cotas, incluindo conceitos sobre cada elemento da legislação. Adicionalmente, foi definido um exercício aberto, no qual, a tarefa proposta consistia em definir os requisitos da primeira versão da lei nº 12.711 (Figura 52).

Figura 52 – Versão proposta para exercício da DSL



Fonte: Elaborada pelo autor (2020).

Essa versão foi escolhida por ser mais simples se comparada as versões mais recentes, requerendo um tempo menor de dedicação ao exercício e adequando a complexidade para todos os grupos dos diferentes perfis de usuários.

Com o intuito de avaliar o exercício, todas as 20 respostas foram salvas imediatamente após o término no sistema de controle de versão `git`. Cada resposta foi salva em um `branch` disponível no repositório `https://github.com/spgroup/dsl-cotas/branches`.

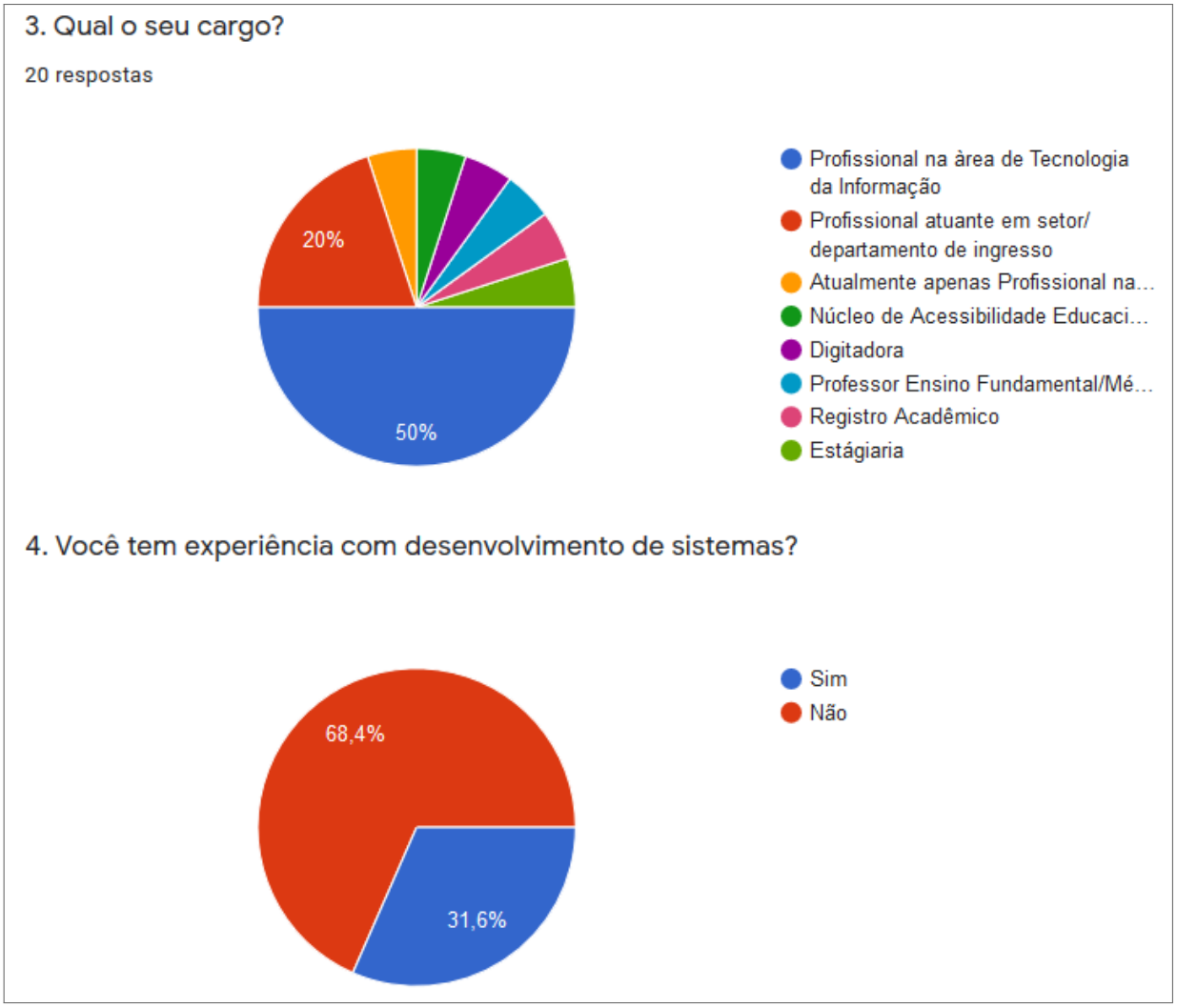
Após o exercício, os participantes foram convidados a responder ao questionário, no qual buscou-se obter as informações de cada perfil de usuário da DSL, assim como, entender as principais dificuldades encontradas por eles na utilização da linguagem.

As 5 (cinco) primeiras perguntas do questionário tiveram como objetivo levantar dados demográficos, tais como: nível de formação escolar, curso de formação acadêmica, cargo, tempo de experiência profissional e a informação se elas possuem experiência com desenvolvimento de sistemas.

O formulário *on-line* recebeu 20 respostas, das quais contou com a representatividade de profissionais da área de tecnologia da informação (50%) e profissionais atuantes em setor de ingresso de estudantes (20%), além de outros setores como pode ser observado no gráfico da

Figura 53. Desses profissionais, 68,4% indicaram não ter experiência prévia com desenvolvimento de sistemas, enquanto 31,6% informaram já terem experiência de desenvolvimento.

Figura 53 – Cargo e experiência de desenvolvimento

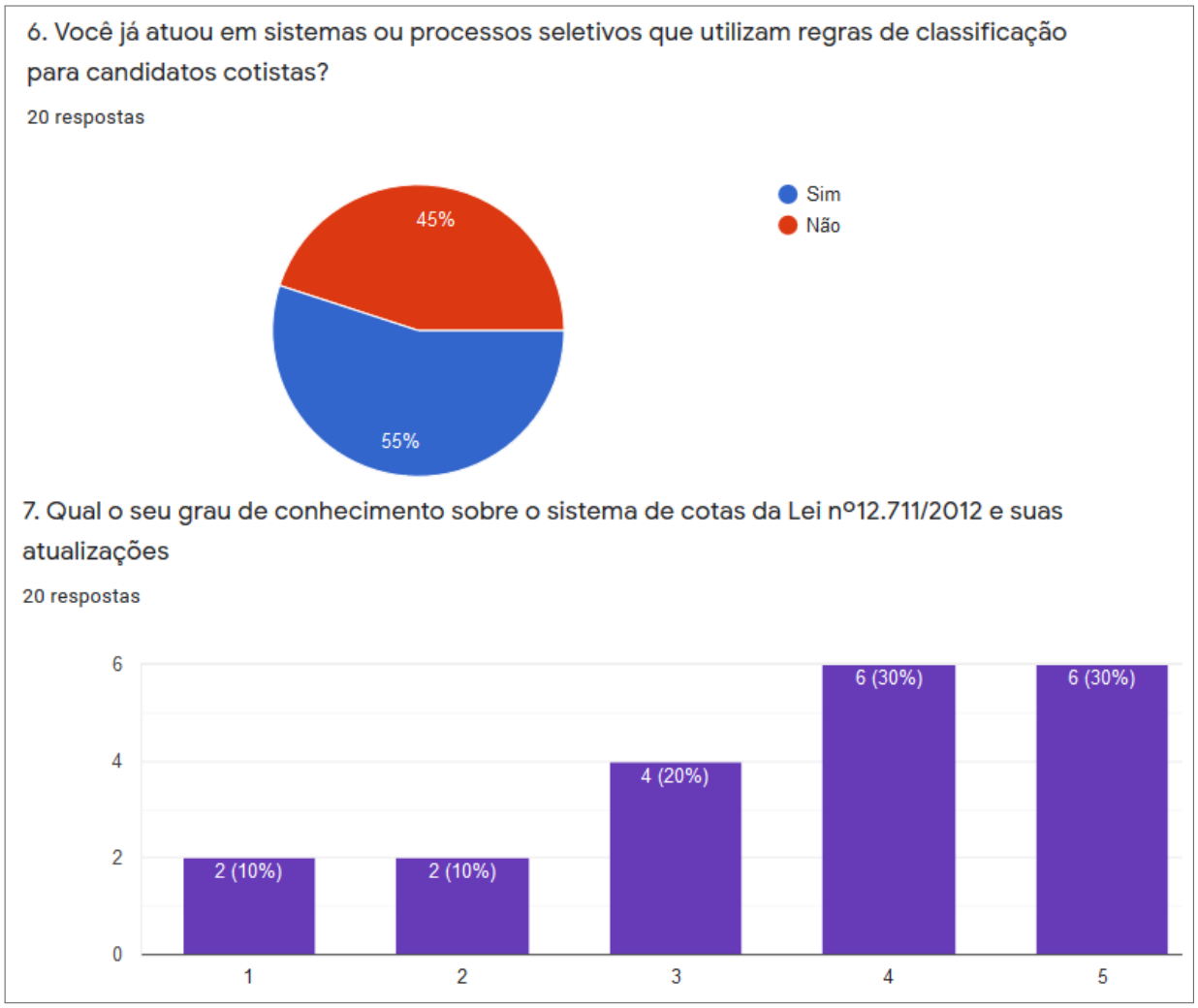


Fonte: Elaborada pelo autor (2020).

Com relação à pergunta "5. Quanto tempo de experiência profissional?", 50% dos usuários indicaram ter mais de 10 anos, 20% de 5 a 10 anos, 20% de 3 a 5 anos e 10% de 0 a 2 anos de experiência profissional.

Para levantar o conhecimento e a experiência sobre o sistema de cotas da rede de ensino federal foram submetidas as seguintes perguntas: "6. Você já atuou em sistemas ou processos seletivos que utilizam regras de classificação para candidatos cotistas?" e "7. Qual o seu grau de conhecimento sobre o sistema de cotas da Lei nº 12.711/2012 e suas atualizações?", cujas respostas podem ser observadas na Figura 54.

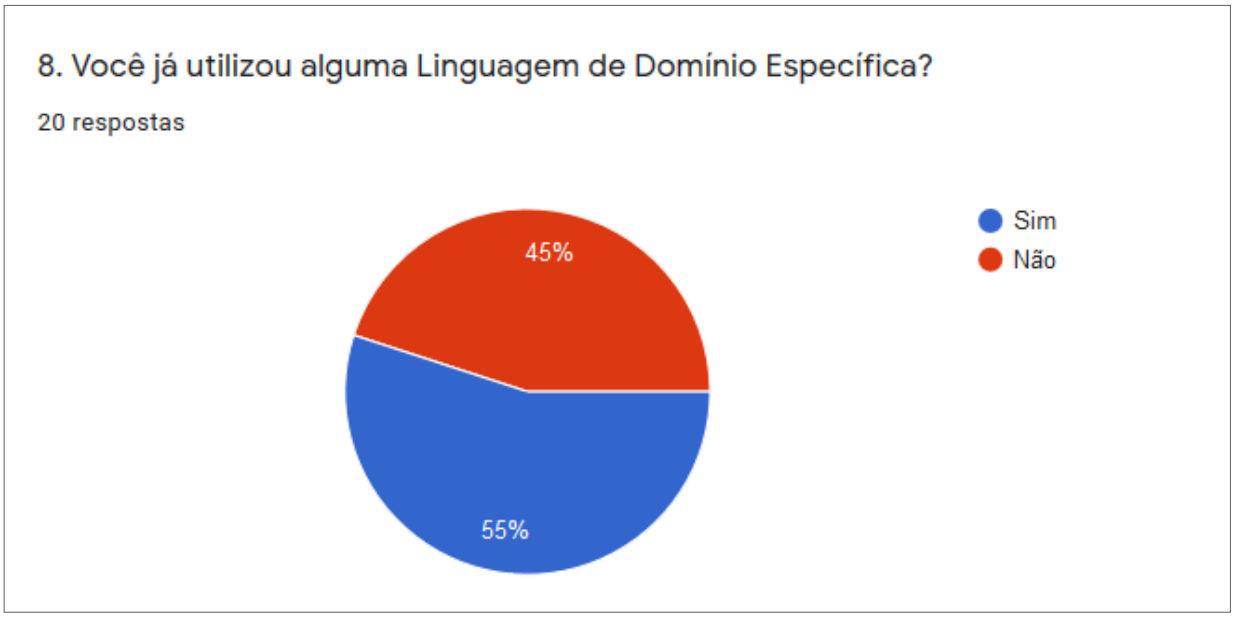
Figura 54 – Perfil de conhecimento e atuação em processos



Fonte: Elaborada pelo autor (2020).

Com o intuito de identificar se os usuários já tiveram contato com outras DSLs, a pergunta "8. Você já utilizou alguma Linguagem de Domínio Específica? Exemplos: HTML, Excel, Latex, SQL, etc.", mostrou que 55% dos usuários já tiveram contato com outras aplicações ou softwares que utilizam alguma DSL, enquanto os demais apontaram não conhecer nenhuma DSL (Figura 55).

Figura 55 – Experiência prévia com DSLs



Fonte: Elaborada pelo autor (2020).

Em continuidade, foram adicionadas as seguintes perguntas responsáveis pelo levantamento de dificuldades de uso com a DSL:

a) 9. De modo geral, qual o grau de dificuldade para execução do exercício proposto?;

b) 10. Considerando a sua resposta na pergunta anterior, qual(is) a(s) dificuldade(s) encontrada(s)?;

c) 11. Na sua avaliação quais as principais limitações da linguagem proposta?;

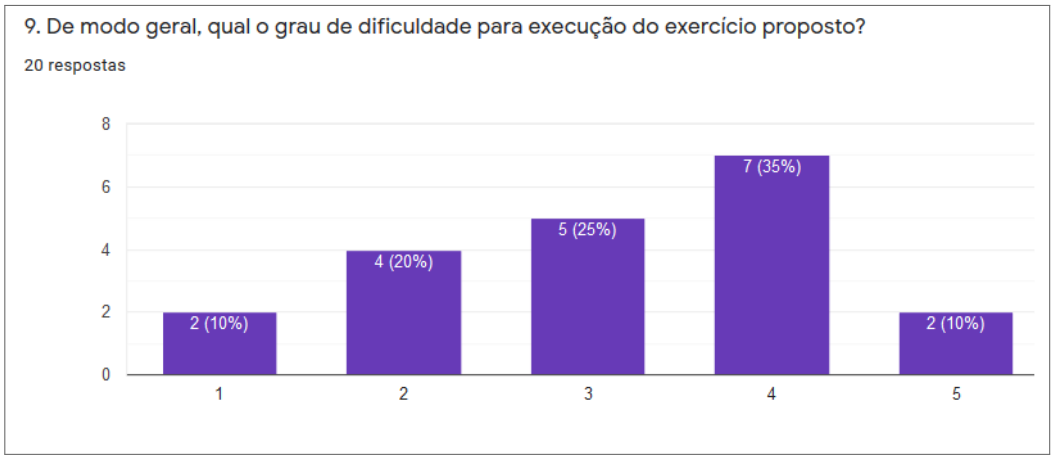
d) 12. Quais dos materiais abaixo você utilizou para execução do exercício?;

e) 13. No caso de a linguagem apresentar erros "destaques em vermelho" ou avisos "destaques em amarelo", as mensagens foram claras e ajudaram a resolver o(s) problema(s) apresentado(s)?;

f) 14. Quanto tempo, aproximadamente, você utilizou para executar o exercício proposto?.

Em linhas gerais, o grau de dificuldade de uso da DSL, em escala de 1 (difícil) e 5 (fácil), pode ser observado na Figura 56. O formulário completo foi disponibilizado no (APÊNDICE E).

Figura 56 – Dificuldade de uso e deficiências da DSL



Fonte: Elaborada pelo autor (2020).

Os demais resultados obtidos com o formulário e os exercícios são descritos com maiores detalhes nas próximas Seções.

Durante essa análise foram contabilizados os acertos em relação aos seguintes elementos da linguagem: distribuição de vagas (níveis e categorias criadas), configurações de percentuais (definição e utilização durante a distribuição), definição completa da ordem de prioridade (ordem indicada e quantidade de categorias presentes), quantidade de *errors* e *warnings* não resolvidos, e a presença de versão da legislação mais recente implementada opcionalmente por alguns usuários.

A seguir são apresentados os resultados da aplicação do exercício para cada grupo.

6.2.1 Resultados do Grupo DEV-ESP

Em relação à pergunta "9. De modo geral, qual o grau de dificuldade para execução do exercício proposto?", em uma escala de 0(difícil) e 5(fácil), nesse grupo de usuários todos indicaram a nota 4 e 5. Ademais, foram realizadas as seguintes considerações:

a) "Dúvidas de primeiro uso, do tipo -o que mesmo que eu devia digitar maiúsculo?-, -será que preciso mesmo digitar 50.0 ou só 50?-, entre outras do gênero. Como é execução de um exercício que é feito pela primeira vez, esse tipo de dúvida acaba não deixando totalmente fácil, pois eventualmente é necessário rever alguma instrução. Mas de resto, se já tiver esses detalhes na cabeça, a execução seria bem fácil (Usuário 20).";

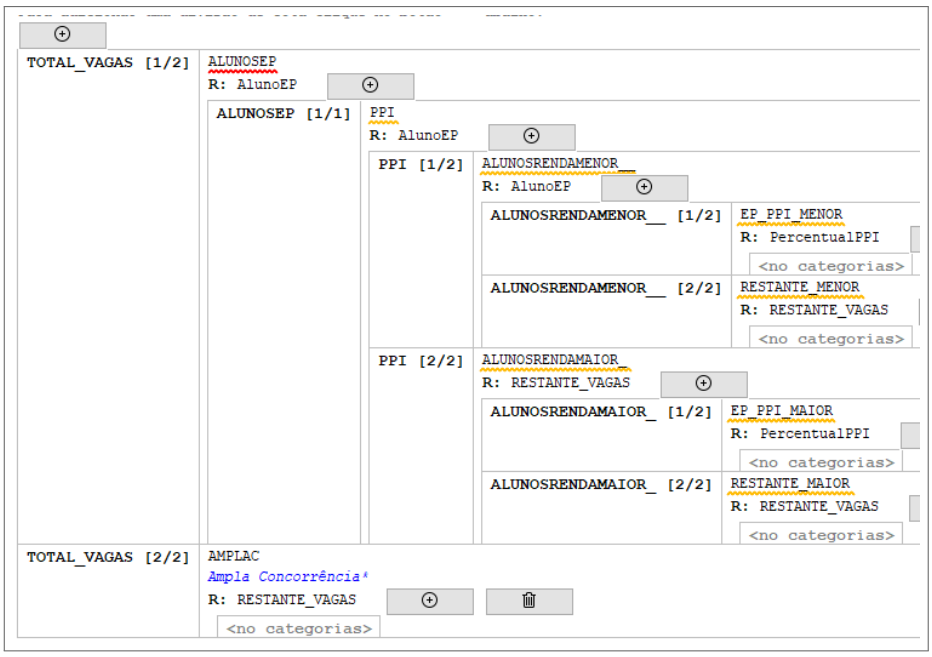
b) "A sequência de prioridades final onde se define quais candidatos se seleciona primeiro, foi um pouco mais difícil fazer a seleção... A dificuldade do último item pode ser melhorada com implementação ou instruções. No mais ficou bem interessante para configurar a árvore de decisão e percentuais aplicados (Usuário 3).".

Conforme relatado pelo Usuário 20, no primeiro contato com a DSL surgem algumas dúvidas sobre o formato de preenchimento dos percentuais e de siglas de categorias, o que pode levar a necessidade de reescrita de algumas instruções da linguagem. De modo geral, isso pode ser resolvido a medida que se acostuma com a linguagem.

Outro aspecto levantado pelo Usuário 3 relaciona-se com problemas durante a definição da ordem de prioridade, em que o comando de adição de novos itens na lista não estava claro o suficiente na linguagem, sendo necessário melhorar a usabilidade desse elemento.

Os exercícios foram finalizados entre 15 a 30 minutos, sendo que 2 (dois) desses usuários (Usuário 20 e 16) fizeram completamente a definição da árvore de distribuição, sem *errors* ou *warnings*. Os outros dois usuários (Usuário 3 e 12) não resolveram alguns erros e avisos da linguagem, um deles criou um nível a mais de distribuição, gerando uma árvore de distribuição incorreta (Figura 57).

Figura 57 – Exercício com erro na árvore de distribuição



Fonte: Dados da pesquisa (2020).

Com relação às respostas presentes na questão "11. Na sua avaliação quais as principais limitações da linguagem proposta?", 2 (dois) usuários informaram que: "As sugestões das mensagens informativas não são claras o suficiente (Usuários 16 e 20)", o que pode ter levado aos problemas descritos anteriormente.

Em relação a pergunta "13. No caso de a linguagem apresentar erros 'destaques em vermelho' ou avisos 'destaques em amarelo', as mensagens foram claras e ajudaram a resolver o(s) problema(s) apresentado(s)?", esse grupo indicou conseguir de maneira geral resolver os problemas tendo como base as mensagens apresentadas pela DSL, todos indicaram a pontuação de escala 4 e 5 (quatro e cinco), considerando a escala de 1(difícil) e 5(fácil).

Contudo, após o levantamento apresentado na Figura 58, foi possível identificar que os demais recursos e percentuais foram preenchidos corretamente. Destaca-se que o grupo em análise, possui bom conhecimento na área de domínio, além de ter familiaridade com ferramentas de desenvolvimento e outras DSLs, o que pode ter sido preponderante para que tenham indicado que a linguagem foi de fácil uso e entendimento.

Figura 58 – Quadro da análise do grupo DEV-ESP

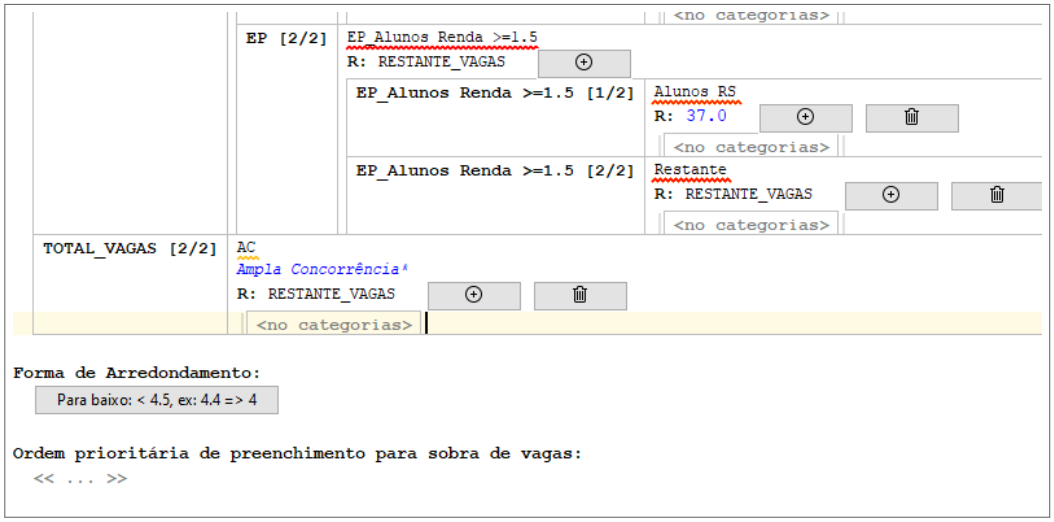
		Grupo DEV-ESP			
		Usuário 3	**Usuário 12**	**Usuário 16**	**Usuário 20**
Recursos utilizados	Quantidade de configurações[0-n]	4	2	1	4
	Configurações utilizadas[0-n]	3	2	0	4
	Níveis de distribuição da árvore de cotas[0-4]	4	5	4	4
	Categorias definidas[0-5]	5	5	5	5
	Categorias preenchidas na ordem de prioridade[0-5]	5	5	5	5
	Sequência correta na ordem de prioridade[0-5]	5	5	5	5
	Implementação de versão mais atualizada da legislação (não solicitado pelo exercício)[0|5]	0	0	0	0
Erros/Avisos encontrados	Quantidade de erros[0-n]	2	1	0	0
	Quantidade de warnings[0-n]	0	5	0	0
Tempo de exercício	-	De 15 min até 30 min	De 15 min até 30 min	Menos de 15 min	De 15 min até 30 min

Fonte: Elaborada pelo autor (2020).

6.2.2 Resultados do Grupo DEV-NESP

Nesse grupo, com relação à pergunta sobre o grau de dificuldade, os usuários apontaram os níveis 2 e 3 (dois e três), considerando a escala de 1(difícil) e 5(fácil). No exercício do Usuário 6 constaram vários erros não resolvidos na distribuição de vagas, em sua maioria em relação ao padrão de nomenclatura das siglas das categorias de cotas (Figura 59).

Figura 59 – Exercício do Usuário 11



Fonte: Dados da pesquisa (2020).

Adicionalmente à questão sobre o grau de dificuldade foram feitos os seguintes comentários:

a) "Foi difícil entender o funcionamento do sistema (Usuário 5)";

b) "No exercício não compreendi se era pra utilizar a a ordem de prioridade ou não, se segue o padrão de uma árvore ou teria que informar. A questão do arredondamento ficou um pouco confusa, depois que entendi que é apenas uma configuração caso utilizar números quebrados nos percentuais (Usuário 11)";

c) "Tive dificuldade em entender a proposta por não conhecer a lei específica. (Usuário 14)".

Portanto, nota-se a dificuldade de entendimento de questões relacionadas à legislação do sistema de cotas e, adicionalmente, algumas dificuldades sobre uso do elemento de ordem de prioridade da linguagem.

Em continuidade a essa análise, apresentam-se as considerações para a pergunta que trata sobre as limitações da linguagem: "Não observei a execução de uma emulação de processo em prática (Usuário 5)", "As sugestões das mensagens informativas não são claras o suficiente (Usuário 11)".

Considerando a limitação apontada pelo Usuário 5, observa-se a falta do *feedback* por parte da DSL para simulação da distribuição de vagas, uma vez que a função responsável por fazer os cálculos do quadro de vagas foi implementada apenas na API DSL Cotas.

Novamente foram apontados problemas nas mensagens geradas pela linguagem, conforme relato do Usuário 11. Esse usuário afirma que: "O vídeo didático poderia ser mais alto e com mais instruções de utilização.", indicando que são necessárias mais instruções no manual e no vídeo explicativo para conseguir melhorar o entendimento da linguagem.

Ademais, as seguintes sugestões foram levantadas por meio da pergunta número 15 do questionário:

> A linguagem auxilia a documentar o processo. Acredito que ela implemente a execução da coleta de dados. Espero que ela saiba ler os dados de várias bases diferentes, e não exija um tratamento nestes dados muito extenso, senão é talvez mais viável inserir diretamente os dados em uma base relacional e executar as ordenações necessárias (Usuário 5).

Tendo em vista os apontamentos descritos para o grupo, observa-se que a falta de entendimento nas regras de domínio dificulta o uso da linguagem (Usuários 11 e 14), no entanto, há maiores preocupações relacionadas à simulação e aos detalhes de implementação para o processamento final das regras (Usuário 5). Conforme o levantamento descrito na Figura 60, observou-se um tempo maior para o exercício do Usuário 5 (Mais de 30min).

Figura 60 – Quadro da análise do grupo DEV-NESP

		Grupo DEV-NESP		
		Usuário 5	**Usuário 11**	**Usuário 14**
Recursos utilizados	Quantidade de configurações[0-n]	3	1	1
	Configurações utilizadas[0-n]	3	1	1
	Níveis de distribuição da árvore de cotas[0-4]	4	4	4
	Categorias definidas[0-5]	5	5	5
	Categorias preenchidas na ordem de prioridade[0-5]	5	**0**	5
	Sequência correta na ordem de prioridade[0-5]	5	**0**	5
	Implementação de versão mais atualizada da legislação (não solicitado pelo exercício)[0|1]	0	0	0
Erros/Avisos encontrados	Quantidade de erros[0-n]	0	6	0
	Quantidade de warnings[0-n]	0	1	0
Tempo de exercício	-	Mais de 30 minutos	Menos de 15 minutos	De 15 minutos até 30 minutos

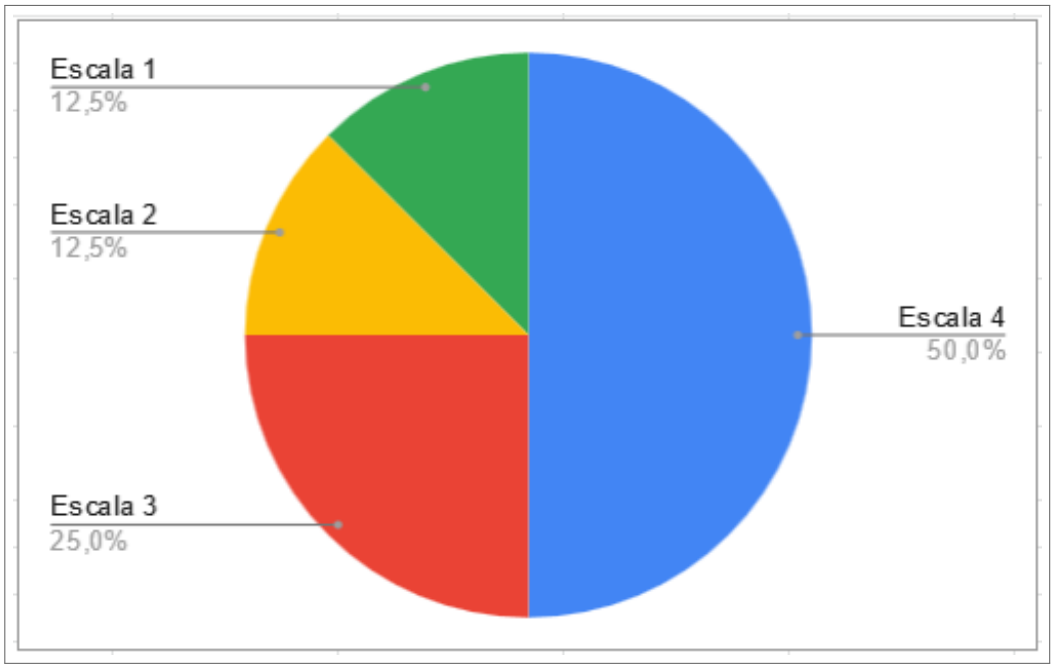
Fonte: Elaborada pelo autor (2020).

6.2.3 Resultados do Grupo NDEV-ESP

Nessa Subseção são analisados os dados dos participantes cujos perfis indicam maior proximidade com os usuários finais da DSL Cotas. Esses usuários, em sua maioria, já participaram da organização de processos seletivos com base no sistema de cotas, no entanto, não possuem nenhuma experiência com ferramentas de desenvolvimento de sistemas.

A Figura 61 demonstra o gráfico elaborado para as 8 (oito) respostas do grupo em relação à pergunta 11 do formulário, na qual em uma escala de 1 (difícil) a 5 (fácil) foi possível identificar que 50% dos usuários tiveram maior facilidade (escala 4), 25% indicaram a escala intermediária de dificuldade (escala 3) e os demais consideraram o uso da DSL como difícil (escalas 1 e 2).

Figura 61 – Gráfico de respostas com a escala de dificuldade



Fonte: Elaborado pelo autor (2020).

Considerando os comentários dos usuários sobre essa pergunta, destacam-se as seguintes observações:

a) "É necessário familiarizar-se com o ambiente proposto para o exercício. (Usuário 1)";

b) "Apenas dificuldade inicial para entender a lógica dos comandos. Logo após este entendimento, tornou-se fácil a utilização. O tutorial está muito claro, mas talvez um melhor suporte com menagens informativas no próprio ambiente, ao clicar nas etapas (Usuário 7)";

c) "Primeiro momento parece simples, mas como preencher corretamente, as siglas finais, fiquei um pouco confusa, e ao preencher um percentual no próximo só dar ctrl + backspace ou somente colocar RESTANTE_VAGAS... No final da ordem de como vai ser preenchido os restantes de vaga precisava ser um pouco mais claro, .. no geral é uma ótima ferramenta, mas necessitaria mais algumas aulas para ficar fera (Usuário 10)";

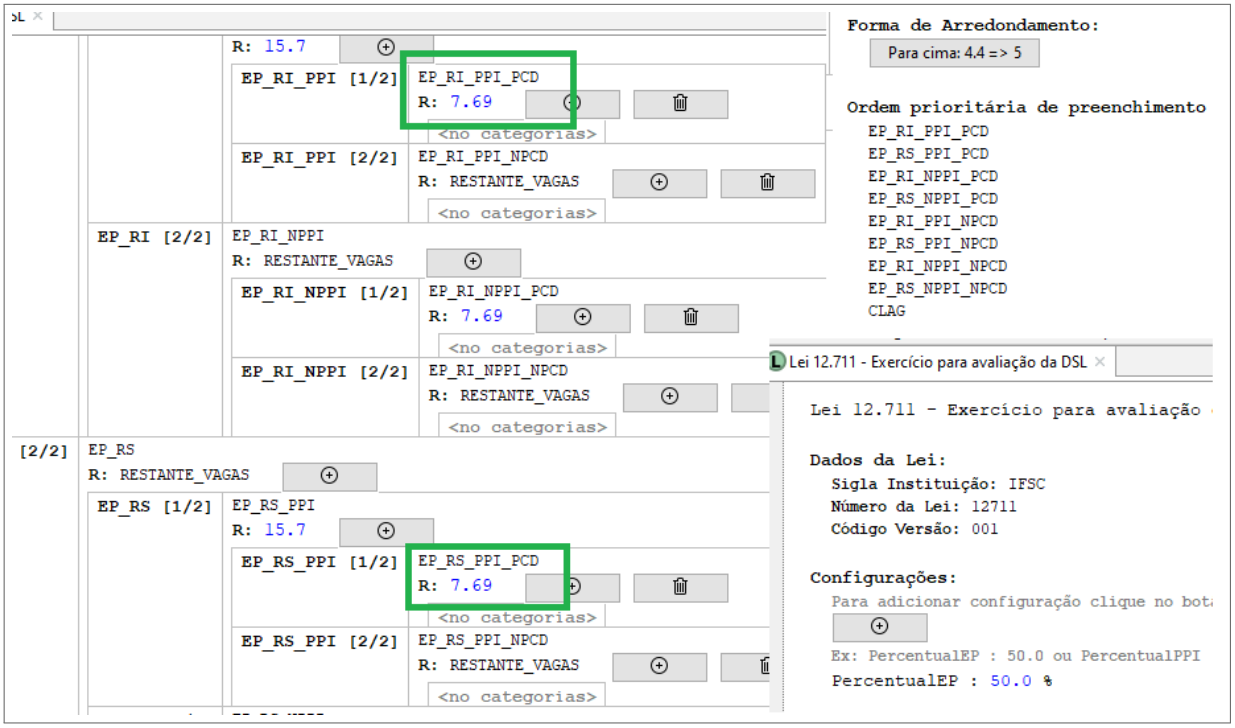
d) "Demorei um pouco para entender o funcionamento, a regra do sistema, mesmo após ver o vídeo explicativo. Após isso, interpretar a lei, ou o exercício, e aplicar no sistema também surtiu uma certa dificuldade - refiz 3 vezes até entender que estava de acordo com o proposto (Usuário 13)";

e) "O preenchimento na distribuição de vagas, sendo o entendimento do funcionamento das regras, após ter uma dificuldade com a variável pré existente de RESTANTE_VAGAS, e a relação com as configurações acima, o procedimento ficou mais claro (Usuário 18)".

Portanto, observa-se que esse grupo teve dificuldades relacionadas ao uso de comandos e regras da DSL, no que concerne à etapa de distribuição de vagas. Todos os usuários citados relatam que essas dificuldades foram resolvidas após algum tempo de uso da DSL Cotas.

Destaca-se que, 2 (dois) usuários do grupo (Usuário 4 e 7), implementaram a versão mais atualizada da legislação, na qual é composta de 5(cinco) níveis de distribuição e 9 (nove) categorias, incluindo a subdivisão para candidatos PCD (Figura 62). Essa versão não foi proposta pelo exercício e, mesmo assim, esses usuários avançaram na utilização da DSL Cotas opcionalmente. Isso pode indicar que, apesar das dificuldades encontradas, foi possível utilizar a DSL Cotas até para situações mais complexas não previstas pelo exercício.

Figura 62 – Exercício com a versão atualizada da lei



Fonte: Dados da pesquisa (2020).

Nesse grupo, o Usuário 4 levou mais de 1 (uma) hora para fazer a implementação do exercício, embora este também tenha utilizado a versão mais recente como base para descrição na DSL Cotas. Os demais, em sua maioria, conseguiram finalizar o exercício entre 15 e 30 minutos (Figura 63).

Sobre a pergunta "11. Na sua avaliação quais as principais limitações da linguagem proposta?", 4 (quatro) usuários apontaram que a principal limitação é: "As sugestões das mensagens informativas não são claras o suficiente". O Usuário 13 relatou que "O layout talvez não seja tão amigável para leigos em programação, o que dificulta um pouco para identificar onde clicar e o que digitar".

Figura 63 – Quadro da análise do grupo NDEV-ESP

		Grupo NDEV-ESP							
		Usr 1	Usr 4	Usr 7	Usr 10	Usr 13	Usr 15	Usr 17	Usr 18
Recursos utilizados	Quantidade de configurações[0-n]	2	1	1	1	4	2	1	2
	Configurações utilizadas[0-n]	0	1	1	1	4	0	1	2
	Níveis de distribuição da árvore de cotas[0-4]	4	5	5	3	4	3	4	4
	Categorias definidas[0-5]	5	9	9	3	5	3	5	5
	Categorias preenchidas na ordem de prioridade[0-5]	3	9	9	3	5	3	3	5
	Sequência correta na ordem de prioridade[0-5]	3	9	9	3	5	3	3	5
	Implementação de versão mais atualizada da legislação (não solicitado pelo exercício)[0|1]	0	1	1	0	0	0	0	0
Erros/Avisos encontrados	Quantidade de erros[0-n]	5	6	0	2	0	7	2	2
	Quantidade de warnings[0-n]	4	0	0	0	0	3	4	0
Tempo de exercício	-	De 15 min até 30 min	Mais de 1 hr	De 15 min até 30 min	De 15 min até 30 min	Mais de 30 min	De 15 min até 30 min	Mais de 30 min	De 15 min até 30 min

Fonte: Elaborada pelo autor (2020).

Assim como nos demais grupos de usuários, o grupo NDEV-ESP sugere melhorias nas mensagens informativas e na apresentação dos elementos da linguagem. Adicionalmente, algumas considerações foram observadas após análise do questionamento número "15. Comentários e Sugestões":

a) "A fim de evitar erros e facilitar a compreensão, o sistema poderia apresentar, ao operador, uma simulação da distribuição de vagas para cada cota. Ex: Inicialmente o sistema mostra na variável RESTANTE_VAGAS o valor 100 (ou outro valor). Após inserida a cota de ampla concorrência (CLAG) com percentagem de 50%, a variável RESTANTE_VAGAS apresenta o valor 50 e a CLAG apresenta o valor 50. E assim para todas as outras cotas. Ao final, seriam apresentadas as quantitativos de vagas para cada uma das cotas conforme os quadros dos editais de ingresso. (Usuário 1)";

b) "O ambiente parece bastante prático e dinâmico, com facilidade de adaptação às necessidades de cada tipo de processo seletivo. (Usuário 7)";

c) "O manual de instruções está bem explicado, porém, por ser uma linguagem nova, pode haver dificuldade na interpretação das instruções, como foi meu caso. Confundi algumas instruções simples pois fiquei focado tentando entender os demais comandos que não conhecia. Porém, saliento que após utilizar a primeira vez, o sistema é de fácil manuseio. (Usuário 17)";

d) "Não tem um entendimento claro sobre as configurações de percentual. Ex: quando nas configurações eu preencho um percentual de PPI - este é sobre o valor total ou sobre o valor da categoria que ele pertencer posteriormente. (Usuário 18)".

Esses comentários rementem às sugestões de melhorias, tais como: a necessidade de simulação prévia da distribuição e maior clareza sobre o modo de configuração dos percentuais das categorias. Ademais, apesar de dificuldades na interpretação e uso dos comandos (Usuário 17), observou-se que há facilidade de adaptação às necessidades de cada tipo de processo seletivo (Usuário 7).

Por fim, considera-se que os usuários do grupo conseguiram avançar no uso da DSL Cotas já no primeiro contato com a linguagem mesmo sem terem conhecimento em desenvolvimento de sistemas. Desse modo, infere-se que após terem maior familiaridade com a ferramenta, os usuários com domínio nas regras do sistema de cotas conseguem utilizar a DSL como meio de descrever as regras existentes na legislação.

6.2.4 Resultados do Grupo NDEV-NESP

Diferentemente dos grupos de usuários desenvolvedores e especialistas, que são os principais envolvidos no processo de compreensão e implementação da legislação, o grupo NDEV-NESP foi escolhido para verificar se o design da DSL Cotas é simples o suficiente para pessoas leigas no assunto. Com isso, na medida em que sintam necessidade, consigam compreender mais facilmente o funcionamento das regras de cotas com o uso da DSL, seja por interesse próprio ou pelo fato de, por exemplo, precisarem atuar em setores ou ações institucionais que envolvam processos de classificação de candidatos.

Com relação ao levantamento da cursos da formação acadêmica do grupo foram informados os seguintes cursos: Pedagogia, Licenciatura em Matemática, Gestão em TI e Direito. Os dados levantados sobre o cargo atual dos usuários foram: Professor Ensino Fundamental/Médio, Estagiário, TI, Digitador e Profissional do Núcleo de Acessibilidade Educacional.

Em relação ao tempo de execução do exercício, esse grupo apresentou resultado variado. Os Usuários 6 e 19 levaram mais de 30 minutos, os Usuários 2 e 9 levaram de 15 até 30 minutos e o Usuário 8 levou menos de 15 minutos (Figura 64).

Figura 64 – Quadro da análise do grupo NDEV-NESP

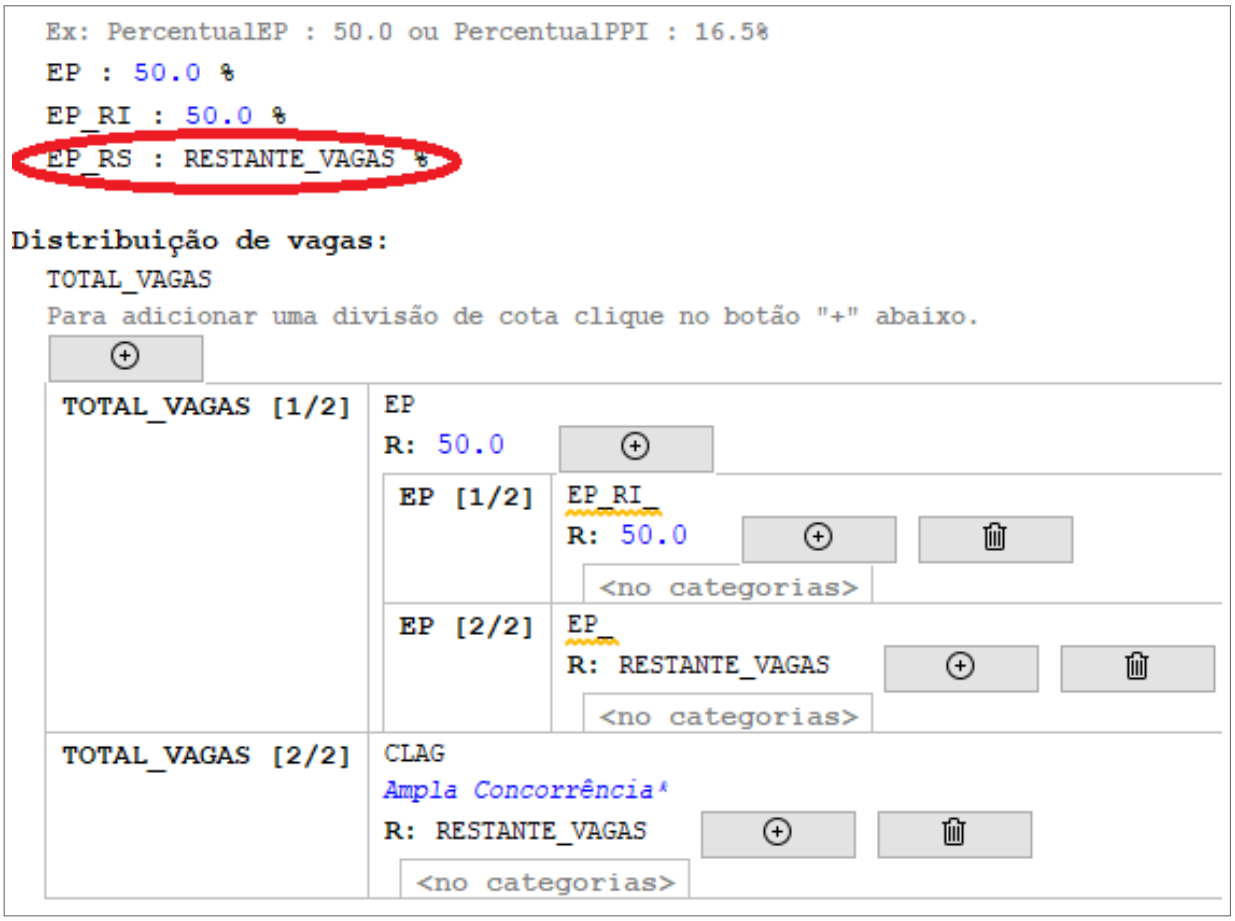
		Grupo NDEV-NESP				
		Usr 2	**Usr 6**	**Usr 8**	**Usr 9**	**Usr 19**
Recursos utilizados	Quantidade de configurações[0-n]	3	1	2	1	5
	Configurações utilizadas[0-n]	0	1	1	1	4
	Níveis de distribuição da árvore de cotas[0-4]	3	4	2	4	4
	Categorias definidas[0-5]	3	6	2	3	5
	Categorias preenchidas na ordem de prioridade[0-5]	2	5	0	3	5
	Sequência correta na ordem de prioridade[0-5]	1	4	0	3	5
	Implementação de versão mais atualizada da legislação (não solicitado pelo exercício)[0|1]	0	0	**0**	0	0
Erros/Avisos encontrados	Quantidade de erros[0-n]	2	7	4	6	0
	Quantidade de warnings[0-n]	2	1	0	0	2
Tempo de exercício	-	De 15 minutos até 30 minutos	Mais de 30 minutos	Menos de 15 minutos	De 15 minutos até 30 minutos	Mais de 30 minutos

Fonte: Elaborada pelo autor (2020).

Após o levantamento, foi possível identificar que 3 (três) dos 5 (cinco) usuários, não implementaram corretamente a quantidade de níveis de distribuição proposta pelo exercício, os Usuários 2 e 8 implementaram, respectivamente, apenas 3 (três) e 2 (dois) níveis da árvore. O Usuário 6 criou um nível extra sem subdivisões o que gerou um número de 6 categorias. Os problemas criados durante a definição da distribuição ocasionaram no aumento da contabilização de erros para esse grupo.

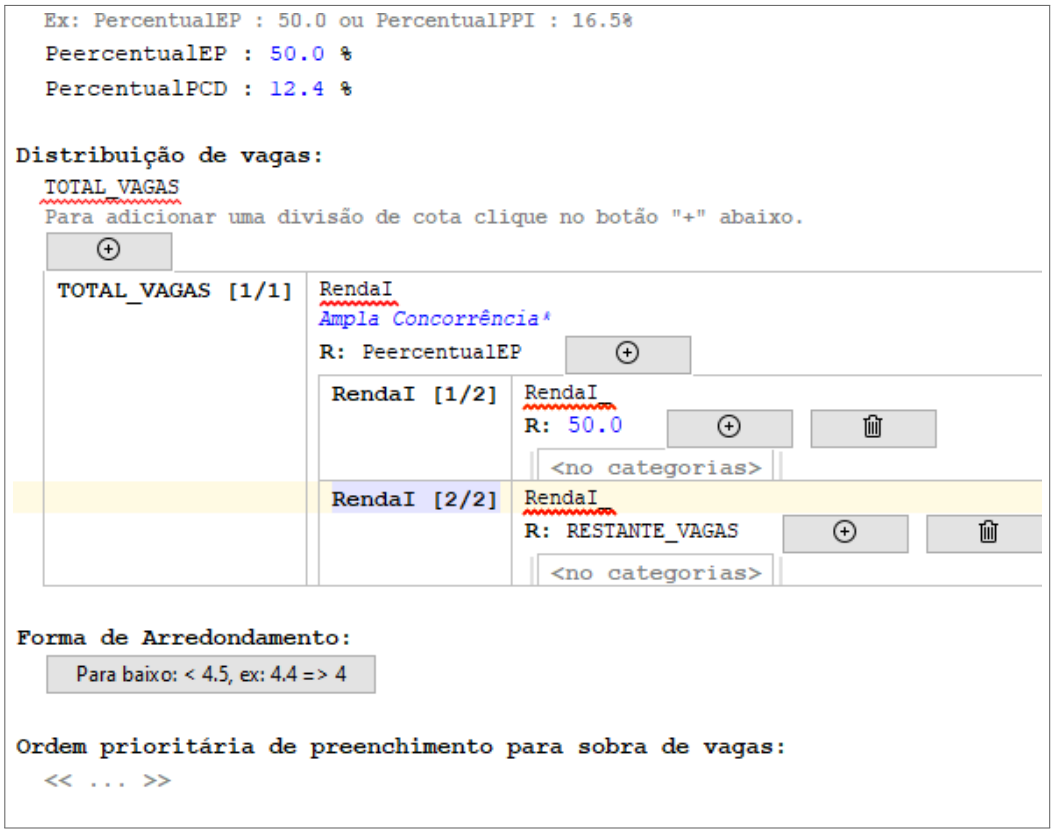
Destaca-se o caso do Usuário 2, o qual utilizou a constante RESTANTE_VAGAS fora do contexto de distribuição, o que levou o pesquisador a descobrir uma falha durante a checagem de escopo da DSL Cotas (Figura 65). Ademais, o Usuário 8 acabou desistindo da correção dos erros apontados pela DSL Cotas no segundo nível de distribuição, indicando a finalização do exercício abaixo de 15 minutos (Figura 66).

Figura 65 – Falha nas restrições de escopo



Fonte: Dados da pesquisa (2020).

Figura 66 – Problemas na definição da distribuição



Fonte: Dados da pesquisa (2020).

Para o levantamento sobre o grau de dificuldade nas escalas 1 (difícil) até 5 (fácil) observa-se que os Usuários: 2, 6 e 19 apontaram a escala 3, enquanto os usuários 8 e 9 apontaram a escala 4 e 2, respectivamente. O que pode indicar que na maioria dos casos houve dificuldade para entendimento e uso da DSL Cotas.

Em continuidade à análise das dificuldades, consideram-se as seguintes observações dos usuários:

a) "Uma das dificuldades é por ser o primeiro contato com a ferramenta e não ter domínio sobre a mesma. Depois, fiquei com dúvida no preenchimento dos critérios: Eles serão baseados na lei de cotas, ou seja, os percentuais de destinação para cada vaga já estão estabelecidos, fica a critério de quem for criar a árvore a ordem de destino das vagas então? Fiquei com dúvida também se as abreviações deveriam ser conforme o que já consta nos editais e como os inscritos se cadastram no sistema de ingresso, ou se isso não faz diferença. (Usuário 2)";

b) "Como usuário iniciante, o guia de instruções estava com poucas informações para execução da tarefa. (Usuário 6)";

c) "Como pessoa que não exerce atividade nesse segmento, achei pouco intuitivo. A parte para definição de ordem prioritária, aparentava existirem campos repetidos, pois referente

aos alunos RI e RS não encontrei diferenciação entre os 37% e o restante de vagas (dá a ideia de existirem apenas 3 campos - Alunos RI - Alunos RS - AC). (Usuário 9)";

d) "Para organizar as informações no exercício precisei rever o vídeo com as instruções e após a segunda visualização do material com exemplos consegui usar os recursos mais facilmente. (Usuário 19)".

Desse modo, pontuam-se algumas das dificuldades apresentadas por esse grupo, tais como: dúvidas sobre o preenchimento correto dos percentuais e siglas das categorias (Usuário 2), falta de instruções com passos mais detalhados (Usuário 6), problemas durante a definição das categorias e sua utilização na ordem de prioridade (Usuário 9) e a necessidade de várias consultas no manual e ao vídeo explicativo da DSL (Usuário 19).

Com relação ao levantamento sobre as deficiências da linguagem, os Usuários 2 e 19 marcaram a opção "O tempo de aprendizagem da linguagem é extenso", enquanto os demais usuários apontaram que "As sugestões das mensagens informativas não são claras o suficiente".

Em sequência a esse levantamento, os usuários apontaram como comentários e sugestões:

a) "Achei o sistema em certa medida intuitivo e de fácil manuseio, mas tive dificuldades com o tamanho da fonte das palavras (estava pequeno e não foi possível aumentar usando ctrl+). Acredito que com critérios bem estabelecidos (quais abreviações usar, etc) e um tempo maior de uso a ferramenta será de grande utilidade (Usuário 2)";

b) "O guia para 1ª utilização do sistema poderia ser mais detalhado; mensagens de erro poderiam ser exibidas em caixas de texto; tamanho das letras nas caixas de opções, muito pequenas. (Usuário 6)";

c) "Nas mensagens de erro ou aviso, constar exemplo correto de preenchimento. (Usuário 8)".

Por fim, notou-se maior dificuldade de entendimento das mensagens e nos recursos de *feedback* ao usuário, se comparado aos grupos com conhecimento na área de domínio e/ou desenvolvimento de sistemas. Na Subseção 6.2.5 é descrito o resumo comparativo entre os grupos analisados.

6.2.5 Comparativo entre os grupos de análise

Em relação aos critérios estabelecidos para a análise, considerando o levantamento apresentado pelas Figuras 58, 60, 63 e 64, observou-se que o grupo DEV-ESP teve mais facilidade de entendimento e utilização dos recursos da DSL, uma vez que todos os usuários conseguiram definir as regras propostas pelo exercício, além das soluções apresentarem poucos *errors* e *warnings*. Esse fato reforça que: "o uso de DSLs com entendimento do domínio, deixa o pensamento ser expresso de maneira mais clara quando o código escrito não está repleto de detalhes de implementação" (VOELTER et al., 2013, p.41, tradução nossa).

Em segundo lugar, considerando a correta implementação das regras propostas, no grupo NDEV-ESP apenas 2 (dois) dos 8 (oito) usuários não conseguiram configurar a distribuição completa, no entanto, 6 (seis) usuários fizeram as definições conforme o exercício, sendo que 2 (dois) deles avançaram no desenvolvimento da versão mais recente e complexa da legislação, contemplando candidatos inscritos na categoria PCD.

O fato dos usuários NDEV-ESP terem conseguido atender uma legislação diferente da proposta, pode se relacionar com um dos benefícios de DSLs:

> O uso de DSLs específicas de domínio, podem parecer, a primeiro momento, difícil de se justificar, porém essas DSLs são normalmente atreladas ao *know-how* do negócio, provendo um meio de descrever conhecimento de maneira formal, organizada e sustentável. (VOELTER et al., 2013, p.43, tradução nossa).

Em continuidade a esse comparativo, o grupo DEV-NESP ficou em terceiro lugar no que diz respeito ao entendimento sobre os recursos utilizados, esses usuários apontaram no questionário um grau de dificuldade elevado, o que foi agravado pelo fato de desconhecerem a área de domínio. Contudo, apesar das dificuldades, na maioria dos casos, todos os recursos foram utilizados sem que fossem gerados muitos erros ou avisos.

A análise do grupo NDEV-NESP mostra que há potencial de uso da DSL, no entanto, destaca-se que há necessidade de mais explicações sobre as regras de domínio, assim como de capacitação para uso na linguagem. Isso se justifica, pelo fato de que esse grupo apresentou maior quantidade de erros além de alguns exercícios terem sido entregues de forma incompleta.

Por fim, a Subseção 6.2.6 relaciona algumas melhorias realizadas na DSL Cotas a fim de melhorar o entendimento sobre as regras e as instruções da linguagem.

6.2.6 Mudanças resultantes da avaliação

Considerando as principais dificuldades relatadas pelos usuários da DSL Cotas, nessa Subseção são descritas as melhorias implementadas na linguagem que foram selecionadas com o objetivo de reduzir as lacunas de entendimento sobre a distribuição das vagas e também melhorar a clareza das mensagens de erros e avisos aos usuários.

A primeira melhoria é a responsável pelo tratamento das sugestões apontadas pelos usuários do grupo DEV-ESP e NDEV-ESP, os quais relataram a necessidade de visualização prévia de simulação das vagas. Para tanto, foi adicionado um novo elemento do tipo `behavior`, `Distribuicao_Behavior` no qual foram inseridos métodos para varrer a AST e fazer os cálculos do quadro de vagas seguindo as definições preenchidas pelo usuário da DSL (Código Fonte 9).

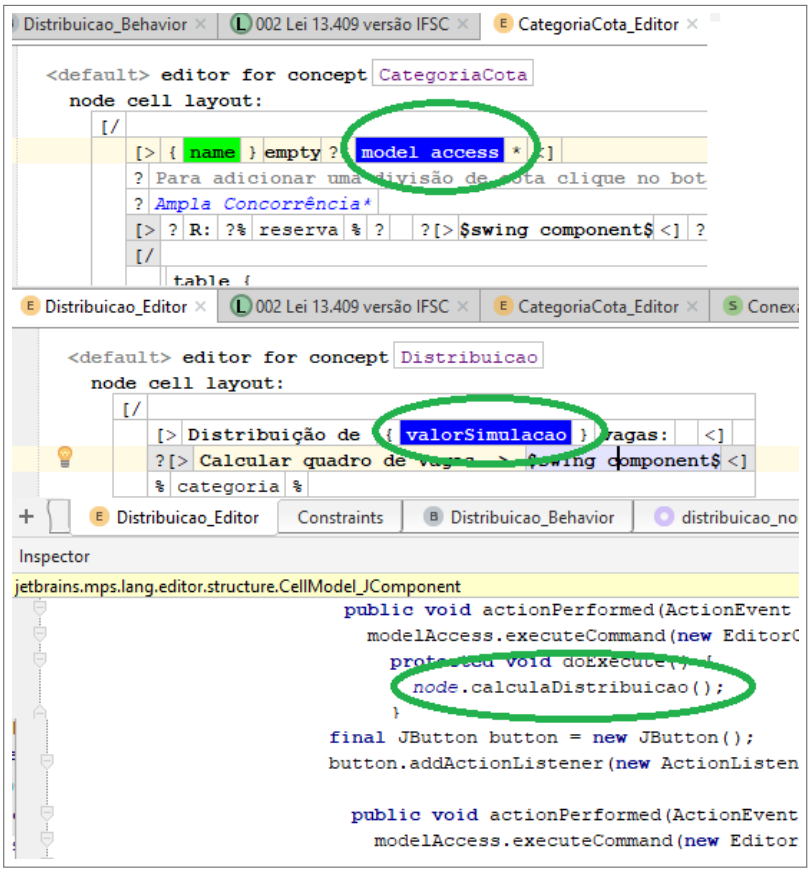
Código Fonte 9 – Behavior para cálculo do quadro de vagas

```
1  concept behavior Distribuicao {

3    public void calculaDistribuicao() {
     this.ancestor<concept = LeiDeCota>.categoriasSimuladas.clear;
5
     if (this.categoria.categorias != null && !this.categoria.categorias.isEmpty) {
7      node<LeiDeCota> lei = this.ancestor<concept = LeiDeCota>;
       Int arredondamentoParaCima = lei.arredondamento;
9      this.categoria.numeroVagas = 0;
       calculaVagasCategoria(this.categoria, this.valorSimulacao,
           arredondamentoParaCima);
11
     ...
13
     private void calculaVagasCategoria(node<CategoriaCota> categoria, int
         quantidadeVagas, Int formaArredondamento) {
15
       if (categoria.parent.isInstanceOf(Distribuicao)) {
17       categoria.numeroVagas = quantidadeVagas;

19     } else if (categoria.parent.isInstanceOf(CategoriaCota)) {

21       Double percentual = getPercentualReserva(categoria.ancestor<concept =
             LeiDeCota>, categoria.reserva);

23       if (percentual != null) {
           categoria.numeroVagas = getVagasArredondamento(quantidadeVagas, percentual,
               formaArredondamento);
25       } else {
           if (categoria.reserva != null && categoria.reserva.getDetailedPresentation()
               .equals("RESTANTE_VAGAS")) {
27           categoria.numeroVagas = categoria.parent:CategoriaCota.numeroVagas -
                 categoria.parent:CategoriaCota.somaVagas();
       ...
```

O método `calculaDistribuicao` (Linha 3) acessa a `CategoriaCota` raiz, o valor de simulação desejado e a forma de arredondamento definida pelo usuário, e em sequência, por meio do método recursivo `calculaVagasCategoria` (Linha 14) são realizadas iterações em todas as categorias subsequentes da distribuição, calculando as vagas conforme os respectivos percentuais (Linha 24) ou fazendo o cálculo de soma das vagas para a constante `RESTANTE_VAGAS` (Linha 27).

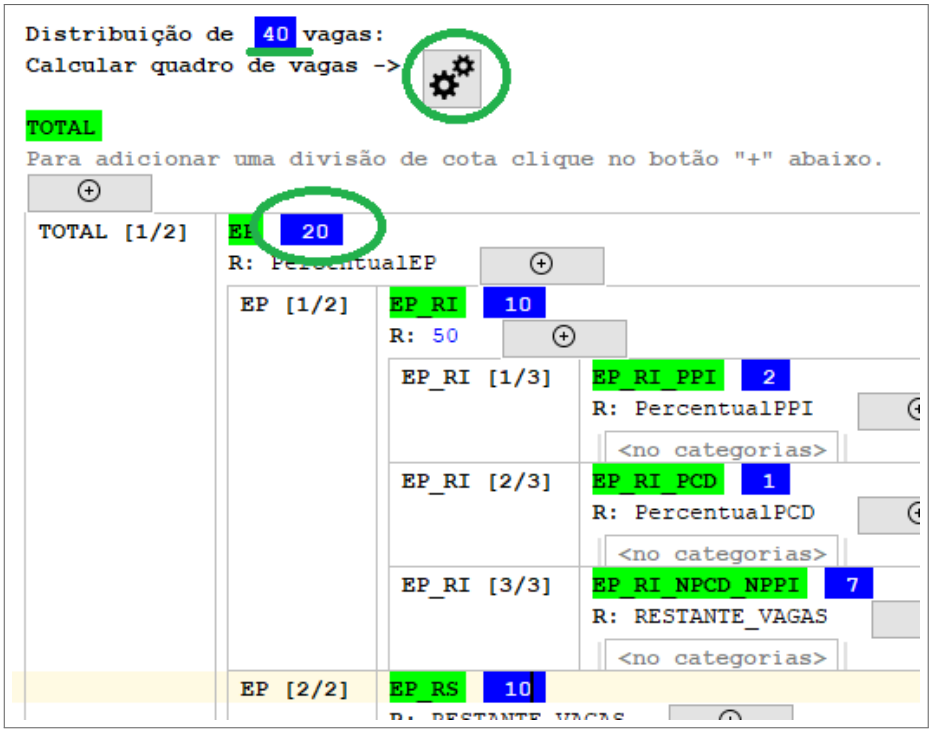
Os números de vagas para cada categoria são armazenados em um novo atributo do conceito, nomeado de `numeroVagas`, esses valores são atualizados sempre que o usuário aciona a função de simulação na DSL ou no caso da forma de arredondamento ser alterada. Desse modo, foi possível adicionar novos elementos no editor dos conceitos `CategoriaCota` e `Distribuicao` para apresentar o valor calculado, além de gerar uma listagem do quadro de vagas (Figuras 67, 68 e 69).

Figura 67 – Editores alterados para simulação de vagas



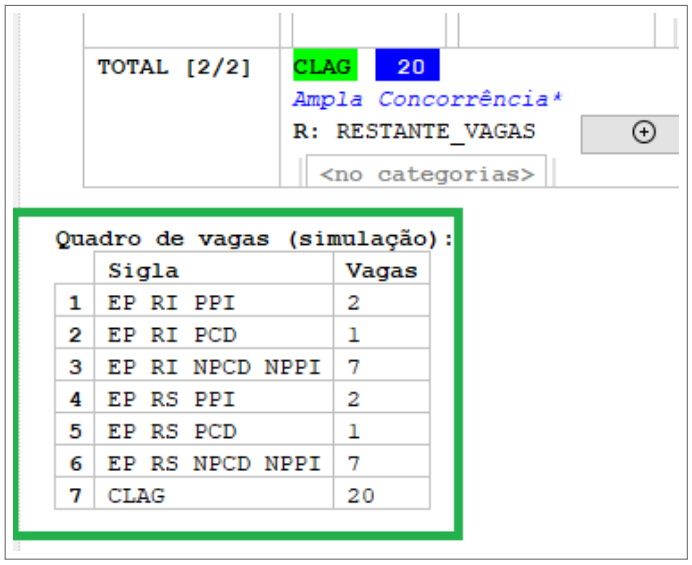
Fonte: Elaborado pelo autor (2020).

Figura 68 – Simulação de vagas durante a definição da distribuição



Fonte: Elaborado pelo autor (2020).

Figura 69 – Simulação do quadro de vagas

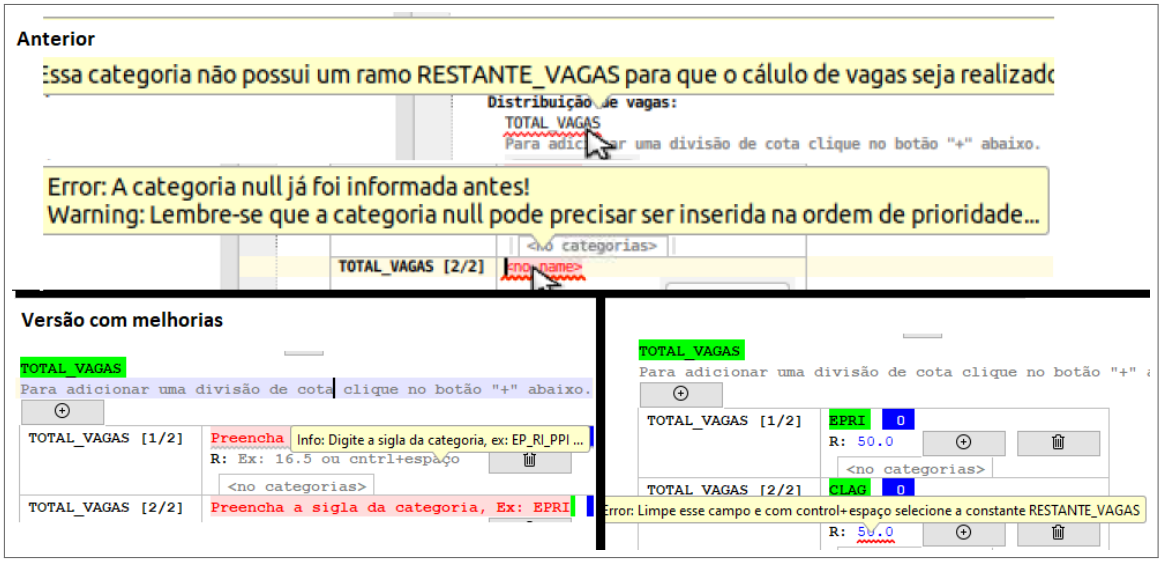


Fonte: Elaborado pelo autor (2020).

A visualização prévia dos cálculos pode auxiliar no entendimento dos requisitos de distribuição de vagas para todos os grupos de usuários analisados, possibilitando que tenham uma simulação imediata dos elementos da linguagem, além de auxiliar no caso de dúvidas sobre como os percentuais são aplicados no contexto da árvore de distribuição.

Com relação às dificuldades de entendimento das mensagens de erro e dos avisos apresentados pela linguagem, foram alteradas as instruções para incluir exemplos de formato de preenchimento, além de tratamentos para priorizar as mensagens exibidas de modo que não fossem geradas inconsistências antecipadamente, o que pode contribuir para confundir o usuário sobre o momento correto da sua resolução (Figura 70).

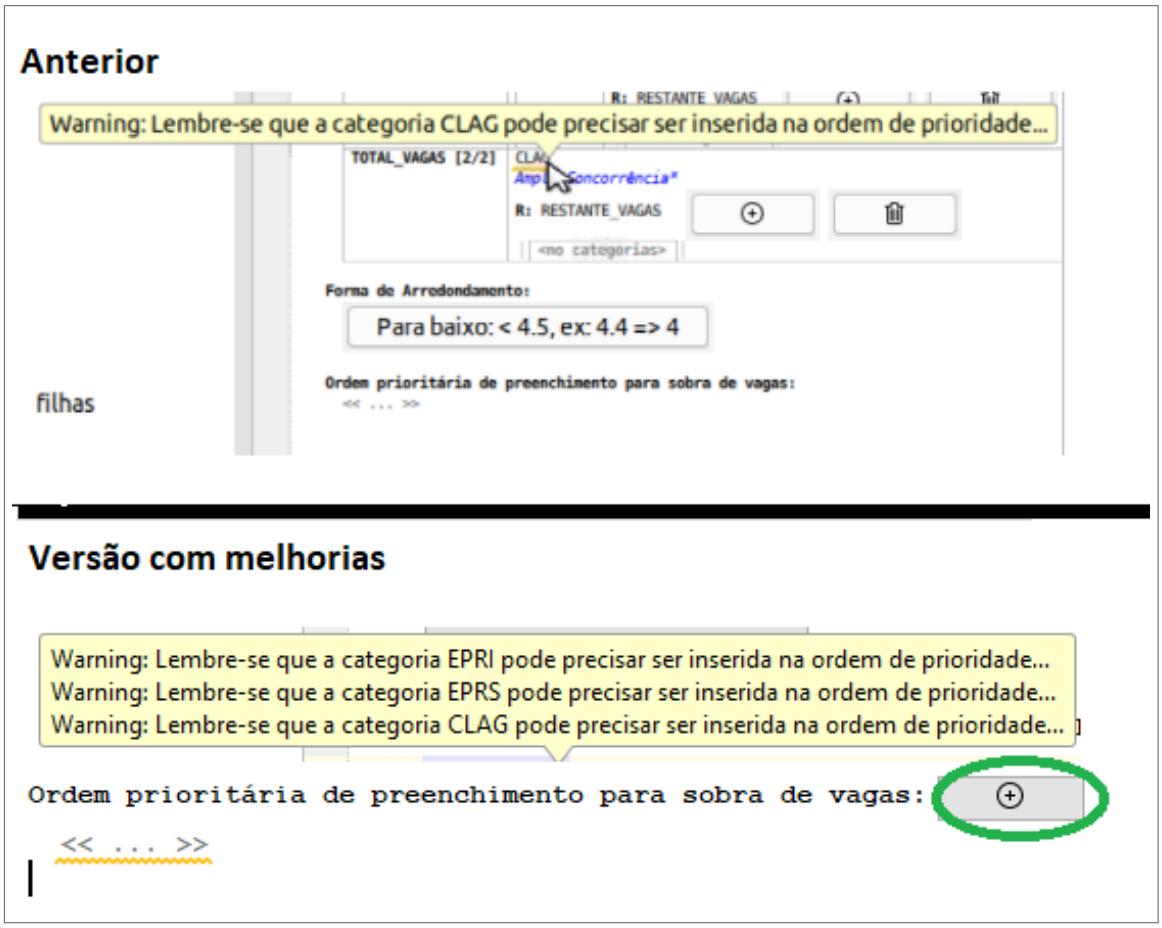
Figura 70 – Melhorias nas instruções e mensagens



Fonte: Elaborado pelo autor (2020).

No que diz respeito às dificuldades dos usuários para a seleção das categorias na seção `OrdemPrioridade`, foi alterado o editor para permitir adicionar novas categorias por meio do componente `JButton`, o qual facilita a seleção da categoria pelo usuário, sem precisar utilizar apenas o teclado para novas inclusões (Figura 71). Destaca-se também, a modificação nas mensagens instrutivas sobre o preenchimento da ordem de prioridade, que deixaram de ser apresentadas em cada categoria da árvore de distribuição, para serem requisitadas apenas no momento de definição da ordem de prioridade.

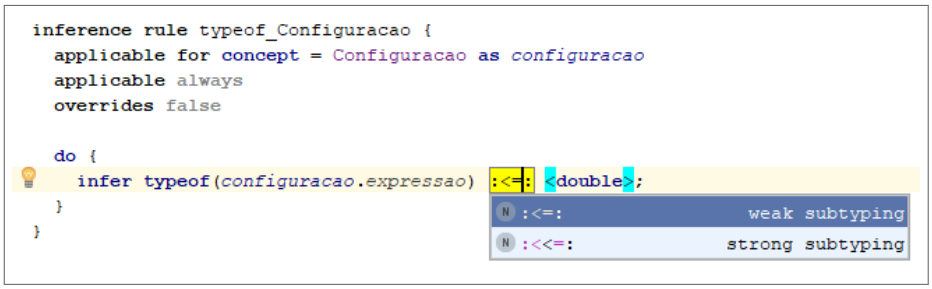
Figura 71 – Melhorias na seção Ordem de Prioridade



Fonte: Elaborado pelo autor (2020).

Para tratar das dificuldades relatadas sobre o formato de preenchimento dos percentuais, foi alterada a regra de inferência dos elementos `Configuracao` e `CategoriaCota` para aceitar tanto valores inteiros como valores fracionados, evitando que o usuário tenha que preencher por exemplo "50.0". Isso foi possível pela comparação de tipagem fraca (*weak subtype*) ao invés da comparação exata de tipos (*strong subtype*).

Figura 72 – Regra de inferência com *weak sybtype*



Fonte: Elaborado pelo autor (2020).

As alterações descritas foram realizadas tendo como base as principais dificuldades apontadas pelos usuários da DSL. Desse modo, a versão final pode agregar valor ao presente estudo no que diz respeito ao uso de DSL como ferramenta de melhoria na comunicação entre os envolvidos, reduzindo eventuais incompreensões no uso da linguagem.

Na próxima Seção serão apresentados os resultados da análise com relação aos testes da API DSL Cotas.

6.3 RESULTADOS OBTIDOS COM A API DSL COTAS

Nessa Seção são descritos os resultados provenientes dos testes da API DSL Cotas, nos quais tiveram como objetivo principal a comparação do histórico de processos seletivos do IFSC com o resultado gerado pela API, de modo a verificar a conformidade de seus resultados em relação às legislações vigentes e/ou mais antigas.

Para a realização dos testes da API, o autor da presente pesquisa replicou as regras e as siglas equivalentes da primeira e da versão mais recente da legislação na DSL Cotas. Foi necessário inicializar a lista de candidatos na base de dados com os parâmetros iniciais antes da classificação, de modo a simular o processo de primeira chamada ocorrido na época.

Essa análise contou com a seleção aleatória de 16 processos seletivos, incluindo 403 cursos do tipo: técnico integrado, concomitante, subsequente, proeja e graduação. Nesse contexto, foram considerados os resultados de 13494 candidatos no período de 2013 até 2020, nos quais foi possível testar 2 (duas) das versões já processadas pelo sistema de ingresso, além das demais alterações de código ocorridas no período.

Não foi possível fazer o comparativo com a versão intermediária descrita na Subseção 2.2.2 do Capítulo 2, uma vez que não houve processamento de resultados na época, pois o código precisou ser atualizado com a versão mais recente disponibilizada pelo MEC, antes da sua utilização nos processos seletivos.

Como resultado dos testes foi gerado um relatório contendo as colunas, "Versão de lei utilizada", "Edital", "Identificador do Curso", "Número de vagas", "Código de Inscrição", "Classificação", "Categoria esperada" e "Resultado da conferência". Para cada candidato foi acionado o *endpoint* `aprovaCandidatos` em classe de testes do `JUnit`, na qual foram reali-

zadas requisições HTTP passando a lista de inicial de candidatos inscritos, sem a situação de classificação.

A listagem resultante da API passou por comparação de todas as siglas de situação de classificação conforme o sistema de cotas, sendo possível retornar e comparar a sigla de situação original presente na base do sistema de ingresso. Desse modo, foi verificado que 81 dos 403 cursos apresentaram divergências nas listas de classificação de candidatos.

Essas divergências foram marcadas em 297 candidatos dos 13494 registros utilizados. Para cada curso com divergência foi realizada uma análise manual que teve como objetivo identificar o problema, o motivo e uma possível solução. Essas divergências foram agrupadas nas seguintes *issues* cadastradas no repositório `https://github.com/estrazulas/dsl-cotas-gen/issues`:

a) **Candidato sem situação de classificação**: Situação encontrada em 2 (dois) cursos, o problema apresenta alguns candidatos sem sigla de categoria na lista resultante com o uso da API. Nos 2 (dois) casos, o motivo da divergência aponta para o fato de o teste ter sido realizado apenas em candidatos de primeira chamada, sendo que os candidatos com problema foram convocados posteriormente pelo sistema de ingresso. Não sendo um problema a ser solucionado pela API, uma vez que em chamadas posteriores as situações seriam recalculadas;

b) **Cursos com candidatos em rechamada**: Em 31 dos cursos testados foram encontrados candidatos em situação de inscrição "REC" ou re-chamados, no entanto, a seleção de candidatos para aplicação do testes apenas considerou a busca por candidatos de primeira chamada e em situação "CLA", sigla utilizada antes da etapa de aprovação e atribuição de categorias. Os candidatos re-chamados faziam parte de uma regra de negócio antiga do sistema, que dava a oportunidade de serem reconvocados para matrícula em chamadas posteriores, e por esse motivo não entraram no filtro de comparação de candidatos em primeira chamada;

c) **Candidato cotista convocado como ampla concorrência**: Em apenas um curso foi encontrado um candidato de inscrição por cotas que na API foi marcado para a categoria correta, no entanto, no sistema de ingresso consta com a categoria de ampla concorrência. Após análise manual identificou-se que seguindo as regras de processamento, o candidato da posição 18 deveria ter sido convocado como cotista. No entanto, a causa dessa divergência é desconhecida, sendo possível que na época esse candidato tenha sido alterado manualmente em base de dados;

d) **Candidato da ampla concorrência classificado como cotista**: A divergência com maior número de casos (42 cursos do SISU). Após análise no processamento foi identificado que os candidatos inscritos como cotistas com pontuação superior aos de candidatos da ampla concorrência (CLAG) foram aprovados pelo SISU como cotistas, quando em todos os demais processos seletivos do sistema o funcionamento sugere que os pri-

meiros colocados sejam selecionados e classificados pela categoria CLAG. Em síntese, o processamento foi diferente ao adotado pelo SISU, o que não pode ser resolvido no contexto da presente pesquisa.

Destaca-se que as divergências encontradas foram provenientes de situações ocorridas durante o andamento dos processos de inscrições de candidatos, não sendo possíveis de serem tratadas pela DSL Cotas, uma vez que o próprio sistema de ingresso modificou as situações de classificações na medida em que outras chamadas foram sendo realizadas.

Nos demais casos de divergência foram identificados candidatos com problemas relacionados a operações na base de dados em função de abertura de chamados do setor demandante, ou situações que puderam ser identificadas e resolvidas por meio de correção do código da API. O detalhamento completo das divergências, assim como o relatório utilizado para a análise estão disponíveis no repositório *github* citado anteriormente.

Por fim, essa análise sugere que na maioria dos casos de cursos testados foi possível combinar o formalismo definido pela DSL Cotas em conjunto com a API, de modo a validar os diferentes tipos de cursos e versões da legislação já utilizadas pelo IFSC. A sua utilização pode favorecer a produtividade e a evolução de novas alterações em lei, de maneira agnóstica sem estar atrelada a uma GPL específica, uma vez que sua construção foi concebida a nível de serviços *web*.

7 CONCLUSÕES

7.1 TRABALHOS RELACIONADOS

Segundo Taha (2009), existem duas maneiras diferentes pelas quais um domínio pode ser definido. A primeira é quando o domínio está bem definido matematicamente, a segunda quando o domínio é definido por atividades puramente humanas. As DSLs de domínio humano, definem uma linguagem especial ou jargões para comunicar ideias relacionadas ao seu domínio.

Assim como a DSL Cotas, os trabalhos relacionados encontrados mostraram formas de modelar e abstrair conceitos e jargões envolvendo questões legais, contratuais e regras sobre domínios financeiros. Os domínios dessas DSLs foram definidos com objetivo de apoiar na compreensão de atividades humanas e não de tratar de problemas computacionais ou matemáticos.

Para tanto, as DSLs descritas nas próximas Subseções foram encontradas por meio da busca pelas palavras-chave (domain specific languages, DSL, law e legal) nos repositórios, *Researchgate*, *Google Scholar* e *IEEEXplore*.

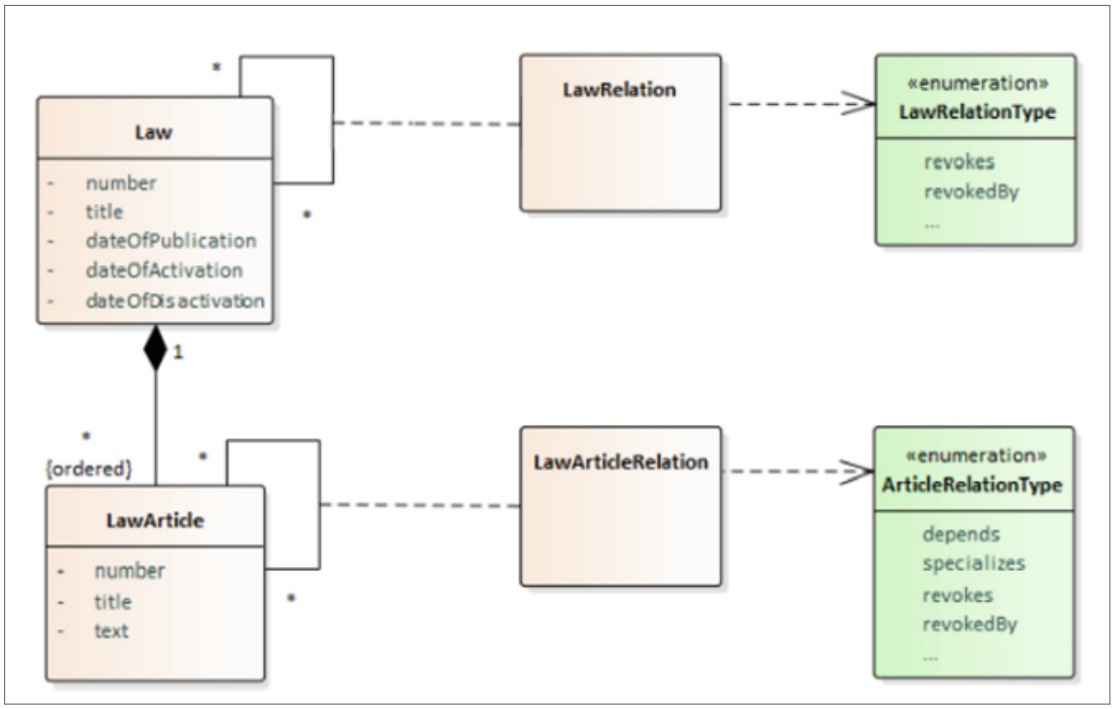
7.1.1 LegalLanguage

A DSL `LegalLanguage` é resultado da análise de ontologias que tratam de estrutura de documentos legais, sua construção busca estabelecer modelos conceituais com propósito de melhoria de comunicação e aprendizado sobre definições encontradas em textos normativos.

Apesar de tratar de domínio de questões legais, sua modelagem difere do presente trabalho pois a sua construção foi baseada em ontologias já existentes, com propósito de auxiliar em definições de documentos legais de maneira geral, enquanto a DSL Cotas foi modelada a partir do histórico de controle de versão entre as diferentes versões da legislação e trata especificamente da lei de cotas.

Segundo Soares, Martins e Silva (2019), o modelo da `LegalLanguage` é composto de *Laws*, que podem ser classificadas como: Leis internacionais, regulamentos públicos, regulamentos privados, constituições e leis ordinárias. As leis são compostas por uma série de artigos ordenados sequencialmente, os quais são organizados em divisões. É possível criar relacionamentos hierárquicos entre artigos de diferentes elementos de lei, bem como indicar revogações quando novas leis surgem (Figura 73).

Figura 73 – Modelo da DSL LegalLanguage



Fonte: Soares, Martins e Silva (2019).

Ainda para esses autores: "O desenvolvimento dessa linguagem tem como objetivo apoiar parlamentares que executam atividades inseridas em processos legislativos de redação de leis" (SOARES; MARTINS; SILVA, 2019, p. 45, tradução nossa). Portanto, os autores a criaram utilizando a ferramenta `Xtext`, um exemplo de definição e uso do elemento Law pode ser observado nas Figuras 74 e 75.

Figura 74 – Definição do elemento Law na LegalLanguage

```
Law:    'Law' name=ID  (nameAlias=STRING)? ':' type=LawType (':' subType=LawSubType)? ('['
                ('domain' subject=LawDomainType)?
                ('state' state=LawStateType)?
                (relationType+= LawRelationType relations+= RefLaw)*
                ('number' number=INT '/' year=INT)?
                ('title' title=STRING)?
                ('dateOfPublication' dateOfPublication=Date)?
                ('dateOfActivation' dateOfActivation=Date)?
                ('dateOfDisactivation' dateOfDisactivation=Date)?
                ('description' description=STRING)?
                (lawArticles+=LawArticle*)
                (lawDivisions+=LawDivision*)
        ']')? ;
enum LawType: Constitution | InternationalLaw | OrdinaryLaw |  PublicRegulation | PrivateRegulation |...;
enum LawRelationType: revokes | revokedBy;
```

Fonte: Soares, Martins e Silva (2019).

Figura 75 – Exemplo ilustrativo da LegalLanguage

```
Package EU_Laws
Law Directive_EC_46_95 "Directive n.º 95/46": InternationalLaw: InternationalLaw_Directive [
        domain IT
        state Revoked
        revokedBy EU_Laws.Regulation_EU_679_2016
        number 95/46
        dateOfPublication 24-Oct-1995
description "DIRECTIVE 95/46/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of
24 October 1995 on the protection of individuals with regard to the processing of personal data and on the
free movement of such data. THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE
EUROPEAN UNION, ..."
```

Fonte: Soares, Martins e Silva (2019).

7.1.2 Ergo uma DSL para Contratos Legais Inteligentes

Ergo é uma linguagem específica de domínio que faz parte do projeto Accord, o qual define a base jurídica e técnica para contratos legais inteligentes, com objetivo de atender aos problemas relacionados à falta de padronização em contratos legais.

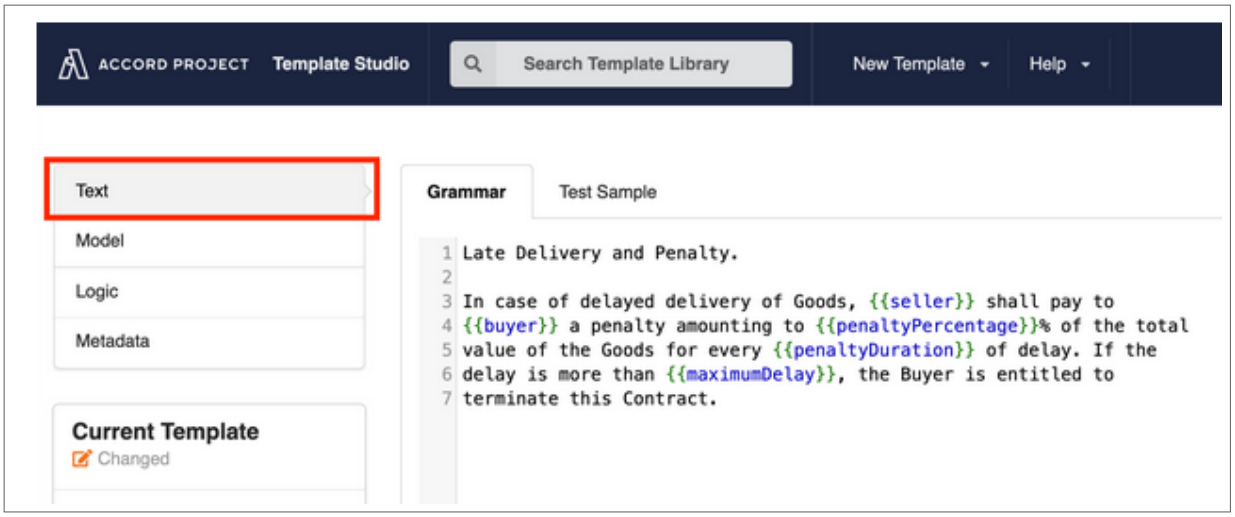
Segundo Accord (2019), a linguagem tem o foco de ajudar desenvolvedores da área de tecnologia jurídica na escrita de contratos legais que possam ser computados. Um contrato legal inteligente é um contrato legível por seres humanos e por máquinas, por exemplo, uma cláusula de cobrança de pagamento pode estar presente em um contrato, de modo que se possa utilizar o texto descrito em linguagem natural, para extração dos dados de cobrança e posterior aplicação de cálculos de multas e geração de eventos de notificações.

De forma similar a DSL Cotas, sua modelagem garante que as definições dos usuários sejam escritas com expressividade limitada, por outro lado, enquanto a DSL Cotas utiliza a definição das regras de distribuição de cotas em formato de tabelas, a Ergo DSL utiliza um formato textual controlado, similar ao de estrutura de classes para contratos e métodos para suas cláusulas.

É uma linguagem fortemente tipada e independente de plataforma, seu código pode ser compilado nas plataformas `JavaScript` e `Java`. Ela provê uma linguagem de expressões, pela qual é possível descrever funções e cláusulas de modo que possa ser definida a lógica de contratos inteligentes.

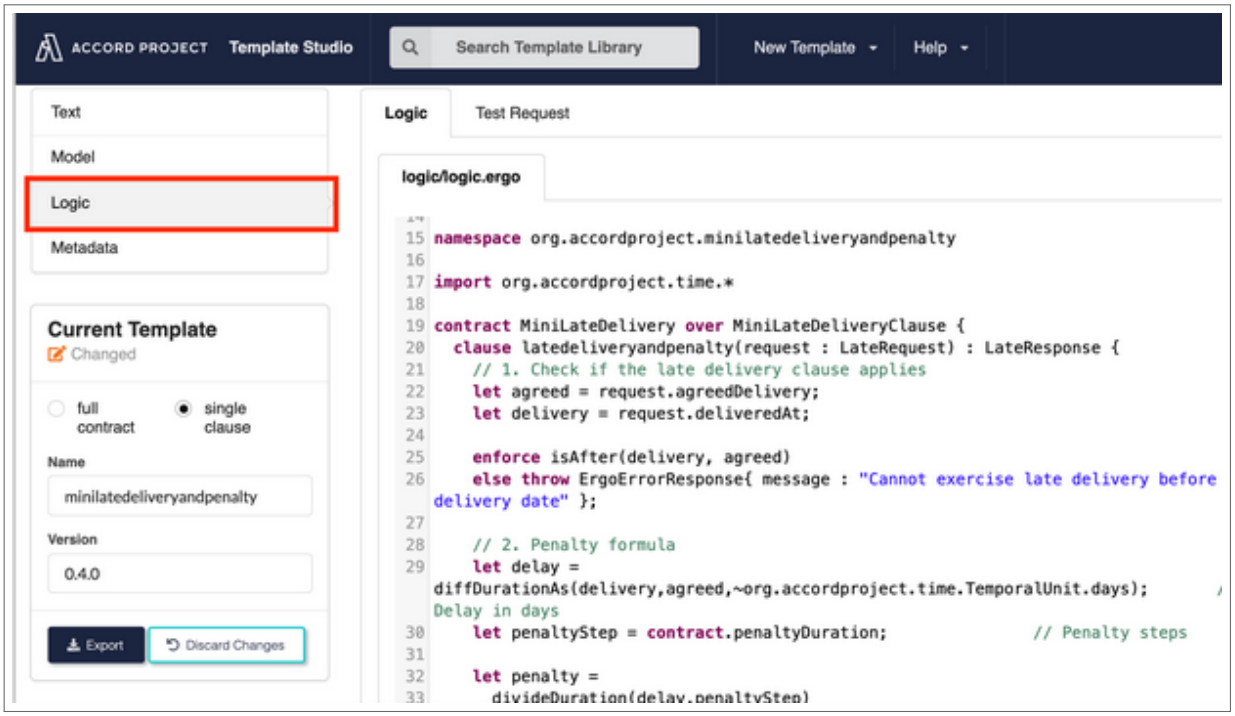
Um exemplo de linguagem natural pode ser observado na Figura 76, no qual os dados sobre a cláusula de cobrança são mapeados para posteriormente serem modelados e terem a lógica de cobrança seja definida (Figura 77).

Figura 76 – Exemplo ilustrativo da Ergo DSL



Fonte: Accord (2019).

Figura 77 – Definição da lógica de contratos



Fonte: Accord (2019).

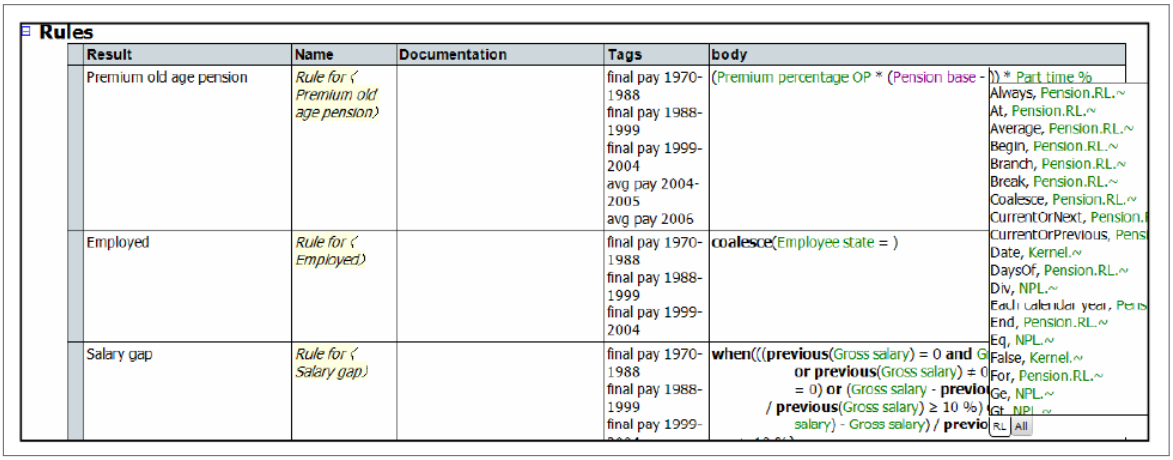
7.1.3 DSL Capgemini Pension

Segundo Fuller (2013), foi criada com o objetivo de atender ao domínio de planos de previdência e fornecimento de planos de seguros de pensão, no qual cada plano pode conter centenas de regras específicas, tratadas conforme o histórico de até 40 anos de dados funcionais para milhares de segurados.

Para Kolk e Volter (2008), essa linguagem advém da demanda de inovação em produtos da área e é baseada em interesses governamentais, no sentido de que há a necessidade de adequação às novas leis de pensão, dar transparência sobre as regras e fornecer garantia de qualidade. Portanto, essa DSL se relaciona ao presente trabalho no que diz respeito à necessidade de adequação, inovação e evolução para questões de interesses governamentais e para atendimento de novas legislações.

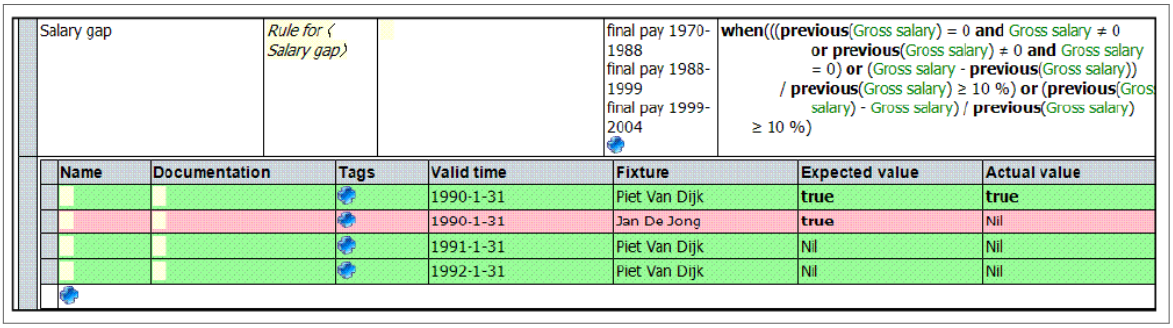
Criada com a ferramenta `Intentional Software` para construção de DSLs projecionais, ela permite que as definições das regras dos planos sejam realizadas em formato gráfico no estilo de tabelas de Excel (Figura 78) e permite fazer testes unitários para validar as regras em tempo real (Figura 79).

Figura 78 – Editor projecional das regras de pensão



Fonte: Kolk e Volter (2008).

Figura 79 – Testes unitários das regras



Fonte: Kolk e Volter (2008).

O ambiente da linguagem substitui uma série de planilhas, documentos de texto e outras fontes de controle utilizadas por múltiplos especialistas de domínio no contexto de gerenciamento dos planos de pensão. Dessa forma, a *Pension Workbench* traz um ambiente colaborativo controlado, com sistema integrado de controle de versão, sendo passível criar testes unitários e gerar a *engine* responsável pelos cálculos para a administração dos planos e fundos de pensão (FULLER, 2013).

Portanto, esses trabalhos relacionados assemelham-se à presente pesquisa no sentido de abordarem DSLs como meio de abstração de conhecimento de negócio. Eles visam a melhoria de processos que necessitem de maior clareza por parte dos envolvidos, criando um ambiente controlado que possa favorecer ao desenvolvimento do produto desejado pelos usuários especialistas.

Considerando a DSL desenvolvida por esse estudo, a seguir serão apresentados: o escopo negativo da pesquisa, as suas principais contribuições e os trabalhos futuros sugeridos.

7.2 ESCOPO NEGATIVO DA PESQUISA

Essa pesquisa é proveniente dos decorrentes problemas de entendimento, comunicação e manutenção dos sistemas do IFSC que utilizam regras relacionadas ao sistema de cotas da rede de ensino pública federal. Nesse contexto, procurou-se aplicar conceitos sobre linguagens específicas de domínio como meio de apoio durante as etapas de descrição e implementação das regras por parte usuários especialistas e desenvolvedores.

Portanto, limitou-se ao contexto da realidade do IFSC, no sentido de que a análise foi baseada em documentações internas, no sistema de controle de versão institucional e com usuários finais em sua maioria atuantes no setor Departamento de Ingresso (DEING). Apesar de ser baseada em uma legislação utilizada por várias instituições públicas federais, os resultados apresentados podem ter entendimento distintos por outras organizações ou setores, uma vez que os textos presentes na lei podem estar sujeitos a outras interpretações. Contudo, essas instituições podem se beneficiar da estrutura base da DSL desenvolvida adequando as definições conforme sua realidade e características regionais.

Ademais, o modelo da linguagem desenvolvida foi estabelecido a partir de elementos identificados no histórico de modificações dos documentos de lei de 2012 até o presente momento, tentou-se obter um modelo que pudesse contemplar novas modificações em lei, no entanto, as legislações futuras podem estar sujeitas a total reformulação ou até a sua extinção, o que não pode ser previsto no contexto do presente estudo, ficando a cargo dos designers da linguagem fazer as devidas atualizações de modelo. Cabe destacar que as adequações no modelo são facilitadas pelos recursos disponíveis na ferramenta MPS, o qual permite que as DSLs sejam estendidas e reutilizadas em outros contextos de aplicação.

Por uma questão de disponibilidade e de tempo hábil para aprofundamento do estudo foram selecionados 20 usuários para a realização do exercício com a DSL, alguns dos grupos de perfil de usuários tiveram menor representatividade em relação aos demais. Portanto, a

análise limitou-se aos usuários que concordaram em participar da pesquisa, na qual tentou-se obter uma variedade significativa e relevante para o estudo com usuários especialistas, desenvolvedores e leigos. No entanto, entende-se que este número é suficiente para validar os objetivos da presente pesquisa, considerando as afirmações de Nielsen (2012), que sugere que testes de usabilidade com apenas 5 (cinco) usuários permitem encontrar quase tantos problemas de usabilidade do que quando se utilizam estudos com mais participantes. Contudo, o referido autor defende que no caso de usuários de diferentes perfis, recomenda-se a aplicação de 3 (três) a 4 (quatro) usuários por grupo, porque as experiências se sobrepõem e ampliam a cobertura dos testes de usabilidade.

Por fim, sobre a construção da API como prova de conceito sobre a modelagem da DSL Cotas, por uma questão de limitação de acesso aos dados históricos de candidatos, foi realizada a validação com dados históricos de candidatos disponíveis no banco de dados do sistema de ingresso do IFSC. Essa validação não foi aplicada em outras bases de dados externas, o que poderia ampliar o estudo sobre outras realidades de resultados em processos seletivos fora da instituição.

7.3 PRINCIPAIS CONTRIBUIÇÕES

A presente pesquisa contribuiu para o aprofundamento do conhecimento sobre estudos que utilizam DSLs no contexto de modelagem para aplicação de regras estabelecidas em legislação. Do mesmo modo, avaliam-se os benefícios e os desafios da aplicação dessa proposta de engenharia de software para instituições federais de ensino que aplicam em seus processos seletivos as regras de distribuição de cotas.

Outra contribuição advém da análise dos resultados coletados com os 4 (quatro) grupos de usuários, no sentido de apresentar os diferentes relatos de dificuldades e preocupações sobre o uso de DSLs. Esse levantamento sugere que a validade de aplicação de uma nova linguagem pode variar de acordo com o nível de conhecimento na área de negócio e conforme o contexto de experiências profissionais e acadêmicas dos usuários.

Destaca-se, igualmente, a contribuição no âmbito pessoal e profissional do pesquisador em relação aos conhecimentos adquiridos por meio da pesquisa. Esses conhecimentos referem-se ao estudo de problemas causados durante a evolução do sistema de ingresso institucional, à identificação de padrões nas regras presentes em diferentes versões da legislação, o uso de ferramentas projecionais para construção de linguagens específicas de domínio, assim como a geração de código de processamento de candidatos a partir da sintaxe extraída da DSL.

Por fim, infere-se que além da relevância teórico-prática para a implementação de linguagem de domínio específico, o estudo aqui desenvolvido possui uma relevância social ao possibilitar maior comunicação entre a área de negócio e de desenvolvimento. E, assim, garantir que as políticas institucionais de equiparação de oportunidades para o ingresso de cotistas na rede federal de ensino sejam implementadas com mais produtividade e de modo colaborativo entre os envolvidos.

7.4 TRABALHOS FUTUROS

Como proposta de trabalhos futuros sugere-se:

a) Verificar a adesão da DSL Cotas por usuários especialistas e desenvolvedores de outras instituições de ensino pública federais;

b) Disponibilizar publicamente os serviços da API DSL Cotas para implantação em redes de ensino federadas, com intuito de verificar a aplicação em outras bases de dados por meio da colaboração e testes com outros setores de desenvolvimento de sistemas da rede;

c) Criar um repositório de controle de versão centralizado para publicação das definições de regras de cotas presentes nas diferentes instituições e estados. De modo que se possa verificar a transparência nas regras utilizadas e garantir a rastreabilidade de alterações realizadas conforme cada edital de processo seletivo que requer atendimento da legislação de cotas;

d) Investigar outras formas de visualização e geração de código a partir da DSL Cotas, tais como: geração de editais, geração do quadro de vagas, geração de resultados e geração de elementos de interface gráfica para sistemas e aplicativos de inscrição em processos seletivos de modo que possam ser apresentadas as definições de categorias durante o processo de inscrição até o momento final de classificação;

e) Avaliar e modelar alterações na DSL Cotas para aplicação em outros tipos de processos seletivos que envolvam classificação por cotas fora do âmbito da Lei nº 12.711.

7.5 CONSIDERAÇÕES FINAIS

O avanço na complexidade das regras de negócio traz consigo uma série de discussões e estudos que objetivam abstrair e simplificar a sua implementação nos sistemas de informação, de modo a reduzir os impactos durante o seu desenvolvimento, podendo melhorar o desempenho das organizações.

As ferramentas modernas de DSL permitem a especificação de gramáticas e estruturas de linguagem, sem que haja a necessidade de manualmente criar um *parser*. Elas permitem a validação e a criação de estruturas e tipos de dados, assim como fornecem uma série de recursos para apoio ao usuário da linguagem, podendo agilizar o processo de definição de regras e a respectiva geração de código fonte.

Nesse contexto, a presente pesquisa buscou compreender, por meio da elaboração de uma linguagem específica de domínio, a viabilidade de melhoria na comunicação entre usuários de negócio e desenvolvedores, no que concerne à especificação de requisitos e a implementação de regras relacionadas ao sistema de cotas da rede de ensino pública federal.

A pesquisa apresentada buscou contextualizar uma série de demandas de alteração no sistema de ingresso do IFSC, que trazem dependência entre os especialistas de domínio e os desenvolvedores, no sentido de que as regras precisam ser traduzidas em várias linhas de código, para que sejam criadas as funcionalidades que implementam a classificação de candidatos conforme os requisitos de lei.

Portanto, para a criação da DSL Cotas foram fundamentados os principais conceitos de linguagens específicas de domínio, que possibilitaram definir uma sintaxe de definição de regras mais simples para o usuário tratar de questões de negócio, sem que esses precisem se preocupar com conhecimentos mais complexos de programação, mas ainda assim, colaborando com o processo de modelagem e construção da solução.

Adicionalmente ao desenvolvimento da DSL Cotas, foi construída uma API que importa as definições da linguagem para disponibilizar o serviço de classificação e aprovação de candidatos, servindo também como prova de conceito e validade da modelagem estabelecida por esse estudo.

Os sujeitos que fizeram parte dessa investigação foram 20 usuários com perfis diferentes, que foram divididos em 4 (quatro) grupos nomeados de DEV-ESP, DEV-NESP, NDEV-EPS e NDEV-NESP. Esses usuários foram convidados a responder a um exercício de utilização da DSL Cotas e um questionário de avaliação qualitativa da experiência de uso.

Dessas ações - o levantamento de dados do histórico de versionamento, o desenvolvimento da DSL, a aplicação do exercício e o questionário de avaliação - resultou a investigação para responder ao problema de pesquisa.

Em relação ao levantamento do sistema de controle de versão, foi possível identificar as diferenças de implementação entre as alterações de lei já realizadas no IFSC. A respeito do objetivo específico de analisar essas características foi possível atender ao terceiro objetivo, que diz respeito à modelagem de regras por meio de uma linguagem específica de domínio. Com o intuito de verificar o quarto objetivo específico, que trata da avaliação de uso da DSL pelos usuários convidados, observou-se que o grupo que teve mais facilidade de entendimento e utilização dos recursos da DSL Cotas foi o grupo DEV-ESP, no entanto, o grupo NDEV-ESP foi o que conseguiu configurar distribuições mais recentes da legislação, sem que fosse solicitado. Os demais grupos DEV-NESP e NDEV-NESP apontaram um grau de dificuldade maior em relação ao seu uso em função de lacunas de conhecimento sobre domínio ou por dificuldades técnicas.

Assim, por meio da pesquisa realizada compreende-se que a abordagem de aplicação da DSL Cotas no contexto de definição de regras da legislação, mostrou-se viável para contribuir na melhoria de comunicação entre usuários de negócio e desenvolvedores, inferindo-se desse modo, que há a possibilidade no aumento de produtividade da especificação de requisitos e a respectiva implementação de regras concernentes ao sistema de cotas da rede de ensino pública federal.

REFERÊNCIAS

ACCORD, P. *The Accord Project Ergo Overview, 2019*. 2019. Disponível em: <https://docs.accordproject.org/docs/logic-ergo.html>. Acesso em: 05 jun. 2020.

BENTLEY, J. Programming pearls: little languages. *Communications of the ACM*, ACM, v. 29, n. 8, p. 711–721, 1986. Disponível em: <https://dl.acm.org/citation.cfm?id=315691>.

BERGMANN, S. A tool for quickly measuring the size of a php project. 2020. Disponível em: <https://github.com/sebastianbergmann/phploc>. Acesso em: 18 jul. 2020.

BERSTEL SILVA, B. *Verification of Business Rules Programs*. Springer, 2013. ISBN 9783642400384. Disponível em: <https://www.springer.com/br/book/9783642400377>.

BRASIL. *LEI Nº 12.711, DE 29 DE AGOSTO DE 2012*. Brasília: Congresso Nacional, 2012. Acessado: 22 ago. 2019. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2012/lei/l12711.htm>. Acesso em: 22 ago. 2019.

BRASIL. *LEI Nº 13.409, DE 28 DE DEZEMBRO DE 2016*. Brasília: Congresso Nacional, 2016. Acessado: 30 ago. 2019. Disponível em: <http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2016/Lei/L13409.htm>. Acesso em: 30 ago. 2019.

CAMPAGNE, F. *The MPS language workbench: volume I*. [S.l.]: Fabien Campagne, 2014. v. 1.

CHRIS, W.; DANIELA, R. Towards a flexible deployment of business rules. *Expert Systems with Applications*, Elsevier, n. 23, p. 385–394, 2002.

EYSHOLDT, M.; BEHRENS, H. Xtext: implement your language faster than the quick and dirty way. In: ACM. *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. 2010. p. 307–309. Disponível em: <https://dl.acm.org/citation.cfm?id=1869625>. Acesso em: 12 set. 2019.

FOWLER, M. Language workbenches: The killer-app for domain specific languages. 2005. Disponível em: <https://martinfowler.com/articles/languageWorkbench.html>. Acesso em: 12 set. 2019.

FOWLER, M. *ProjectionalEditing*. dslbook. org, 2008. Disponível em: <https://martinfowler.com/bliki/ProjectionalEditing.html>. Acesso em: 03 set. 2019.

FULLER, G. S. G. Care to cram. 2013. Disponível em: <https://pdfs.semanticscholar.org/cb55/cde363e61b99d06ea74572f0947a7a49e900.pdf>. Acesso em: 05 jun. 2020.

GHOSH, D. Dsl for the uninitiated. *Communications of the ACM*, ACM, v. 54, n. 7, p. 44–50, 2011. Disponível em: <https://homepages.dcc.ufmg.br/~mtov/arqsw/p44-ghosh.pdf>. Acesso em: 03 set. 2019.

GIL, A. C. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas São Paulo, 2002. v. 4.

HOFFMANN, B.; CHALMERS, K.; URQUHART, N.; GUCKERT, M. Athos - a model driven approach to describe and solve optimisation problems: An application to the vehicle routing problem with time windows. In: *Proceedings of the 4th ACM International*

Workshop on Real World Domain Specific Languages. New York, NY, USA: Association for Computing Machinery, 2019. (RWDSL '19). ISBN 9781450366373. Disponível em: <https://doi.org/10.1145/3300111.3300114>.

ITEMIS, T. Engineering the future of embedded software. *The website of mbeddr*, 2020. Disponível em: <http://mbeddr.com/>. Acesso em: 24 mai. 2020.

JETBRAINS, M. Mps editor cookbook. *Jetbrains MPS User's Guide*, 2019. Disponível em: <https://www.jetbrains.com/help/mps/>. Acesso em: 21 mai. 2020.

KATS, L. C.; VISSER, E. The spoofax language workbench: rules for declarative specification of languages and ides. *ACM sigplan notices*, ACM, v. 45, n. 10, p. 444–463, 2010. Disponível em: <https://dl.acm.org/citation.cfm?id=1869497>. Acesso em: 12 set. 2019.

KOLK, H.; VOLTER, M. *Democratizing Software Creation Presentation*. 2008. Disponível em: <http://voelter.de/data/presentations/KolkVoelter_IntentionalSoftware.pdf>. Acesso em: 05 jun. 2020.

MEC, M. d. E. Portaria normativa nº18, de 11 de outubro de 2012. *Diário Oficial da União*, p. 16–17, 2012. Disponível em: <http://sisugestao.mec.gov.br/docs/portaria-18-2012.pdf>.

MEC, M. d. E. *Entenda as cotas para quem estudou todo o ensino médio em escolas públicas*. Brasília: Ministério da Educação, 2018. Acessado: 30 ago. 2019. Disponível em: <http://portal.mec.gov.br/cotas/>. Acesso em: 30 ago. 2019.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM, v. 37, n. 4, p. 316–344, 2005. Disponível em: <https://dl.acm.org/citation.cfm?doid=1118890.1118892>. Acesso em: 03 set. 2019.

MORAN, K. Writing tasks for quantitative and qualitative usability studies. *Nielsen Norman Group*, 2018. Disponível em: <https://www.nngroup.com/articles/test-tasks-quant-qualitative/>. Acesso em: 17 mai. 2020.

NIELSEN, J. How many test users in a usability study. *Nielsen Norman Group*, v. 4, n. 06, 2012.

NOVÁK, M. Easy implementation of domain specific language using xml. In: *10th Scientific Conference of Young Researchers (SCYR 2010)*. [s.n.], 2010. Disponível em: <https://www.researchgate.net/publication/228458637_Easy_Implementation_of_Domain_Specific_Language_using_XML>. Acesso em: 06 set. 2019.

POLTRONIERI, I.; ZORZO, A. F.; BERNARDINO, M.; CAMPOS, M. de B. Usability evaluation framework for domain-specific language: a focus group study. *ACM SIGAPP Applied Computing Review*, ACM New York, NY, USA, v. 18, n. 3, 2018. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/3284971.3284973>. Acesso em: 01 jun. 2020.

PRIMAO PACHECO, A. *Catálogo de Serviços de TIC*. Florianópolis: Instituto Federal de Santa Catarina, 2019. Acessado: 22 ago. 2019. Disponível em: <https://dtic.ifsc.edu.br/servicos-de-tic/>. Acesso em: 22 ago. 2019.

RUNESON, P.; HöST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, Kluwer Academic Publishers, USA, v. 14, n. 2, p. 131–164, abr. 2009. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-008-9102-8>.

SILVA, E. L. d. M. d. pesquisa e elaboração de dissertação/edna lúcia da silva, estera muszkat menezes.–4. ed. rev. atual. *Florianópolis: UFSC*, p. 138, 2005.

SIMONYI, C.; CHRISTERSON, M.; CLIFFORD, S. Intentional software. In: ACM. *ACM SIGPLAN Notices*. 2006. v. 41, n. 10, p. 451–464. Disponível em: <https://dl.acm.org/citation.cfm?id=1167511>. Acesso em: 12 set. 2019.

SOARES, A.; MARTINS, P.; SILVA, A. Legallanguage: A domain-specific language for legal contexts. In: _____. [S.l.: s.n.], 2019. ISBN 978-3-030-37932-2.

SPINELLIS, D. Notable design patterns for domain-specific languages. *Journal of systems and software*, Elsevier, v. 56, n. 1, p. 91–99, 2001.

TAHA, W. *Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings*. [S.l.: s.n.], 2009.

VOELTER, M. *Generic tools, specific languages*. Citeseer, 2014. ISBN 9781500359430. Disponível em: <http://voelter.de/data/books/GenericToolsSpecificLanguages-1.0-web.pdf>. Acesso em: 10 set. 2019.

VOELTER, M. *The Design, Evolution, and Use of KernelF*. Springer, Cham, 2018. ISBN 9783319933160. Disponível em: <https://link.springer.com/chapter/10.1007%2F978-3-319-93317-7_1>.

VOELTER, M.; BENZ, S.; DIETRICH, C.; ENGELMANN, B.; HELANDER, M.; KATS, L. C.; VISSER, E.; WACHSMUTH, G. *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013. ISBN 9781481218580. Disponível em: <http://voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf>.

VOELTER, M.; LISSON, S. Supporting diverse notations in mps'projectional editor. In: . [s.n.], 2014. Disponível em: <https://hal.inria.fr/hal-01074602/document/#page=12>. Acesso em: 27 mai. 2020.

VOLTER, M. Language and ide modularization, extension and composition with mps. *Generative and Transformational Techniques in Software Engineering*, v. 124, 2011. Disponível em: <https://voelter.de/data/pub/Voelter-GTTSE-MPS.pdf>. Acesso em: 13 set. 2019.

WALLS, C. *Spring Boot in action*. Manning Publications, 2016. Disponível em: <https://www.nitinagrawal.com/uploads/2/1/3/6/21361954/spring_boot_in_action.pdf>. Acesso em: 26 mai. 2020.

WEBB, P. et al. Spring boot reference guide. 2018. Disponível em: <https://leon-wtf.github.io/doc/spring-boot-reference.pdf>. Acesso em: 19 mai. 2020.

WEXELBLAT L., R. *History of Programming Languages*. Academic Press Inc, 1981. 594 p. ISBN 0127450408. Disponível em: <https://dl.acm.org/citation.cfm?id=800025>.

XTEXT. *XText 15 Minutes Tutorial*. 2019. Acessado: 12 set. 2019. Disponível em: <https://www.eclipse.org/Xtext/documentation/102_domainmodelwalkthrough.html>. Acesso em: 12 set. 2019.

APÊNDICE A – LEVANTAMENTO DE ALTERAÇÕES DE COTAS

Zimbra **daniel.estrazulas@ifsc.edu.br**

Alterações de cotas

De : R*******@ifsc.edu.br> Ter, 01 de out de 2019 11:13
Assunto : Alterações de cotas 8 anexos
Para : daniel estrazulas <daniel.estrazulas@ifsc.edu.br>

Olá, Daniel.

Tentei resgatar os e-mails das alterações das cotas. Coloquei numa ordem cronológica das alterações numerando os nomes dos arquivos.

Mais informações da legislação tem no link: https://www.ifsc.edu.br/cotas

Obs:
Lei PCD publicada em 28/12/16;
Decreto que regulamenta a lei publicado em 20/04/17;
Portaria Normativa (que detalhou um pouco melhor a lei) somente em 05/05/17.

Por esta razão cronológica, e por não haver orientação do MEC até então, tivemos que inicialmente "interpretar" a lei aqui na instituição (DEING, DAE, PROEN, PGF e Assessoria Técnica) para os editais de 2017/2, que foram publicados antes da existência do Decreto e da Portaria, em março de 2017 (4 cotas com PPI/PCD juntas, para 2017/2).

Após as publicações e com confirmação do MEC e do SISUGestão, ajustamos para 8 cotas com PPI e PCD separados, para 2018/1.

Qualquer dúvida, estou a disposição.

Abraço.

****

Chefe do Departamento de Ingresso

Pró-Reitoria de Ensino - PROEN

(48) 3877-9022

Instituto Federal de Santa Catarina - Reitoria

Rua 14 de Julho , 150, Coqueiros, Florianópolis / SC - CEP: 88075-010

www.ifsc.edu.br

7_reajuste para 8 cotas - 2018.1 - historico chamado.pdf
127 KB

8_resposta do SISUGestão a consulta sobre cotas e percentuais - maio 2018.pdf
420 KB

6_reajuste de 24 para 37 - histórico do chamado.pdf
193 KB

5_reajuste de 24 para 37 - apos divulgação SISU e consulta a assessoria tecnica e gestao.pdf
206 KB

4_histórico ajuste no percentual de 38 para 24 - cota PCD junto com PPI.pdf
170 KB

3_consulta a PGFsobre inserção de cota PCD junto com PPI.pdf
107 KB

2_solicitação de inserção de cota PCD junto com PPI - todo historico do chamado.pdf
282 KB

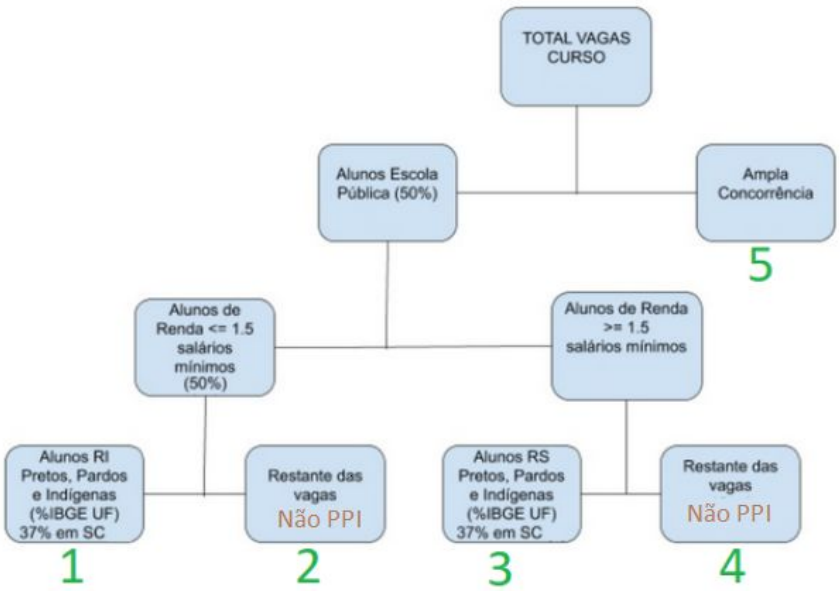
1_solicitação de inserção de cota PCD junto com PPI.pdf
105 KB

APÊNDICE B – EXERCÍCIO DE AVALIAÇÃO

DSL Cotas

Exercício Proposto

Acesse o ambiente remoto, conforme instruções passadas por e-mail, e tente reproduzir a seguinte distribuição de vagas utilizando os recursos e comandos disponíveis pela DSL de cotas:



Considere a ordem de prioridade como sendo a dos números indicados na cor **verde** na figura.

Dica: Informe o percentual ou configuração de cota no campo “R:” e na cota de mesmo nível seguinte utilize a constante “RESTANTE_VAGAS”, esta constante indicará para o sistema de informação alvo que após aplicar o percentual, o que sobrar conforme arredondamento será alocado para a cota seguinte.

Iremos salvar seu exercício para fins de registro na pesquisa, então não é necessário apagar o conteúdo do que tiver conseguido fazer.

Após a execução do exercício pedimos que acesse o link https://forms.gle/Aex1xgJAc7t2LVSP8 para responder algumas perguntas sobre a linguagem criada nessa pesquisa, isso será de muita importância para o estudo. Obrigado!

DSL Cotas
Manual de Utilização

Resumo

1. Conteúdo:

   Convidamos você para fazer um exercício de avaliação sobre o uso da linguagem de domínio específica (DSL Cotas). Essa linguagem foi criada para descrever regras do sistema de classificação de candidatos conforme a Lei nº 12.711 e outros documentos de lei que são utilizados nos sistemas de ingresso da rede pública de ensino federal.

   Nessa linguagem será possível preencher os parâmetros e percentuais, assim como criar as divisões entre as categorias de cotas vinculadas ao sistema de escola pública tais como: renda inferior ou superior a 1,5 salários mínimos per capita, candidatos autodeclarados PPI (pretos, pardos ou indígenas), candidatos com deficiência (PCD) ou outras possíveis categorias de distribuição de vagas.

   Após a execução do exercício proposto no fim desse documento, você poderá acessar o link https://forms.gle/Aex1xgJAc7t2LVSP8 para responder questões sobre impressões de uso da DSL Cotas, contribuindo assim para o resultado final da pesquisa.

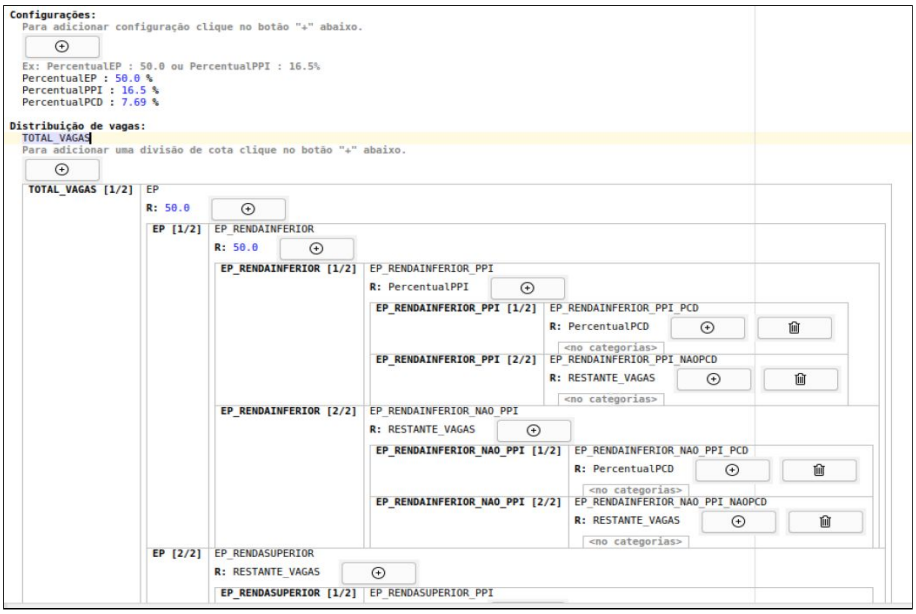
   Importante: Para facilitar o entendimento, fizemos um vídeo exemplificando e comentando o uso da DSL de Cotas, leia este manual e veja o vídeo disponível no link https://youtu.be/RcBNlhD5dTA antes de iniciar.

2. Tempo alocado: De 10 a 20 minutos, aproximadamente. (Por favor anote o tempo total que você utilizou para executar a atividade).
3. Objetivo:

   Após a execução de alguns comandos e recursos da linguagem você terá definido algumas regras que serão de grande utilidade para implementação de funcionalidades de distribuição de vagas em sistemas de informação institucionais.

   Essas definições serão baseadas na seguinte estrutura disponível no portal do MEC: http://portal.mec.gov.br/cotas/sobre-sistema.html

4. Resultado Esperado: Ao final do exercício você terá uma estrutura de definições dos percentuais de cotas de forma similar ao da figura abaixo:



Seções da DSL Cotas

- **Dados da Lei:** Essa seção é responsável por identificar a versão equivalente na legislação e a instituição responsável pela definição das regras em suas unidades.

```
Lei 12.711 - Exercício para avaliação da DSL

Dados da Lei:
  Sigla Instituição: IFSC
  Número da Lei: 12711
  Código Versão: 001
```

- **Configurações:** Seção que manterá a lista de parâmetros de percentuais a serem utilizados na distribuição de vagas. Obs: Utilize sempre o formato fracionário. Ex: **50.0** ou **16.5**.

```
Configurações:
  Para adicionar configuração clique no botão "+" abaixo.
  ⊕
  Ex: PercentualEP : 50.0 ou PercentualPPI : 16.5%
  ConfiguracaoExemplo : 50.0 %
```

Obs: Uma configuração pode ser utilizada no momento de criar a árvore de distribuição de categorias (seção abaixo), por exemplo ao invés de preencher 16.5 em várias categorias onde esse percentual vai ser repetir, utilizando (control + espaço) será possível selecionar esta configuração pré criada.

- **Distribuição de vagas:** Seção em que será possível criar subdivisões de categorias, sempre iniciará com a categoria “Total de Vagas”. **Obs:** A sigla “**R:**” é o local para

selecionar uma configuração da seção “Configurações” ou fazer preenchimento do percentual manualmente, ex: 50.0 (somente números). No campo name utilize siglas em maiúsculas.

- **Forma de Arredondamento:** Seção em que será possível indicar o tipo de arredondamento, por exemplo se os números fracionários serão convertidos sempre para baixo ou para cima (>= 4.5).

- **Ordem de Prioridade:** Seção em que poderá ser indicada a ordem de prioridade para cada uma das categorias de cota **que serão aptas para inscrição de candidatos**. Ela serve para indicar qual categoria seguinte que receberá a sobra de vagas, no caso de candidatos insuficientes em uma determinada cota.

Comandos para Edição das Regras:

Cursor do Mouse	Utilizado para acesso das seções e itens da linguagem, como por exemplo: • clique nos botões de distribuição/remoção de categorias; • adição de novas configurações; • inspeção de erros (“itens em destaque em vermelho”) ; • inspeção de avisos (“itens em destaque em amarelo”).
Comando (Control+Espaço)  + 	Utilizado para indicar e referenciar uma configuração de cota para ser utilizada em uma distribuição. Também pode ser utilizada para selecionar categorias de cotas existentes para preenchimento da ordem de prioridade.
Comando (Tab) 	Tecla para avanço do cursor nos elementos da linguagem. (Alternativa para uso do mouse).
Comandos (Backspace) e (Delete)  	Comando para apagar conteúdo e elementos criados na linguagem.

Comando (Control + Z) 	Desfazer alguma ação realizada.
Comando (Alt + Enter) 	Alguns avisos de erro na linguagem permitem a correção automática. Exemplos: ● Remoção de categoria com nome duplicado; ● Correção de nomenclatura para os sugeridos da categoria; ● Correção automática para a constante “RESERVA_VAGAS” ser a última da lista em determinada categoria.

APÊNDICE D – E-MAIL PARA AVALIAÇÃO DA PESQUISA

Daniel Severo Estrazulas <dse@cin.ufpe.br>

Teste para avaliação da DSL COTAS

Daniel Severo Estrazulas <dse@cin.ufpe.br> 6 de maio de 2020 17:38
Para: ****@gmail.com

Olá,

Conforme informado anteriormente estou dando início aos testes da linguagem DSL Cotas que faz a descrição de regras da legislação para o sistema de cotas da rede de ensino pública federal.

Desse modo, convido você a participar da presente pesquisa realizando os seguintes passos:

1) Ler o roteiro/manual de utilização em anexo

2) Para ver um exemplo de uso acesse o vídeo abaixo
https://youtu.be/RcBNlhD5dTA

3) Baixar o programa "VNC Viewer" para acesso ao ambiente de testes da linguagem.
(acesse o guia em anexo para se conectar ao ambiente de testes)
https://www.realvnc.com/pt/connect/download/viewer/

4) Após iniciar o exercício proposto no manual, **por favor cronometre o tempo levado para finalização**

5) Após finalizar o exercício acesse o link abaixo para responder algumas perguntas sobre a experiência de uso da linguagem, **e responda esse e-mail informando a conclusão** para que eu possa salvar seu exercício e liberar o ambiente para outra pessoa
https://forms.gle/Aex1xgJAc7t2LVSP8

Farei o sorteio de um Voucher no valor de R$ 100,00 reais aos participantes que concluírem o exercício e contribuírem com a pesquisa. Temos o objetivo de atingir pelo menos 20 participantes, e após o término faremos um sorteio que será gravado para divulgar o ganhador.

Obrigado pelo apoio.
Daniel Severo Estrázulas
Aluno do Programa de Mestrado em Ciências da Computação na UFPE

2 anexos

Guia de Instalação do VNC Viewer (dslcotas).pdf
264K

Roteiro de utilização - DSL Cotasv2.pdf
254K

APÊNDICE E – QUESTIONÁRIO DE AVALIAÇÃO

Avaliação de experiência de uso da DSL Cotas

Olá, me chamo Daniel Severo Estrázulas e sou aluno do Programa de Mestrado Profissional em Ciências da Computação da UFPE.

Estou realizando uma pesquisa para avaliar a experiência de uso da Linguagem de Domínio Específica criada para descrever regras de classificação de candidatos ao sistema de cotas da rede pública federal de ensino. Para isso, gostaria de pedir alguns minutos do seu tempo para responder ao questionário.

Sua contribuição é muito importante para a pesquisa, manteremos o anonimato dos participantes, deste modo agradecemos a sua participação.

Aluno: Daniel Severo Estrázulas
Orientador: Leopoldo Motta Teixeira
*Obrigatório

Endereço de e-mail *

1. Qual o seu nível de formação? *

Marcar apenas uma oval.

- ◯ Ensino Médio
- ◯ Ensino Superior
- ◯ Mestrado
- ◯ Doutorado

2. Qual o curso da sua formação acadêmica? *

3. Qual o seu cargo? *

Marcar apenas uma oval.

- ◯ Profissional na àrea de Tecnologia da Informação
- ◯ Profissional atuante em setor/departamento de ingresso
- ◯ Outro: _____

4. Você tem experiência com desenvolvimento de sistemas? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

5. Quanto tempo de experiência profissional? *

Marcar apenas uma oval.

- ◯ 0-2 anos
- ◯ 3-5 anos
- ◯ 5-10 anos
- ◯ > 10 anos

6. Você já atuou em sistemas ou processos seletivos que utilizam regras de classificação para candidatos cotistas? *

Sistema de cotas da escola pública ou sistemas de concurso público.

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

7. Qual o seu grau de conhecimento sobre o sistema de cotas da Lei nº 12.711/2012 e suas atualizações? *

A Lei nº 12.711/2012 rege as regras de classificação de candidatos que concorrem aos cursos da rede pública federal de ensino.

Marcar apenas uma oval.

	1	2	3	4	5	
Nenhum	◯	◯	◯	◯	◯	Conheço Amplamente

8. Você já utilizou alguma Linguagem de Domínio Específica? *

Linguagem de Domínio Específica: São linguagens geralmente pequenas que possuem o propósito de facilitar a realização de tarefas menores para resolução de problemas específicos. Podem estar presentes dentro de um software ou ferramentas de desenvolvimento. Exemplos: HTML, Excel, Latex, SQL, etc.

Marcar apenas uma oval.

◯ Sim
◯ Não

9. De modo geral, qual o grau de dificuldade para execução do exercício proposto? *

Marcar apenas uma oval.

	1	2	3	4	5	
Difícil	◯	◯	◯	◯	◯	Fácil

10. Considerando a sua resposta na pergunta anterior, qual(is) a(s) dificuldade(s) encontrada(s)? *

11. Na sua avaliação quais as principais limitações da linguagem proposta? *

Marque todas que se aplicam.

- [ ] As sugestões das mensagens informativas não são claras o suficiente
- [ ] O tempo de aprendizagem da linguagem é extenso
- [ ] Os elementos da linguagem não atendem todas as regras da legislação
- [ ] Não sei avaliar

Outro: [ ] _____

12. Quais dos materiais abaixo você utilizou para execução do exercício? *

Marque todas que se aplicam.

- [ ] Manual de Utilização
- [ ] Vídeo explicativo contendo exemplo de uso da DSL
- [ ] Ambiente disponibilizado para o exercício

13. No caso de a linguagem apresentar erros "destaques em vermelho" ou avisos "destaques em amarelo", as mensagens foram claras e ajudaram a resolver o(s) problema(s) apresentado(s)? *

Marcar apenas uma oval.

	1	2	3	4	5	
Não ajudaram	◯	◯	◯	◯	◯	Ajudaram Totalmente

14. Quanto tempo, aproximadamente, você utilizou para executar o exercício proposto? *

Por favor, incluir o tempo de leitura do manual e visualização do vídeo de explicativo de uso (se for o caso).

Marcar apenas uma oval.

- ◯ Menos de 15 minutos
- ◯ De 15 minutos até 30 minutos
- ◯ Mais de 30 minutos
- ◯ Mais de 1 hora

15. Comentários e Sugestões