**UNIVERSIDADE FEDERAL DO PAMPA**
**BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO**

**TIAGO LEITE BRITO**

# SISTEMA DE APOIO AO PROJETO E IMPLANTAÇÃO DE INFRAESTRUTURA EM NUVEM

**Bagé**
**2023**

**TIAGO LEITE BRITO**

# SISTEMA DE APOIO AO PROJETO E IMPLANTAÇÃO DE INFRAESTRUTURA EM NUVEM

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientadora: Ana Paula Lüdtke Ferreira

**Bagé**
**2023**

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

B862s Brito, Tiago Leite

Sistema de apoio ao projeto e implantação de infraestrutura em nuvem / Tiago Leite Brito.

- 2023.

104 f.: il.

Orientadora: Ana Paula Lüdtke Ferreira

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Campus Bagé, Bacharelado em Engenharia de Computação, 2023.

1. Arquitetura orientada a modelos. 2. Geração de código. 3. Linguagens de domínio específico. 4. Tradutores. I. Ana Paula Lüdtke Ferreira. II. Título.

SERVIÇO PÚBLICO FEDERAL

MINISTÉRIO DA EDUCAÇÃO

Universidade Federal do Pampa

**TIAGO LEITE BRITO**

# SISTEMA DE APOIO AO PROJETO E IMPLANTAÇÃO DE INFRAESTRUTURA EM NUVEM

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 1o de fevereiro de 2023

Banca examinadora:

_____

Profa. Dra. Ana Paula Lüdtke Ferreira
Orientadora

UNIPAMPA

_____

Prof. Dr. Leonardo Bidese de Pinho
UNIPAMPA

_____

Prof. Dr. Sandro da Silva Camargo
UNIPAMPA

Assinado eletronicamente por **ANA PAULA LUDTKE FERREIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/02/2023, às 00:24, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **LEONARDO BIDESE DE PINHO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/02/2023, às 09:52, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **SANDRO DA SILVA CAMARGO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/02/2023, às 23:01, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1049214** e o código CRC **BF4283BE**.

Referência: Processo nº 23100.002573/2023-40 SEI nº 1049214

## RESUMO

A demanda por profissionais de tecnologia em diversas áreas vem aumentando acentuadamente nos últimos anos e a projeção é que continue a crescer. Em uma situação contrária, a formação de profissionais para ocupar essas posições é insuficiente. Como consequência, as empresas recorrem a contratar pessoas com pouca ou nenhuma formação na área e acabam sobrecarregando os profissionais sêniores. Outro ponto que permeia está situação é a comunicação deficiente entre equipes multidisciplinares, resultado do desconhecimento dos processos das outras áreas ou da falta de experiência dos integrantes da equipe. Para auxiliar na otimização dos processos de provisionamento de infraestrutura em nuvem, uma das áreas tecnológicas carente de profissionais, este trabalho desenvolve uma plataforma para simplificar a comunicação entre equipes diversas, com profissionais em diferentes níveis de expertises e que reduza o número de etapas necessárias para a conclusão do processo. Em uma visão externa, esses dois problemas – comunicação deficitária e falta de profissionais no mercado – aparentam não estar conectados, mas em conjunto acarretam em retrabalho dentro de empresas, não cumprimento dos prazos e, principalmente, custos elevados dos projetos. Uma abordagem visual, intuitiva e baseada em automação é definida para atenuar esse cenário e apresentada neste trabalho. Conceitos de *Model Driven Architecture* são usados para retirar a atenção do ferramental e a colocar no problema que a equipe tem que resolver. O processo é dividido em etapas que vão de um nível mais alto e abstrato, em que o problema é situado no mundo real até etapas pertencentes a um nível mais baixo e concreto, em que o problema é transposto para a infraestrutura. Todo o processo é realizado manipulando diagramas, um para cada tipo de profissional ou etapa do processo, e gerenciados por uma Linguagem Visual de Domínio Específico. Os processos que necessitam de artefatos finais do tipo documentação ou código são automatizados com a utilização das linguagens construídas para cada fim.



**Palavras-chave:** Arquitetura orientada a modelos; Geração de código; Linguagens de domínio específico; Tradutores.

## ABSTRACT

The demand for technology professionals in various fields has increased sharply in recent years and is projected to continue to grow. In contrast, the training of professionals to occupy these positions is insufficient. As a result, companies resort to hiring people with little or no training in the area and overload senior professionals. Another point that permeates this situation is the poor communication between multidisciplinary teams, as a result of the lack of knowledge of the processes of other areas or the lack of experience of the team members. To assist in the optimization of cloud infrastructure provisioning processes, one of the technological areas lacking professionals, this work develops a platform to simplify communication between different teams with professionals at different levels of expertise, reducing the number of steps necessary for the completion of the process. From an external point of view, these two problems – poor communication and lack of professionals in the market – do not appear to be connected, but together they lead to rework within companies, non-compliance with deadlines, and, above all, high project costs. A visual, intuitive, and automation-based approach is defined to mitigate this scenario and presented in this work. Model Driven Architecture concepts are used to take attention away from the tooling and put it on the problem that the team has to solve. The process is divided into steps that go from a higher and abstract level, where the problem is situated in the real world, to steps belonging to a lower and concrete level, where the problem is transposed to the infrastructure. The entire process is performed by manipulating diagrams for each type of professional or process step and managed by a Domain-Specific Visual Language. The processes that require final artifacts such as documentation or code are automated using the languages built for each purpose.



**Keywords:** Code generation; Domain-specific languages; Model driven architecture; Translator.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ACM | *Association for Computing Machinery* |
| API | *Application Programming Interface* |
| ARGON | *An infRastructure modellinG tool for clOud provisioNing)* |
| ARM | *Azure Resource Manager* |
| AST | *Abstract Syntax Tree* |
| AWS | *Amazon Web Services* |
| CI | *Continuous Integration* |
| CIDR | *Classes Inter-Domain Routing* |
| CIM | *Computing Independent Model* |
| CD | *Continuous Delivery* |
| CPS | *Cyber-Physical System* |
| CPU | *Central Processing Unit* |
| CSS | *Cascading Style Sheets* |
| DOM | *Document Object Model* |
| DSL | *Domain-Specific Language* |
| DSS | *Decision Support System* |
| DSVL | *Domain-Specific Visual Language* |
| EDSL | *Embedded Domain-Specific Language* |
| ESM | *ECMAScript Modules* |
| GMUD | Gestão de Mudanças |
| HCL | *HashiCorp Configuration Language* |
| HTML | *HyperText Markup Language* |
| IaaS | *Infrastructure as a Service* |
| IaC | *Infrastructure aS a Code* |

| | |
|---|---|
| IDE | *Integrated Development Environment* |
| IEEE | *Institute of Electrical and Electronics Engineers* |
| IIoT | *Industrial Internet of Things* |
| IP | *Internet Protocol* |
| IoT | *Internet of Things* |
| JSON | *JavaScript Object Notation* |
| JSX | *JavaScript XML* |
| M2M | *Model To Model* |
| M2T | *Model To Text* |
| MDA | *Model-Driven Architecture* |
| MDD | *Model-Driven Development* |
| MDE | *Model-Driven Engineering* |
| MVC | *Model View Controller* |
| MVP | *Model View Presenter* |
| MVVM | *Model View Viewmodel* |
| OMG | *Object Management Group* |
| PaaS | *Platform as a Service* |
| PIM | *Platform Independent Model* |
| PM | *Product Manager* |
| PO | *Product Owner* |
| PSM | *Platform Specific Model* |
| QoS | *Quality of Service* |
| QP | Questão de Pesquisa |
| RECAP | *Reliable Capacity Provisioning and Enhanced Remediation for Distributed Cloud Applications* |
| REST | *Representational State Transfer* |

| | |
|---|---|
| SaaS | *Software as a Service* |
| SLM | *Service Level Manager* |
| SNET | *Subnet* |
| SQL | *Structured Query Language* |
| TCC | Trabalho de Conclusão de Curso |
| TI | Tecnologia da Informação |
| UML | *Unified Modeling Language* |
| UNIPAMPA | Universidade Federal do Pampa |
| UUID | *Universally Unique Identifier* |
| vCPU | *Virtual Central Processing Unit* |
| VM | *Virtual Machine* |
| VNET | *Virtual Network* |
| XAML | *Extensible Application Markup Language* |
| XMI | *XML Metadata Interchange* |
| XML | *eXtensible Markup Language* |
| YAML | *YAML Ain't Markup Language* |

**SUMÁRIO**

# 1 INTRODUÇÃO

## 1.1 Contextualização e justificativa

O mercado brasileiro de Tecnologia da Informação (TI) sofre, desde sua origem, com escassez de profissionais qualificados[1]. O número de estudantes em todos os cursos da área de Computação corresponde a cerca de 5,5% dos ingressos e 4% do total de matrículas no ensino superior brasileiro (DEED/INEP, 2020; NUNES, 2019). Além do número de cursos superiores não conseguirem formar pessoal suficiente para preencher todas as vagas de trabalho existentes, apresenta-se mais um agravante: profissionais mais experientes estão deixando o país, em busca de melhores oportunidades de trabalho[2]. Nesse cenário, o natural repasse do conhecimento existente no ambiente de trabalho acaba não acontecendo, o que agrava a situação da falta de formação e experiência dos novos contratados.

A já existente escassez de pessoas qualificadas na área acentuou-se mais ainda com a demanda por serviços digitais exigida durante a pandemia de COVID-19, que teve início no ano de 2020. Para suprir as vagas de trabalho existentes, as empresas estão tendo que contratar pessoas com formações incompletas nas áreas da Computação ou mesmo sem formação na área, exigindo treinamento específico para que possam realizar as atividades demandadas pelas organizações. A urgência criada por demandas de clientes faz com que esse treinamento tenha que ser feito de forma rápida e sem o aprofundamento dos conceitos relacionados. Embora essa estratégia resolva parcialmente o problema, acaba gerando retrabalho causado por erros em procedimentos e produtos construídos, o que resulta em custos mais elevados de produção.

Atualmente, trabalhando como líder técnico em uma multinacional especializada em serviços de transformação digital, presencio que a falta de qualificação dos novos colaboradores acaba sobrecarregando alguns membros-chave das equipes. Essas pessoas, além de executar suas funções, também precisam dar suporte para os menos experientes,

[1] ANDRION, R. **Pesquisa prevê carência de 408 mil profissionais de TI até 2022**. 2021. Disponível em: Disponível em: <https://canaltech.com.br/carreira/pesquisa-preve-carencia-de-408-mil-profissionais-de-ti-ate-2022-189998/>. Acesso em: 26 abr de 2022.
ANCKEN, R. V. **Escassez de mão de obra de TI faz empresas buscarem alternativas**. 2022. Disponível em: Disponível em: <https://mercadoeconsumo.com.br/2022/02/09/escassez-de-mao-de-obra-de-ti-faz-empresas-buscarem-alternativas/>. Acesso em: 26 abr de 2022.

[2] ESTADÃO CONTEÚDO. **No radar de empresas estrangeiras, profissionais de TI deixam o País**. 2019. Disponível em: Disponível em: <https://exame.com/carreira/no-radar-de-empresas-estrangeiras-profissionais-de-ti-deixam-o-pais/>. Acesso em: 26 abr de 2022.

incluindo a revisão de seus trabalhos. Não são raros os casos em que, por pressão da data de entrega, o profissional mais experiente acabe realizando quase que a totalidade do projeto, incluindo o levantamento de requisitos, projeto da arquitetura, codificação da arquitetura projetada e execução de sua implantação na nuvem.

A sobrecarga de trabalho pode resultar em estresse e *burn-out* dos profissionais, com impacto na sua saúde e na qualidade do seu trabalho. Essas situações potencializam a existência de erros em todas as atividades realizadas. Projetos de arquitetura com erros, quando são traduzidos em valores, tendem a gerar custos que aumentam diariamente, onerando as contas dos clientes nos provedores de nuvem.

A formação rápida e correta de pessoas é peça-chave na solução dos problemas levantados acima. Em uma tentativa de amenizar essa situação, foi desenvolvido neste trabalho uma plataforma para projetar arquiteturas de infraestrutura em nuvem, cujo código de implantação possa ser gerado automaticamente a partir do projeto realizado, reduzindo cerca de três etapas do processo usado atualmente e simplificando a sua execução, com a vantagem adicional da eliminação dos erros inseridos na fase de codificação do projeto. A abordagem construída usa os princípios de orientação a modelos e de desenvolvimento de linguagens de domínio específico.

As seções seguintes deste capítulo apresentam os objetivos da execução deste trabalho e a organização do texto nos demais capítulos desta monografia.

## 1.2 Objetivos

O objetivo deste trabalho é desenvolver a estrutura base de uma ferramenta para projetar arquiteturas em nuvem de forma intuitiva e que gere o código do carregamento de forma consistente e correta, minimizando o tempo despendido no processo.

São ainda objetivos deste trabalho:

- Analisar as ferramentas de apoio ao desenvolvimento de aplicações web para escolha embasada das tecnologias utilizadas neste trabalho.
- Estudar a especificação e construção de linguagens visuais de domínio específico, que servem de base para a ferramenta de projeto.
- Estudar as linguagens correntes de representação de dados para determinação de qual linguagem intermediária (entre a ferramenta de projeto e a ferramenta de carregamento da descrição na nuvem) melhor se adapta ao problema.

- Revisar os fundamentos teóricos necessários ao trabalho, produzindo um texto que possa ser usado como referência por profissionais interessados.

Como resultados do uso do sistema desenvolvido, espera-se colaborar com procedimentos de carregamento de arquiteturas projetadas em sistemas de nuvem da seguinte forma:

- Diminuir o tempo de provisionamento de recursos selecionados utilizando uma interface intuitiva para colaboradores com baixa experiência.
- Reduzir o esforço no desenvolvimento do projeto de infraestrutura com a utilização de linguagens de domínio específico.
- Exportar o projeto da infraestrutura no formato Terraform para submetê-lo através de esteiras ou armazená-lo em um repositório para uso posterior.

## 1.3 Organização do texto

O restante do texto deste trabalho está organizado como se segue, o Capítulo 2 apresenta o relato de como o processo de pesquisa aplicada foi realizado durante o desenvolvimento da plataforma, incluindo a revisão da literatura para levantamento dos trabalhos relacionados e a descrição das ferramentas usadas pelas empresas e que fazem parte deste projeto.

O Capítulo 3 traz a relação dos trabalhos relacionados ao processos de provisionamento de infraestrutura e a fundamentação teórica sobre os assuntos abordados durante o trabalho como linguagens e metodologias.

No Capítulo 4 é apresentado o escopo do trabalho, onde ele se encaixa dentro do processo produtivo, os detalhes do desenvolvimento da linguagem visual específica de domínio e como ela foi utilizada através de uma interface gráfica.

No Capítulo 5 é realizada uma análise dos resultados obtidos durante a execução do trabalho. As considerações finais sobre o trabalho desenvolvido são apresentadas no Capítulo 6.

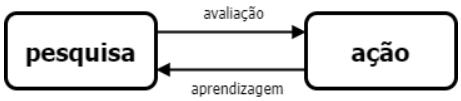
# 2 MATERIAL E MÉTODOS

## 2.1 Caracterização e fases da pesquisa

O trabalho é caracterizado como uma pesquisa aplicada, usando métodos relacionados ao desenvolvimento de *software*, procedimentos de pesquisa-ação e análise de usabilidade (GERHARDT; SILVEIRA, 2009), no que for pertinente. A fundamentação teórico-conceitual e o ferramental usado foram determinados com análise dos trabalhos encontrados usando o método de revisão de escopo (*scope review*) da literatura (MUNN *et al.*, 2018). O processo de execução foi dividido em três etapas: a primeira envolvendo a revisão da literatura relevante, a segunda o projeto e o desenvolvimento da *Integrated Development Environment* (IDE) e das linguagens de domínio específico e a terceira fase consistindo na validação da ferramenta e escrita do texto do trabalho.

A pesquisa-ação é uma abordagem em que o pesquisador não atua como um observador da sua pesquisa, mas como um dos agentes onde a pesquisa é aplicada, mas tomando o cuidado para não manipular os resultados e somente incentivar ou apoiar os participantes. Neste contexto um problema é proposto e submetido a avaliação em campo, com o resultado desta avaliação é aprendido algo, refinado o problema e o ciclo se repete até estabelecer que o problema está resolvido ou que atingiu um nível de maturidade aceitável para uso. Este ciclo pode ser observado na Figura 1.

Figura 1 – Pesquisa-ação

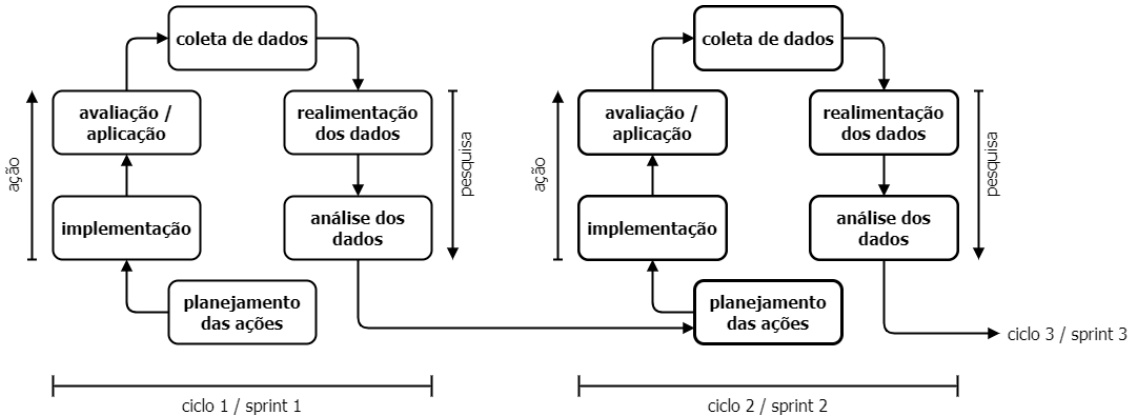



Fonte: Adaptado de Thiollent (2018)

Com a intenção de construir conhecimento em um problema específico de maneira prática e imediata, foi adotado o uso da pesquisa aplicada. Um dos objetivos do trabalho é diminuir o tempo de capacitação de usuários e simplificar o uso da ferramenta é uma das soluções encontradas, para obter uma assertividade maior foi utilizado o procedimento de pesquisa-ação conforme a Figura 2, através de ciclos de testes, mesclando com as *sprints* de desenvolvimento da IDE quando possível. Desta maneira, depois de implementado um conjunto de novas funções ou atualizações a IDE foi submetida a testes na etapa da ação, avançando para a coleta das informações relevantes que foram interpretadas e transpostas para correções necessárias ou novas abordagens na etapa de pesquisa, e o ciclo se reiniciou

para uma nova fase de testes.

Figura 2 – Etapas de testes com o uso de pesquisa-ação




Fonte: Adaptado de Coughlan e Coghlan (2002)

## 2.2 Processo de revisão da literatura

A revisão da literatura faz uso de um protocolo construído com base nos trabalhos de Kitchenham (2004) e de Dermeval e Bittencourt (2020), consistindo das seguintes fases: (*i*) definição das questões de pesquisa da revisão; (*ii*) definição das fontes de pesquisa; (*iii*) definição das *strings* de busca; (*iv*) definição dos critérios de inclusão e exclusão; (*v*) filtragem dos resultados; (*vi*) seleção dos trabalhos encontrados; (*vii*) definição dos critérios de qualidade; (*viii*) revisão e análise dos trabalhos; (*ix*) incorporação de referências; e (*x*) sistematização dos achados (ver Figura 3).

A revisão de escopo, ou mapeamento sistemático da literatura, visa conhecer de forma abrangente o que já existe desenvolvido na área-tema do trabalho e agregar seus resultados em categorias. É um processo de pesquisa em largura, trazendo um número grande de resultados nas buscas, principalmente quando os tópicos possuem poucas evidências presentes na literatura. As seções a seguir apresentam as fases da revisão.

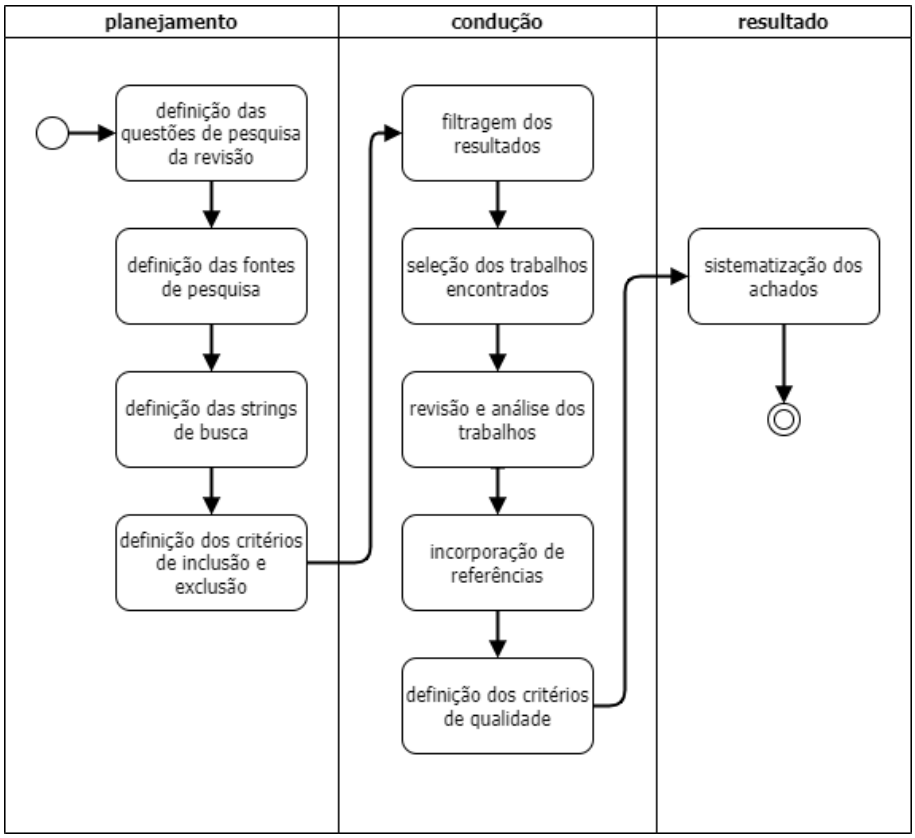
### 2.2.1 Protocolo de revisão

O processo de construção do protocolo e condução da revisão foi feito usando a ferramenta de apoio ao desenvolvimento da revisão da literatura Parsifal (<https://parsif.al/>). Ainda que focada em revisões sistemáticas da literatura para procedimentos

baseados em evidências, a ferramenta auxilia a documentar todo o processo, passando pelas fases de planejamento, condução e resultados. Também desempenha um papel importante em definir os objetivos da revisão, organizar as questões de pesquisa, criar as *strings* de busca e aplicar os critérios de inclusão e exclusão aos trabalhos. Outro ponto importante é o fato de poder importar os trabalhos através de arquivos BibTeX, facilitando relacionar os trabalhos, encontrar duplicados, aplicar a avaliação de qualidade e extrair os dados relevantes para a revisão.

O protocolo construído é descrito a seguir.

Figura 3 – Mapeamento sistemático da literatura




Fonte: Autor (2022)

**Definição do escopo**

O objetivo da revisão bibliográfica é encontrar trabalhos, abordagens ou ferramentas utilizadas para provisionar infraestruturas em provedores em nuvem de forma visual, dando ênfase para trabalhos que utilizem Linguagens Específicas de Domínio (DSL) gráficas, verificando as ferramentas e notações utilizadas e que tenham sido

publicados depois do ano de 2000. A relevância da limitação no ano 2000 é devido ao fato do início do projeto da Amazon Web Services (AWS) permear este ano, o primeiro provedor de nuvem pública comercial.

**Definição das questões de pesquisa da revisão**

Com o foco em encontrar trabalhos que utilizem linguagens visuais, foram formuladas as seguintes questões de pesquisa (QP) para uma revisão de escopo do tipo exploratória:

**QP 1.** Existe algum trabalho ou ferramenta que provisione infraestrutura em nuvem utilizando diagramas?

**QP 2.** Quais ferramentas são utilizadas para o desenvolvimento de uma DSL?

**QP 3.** Quais notações podem ser utilizadas para auxiliar no processo de descrição das etapas do provisionamento?

**Definição das fontes de pesquisa**

Para a definição das fontes de pesquisa foram selecionadas bases de dados que permitissem o uso de palavras-chaves para buscas no repositório e que contivessem estudos na área da Engenharia e da Ciência da Computação. As quatro bases selecionadas foram:

- ACM Digital Library (<http://portal.acm.org>)
- IEEE Digital Library (<http://ieeexplore.ieee.org>)
- ResearchGate (<https://www.researchgate.net>)
- Springer Link (<http://link.springer.com>)

**Definição das *strings* de busca**

Para definir as *strings* de busca, foram utilizados os termos sinônimos aos termos relevantes ao escopo do projeto, listados na Tabela 1. Os termos *"cloud"* e "CPS" não são utilizados em conjunto por *cloud computing* estar contido dentro do domínio de CPS.

De forma geral, os termos mais longos tiveram preferência, para minimizar a quantidade de retornos não relevantes nas buscas.

Foram utilizados os operadores de disjunção lógica ("OR" ou "OU") para incluir sinônimos, conjunção lógica ("AND" ou "E") para adicionar termos distintos e negação lógica ("NOT") para remover termos não relacionados à pesquisa, mas que apareceram nas buscas realizadas. Os termos relacionados para serem removidos na pesquisa atendem ao critério de trazer trabalhos fora do contexto ou que suas siglas coincidiam com a dos termos utilizados no trabalho. Os termos utilizados no operador de negação são:

- *Digital Subscriber Line*,
- *Digital Subscriber Loop*,
- *DMT-Based*,
- *Double Sided Linear*,
- *DSL-SynRM*,
- *Dynamic Spectrum Leasing*,
- *VoDSL*,
- *Voip*.

Tabela 1 – Definição dos termos de busca

| **Termo** | **Sinônimo** |
|---|---|
| Cloud | aws, azure, *cloud computing*, gcp, oci |
| CPS | cyber physical system, cyber-physical system |
| DSL | DSL-based, domain specific language, domain-specific language |
| DSM | domain specific modeling, domain-specific modeling |
| DSVL | domain specific visual language, domain-specific visual language |
| IAC | infraestucture as a code, terraform |

Fonte: Autor (2022)

A Tabela 2 apresenta as *strings* usadas nas buscas dos trabalhos. A primeira é a *string* genérica que foi usada para a construção das demais, na sintaxe específica dos repositórios.

Tabela 2 – Termos de busca por base de dados

| **Base de dados** | **String de busca** |
|---|---|
| genérica | (“CPS” OR “Cyber physical System” OR “Cyber-physical System”) AND (“DSL” OR “DSL-based” OR “domain specific language” OR “domain-specific language”) AND (“DSM” OR “domain specific modeling” OR “domain-specific modeling”) AND (“DSVL” OR “domain specific visual language” OR “domain-specific visual language”) AND (“cloud” OR “AWS” OR “azure” OR “cloud computing” OR “gcp” OR “oci”) AND (“infraestucture as a code” OR “iac” OR “terraform”) NOT (“DMT-based” OR “DSL-SynRM” OR “Digital Subscriber Loop” OR “Double Sided Linear” OR “VoDSL” OR “digital subscriber line” OR “dynamic spectrum leasing” OR “voip”) |
| ACM | AllField:(“Cloud” OR “aws” OR “azure” OR “cloud computing” OR “gcp” OR “oci”) AND AllField:(“IAC” OR “infraestucture as a code” OR “terraform”) AND AllField:(“DSL” OR “DSL-based” OR “domain specific language” OR “domain-specific language”) AND AllField:(NOT “Digital Subscriber Loop” AND NOT “Double Sided Linear” AND NOT “dynamic spectrum leasing” AND NOT “digital subscriber line” AND NOT “DMT-based” AND NOT “DSL-SynRM” AND NOT “VoDSL” AND NOT “voip”) |
| ACM | AllField:(“DSL” OR “DSL-based” OR “domain specific language” OR “domain-specific language”) AND AllField:(“DSM” OR “domain specific modeling” OR “domain-specific modeling”) AND AllField:(“DSVL” OR “domain specific visual language” OR “domain-specific visual language”) AND AllField:(NOT “Digital Subscriber Loop” AND NOT “Double Sided Linear” AND NOT “dynamic spectrum leasing” AND NOT “digital subscriber line” AND NOT “DMT-based” AND NOT “DSL-SynRM” AND NOT “VoDSL” AND NOT “voip”) |
| ACM | AllField:(“CPS” OR “cyber physical system” OR “cyber-physical system”) AND AllField:(“DSM” OR “domain specific modeling” OR “domain-specific modeling”) AND AllField:(“DSL” OR “DSL-based” OR “domain specific language” OR “domain-specific language”) AND AllField:(NOT “Digital Subscriber Loop” AND NOT “Double Sided Linear” AND NOT “dynamic spectrum leasing” AND NOT “digital subscriber line” AND NOT “DMT-based” AND NOT “DSL-SynRM” AND NOT “VoDSL” AND NOT “voip”) |
| ACM | AllField:(“Cloud” OR “aws” OR “azure” OR “cloud computing” OR “gcp” OR “oci”) AND AllField:(“DSVL” OR “domain specific visual language” OR “domain-specific visual language”) |

**Tabela 2 – continuação**

| Base de dados | String de busca |
|---|---|
| IEEE | (("All Metadata":CPS) OR ("All Metadata":cyber physical system) OR ("All Metadata":cyber-physical system)) AND (("All Metadata":DSL) OR ("All Metadata":DSL-based) OR ("All Metadata":domain specific language) OR ("All Metadata":domain-specific language)) AND (("All Metadata":DSM) OR ("All Metadata":domain specific modeling) OR ("All Metadata":domain-specific modeling)) AND (("All Metadata":DSVL) OR ("All Metadata":domain specific visual language) OR ("All Metadata":domain-specific visual language)) NOT ("All Metadata":Digital Subscriber Loop) NOT ("All Metadata":Double Sided Linear) NOT ("All Metadata":dynamic spectrum leasing) NOT ("All Metadata":digital subscriber line) NOT ("All Metadata":DMT-based) NOT ("All Metadata":DSL-SynRM) NOT ("All Metadata":VoDSL) NOT ("All Metadata":voip) |
| IEEE | (("All Metadata":Cloud) OR ("All Metadata":aws) OR ("All Metadata":azure) OR ("All Metadata":cloud computing) OR ("All Metadata":gcp) OR ("All Metadata":oci)) AND (("All Metadata":IAC) OR ("All Metadata":infraestucture as a code) OR ("All Metadata":terraform)) |
| IEEE | (("All Metadata":CPS) OR ("All Metadata":cyber physical system) OR ("All Metadata":cyber-physical system)) AND (("All Metadata":DSVL) OR ("All Metadata":domain specific visual language) OR ("All Metadata":domain-specific visual language)) NOT ("All Metadata":Digital Subscriber Loop) NOT ("All Metadata":Double Sided Linear) NOT ("All Metadata":dynamic spectrum leasing) NOT ("All Metadata":digital subscriber line) NOT ("All Metadata":DMT-based) NOT ("All Metadata":DSL-SynRM) NOT ("All Metadata":VoDSL) NOT ("All Metadata":voip) |
| IEEE | (("All Metadata":Cloud) OR ("All Metadata":aws) OR ("All Metadata":azure) OR ("All Metadata":cloud computing) OR ("All Metadata":gcp) OR ("All Metadata":oci)) AND (("All Metadata":DSL) OR ("All Metadata":DSL-based) OR ("All Metadata":domain specific language) OR ("All Metadata":domain-specific language)) AND (("All Metadata":DSM) OR ("All Metadata":domain specific modeling) OR ("All Metadata":domain-specific modeling)) AND (("All Metadata":DSVL) OR ("All Metadata":domain specific visual language) OR ("All Metadata":domain-specific visual language)) NOT ("All Metadata":Digital Subscriber Loop) NOT ("All Metadata":Double Sided Linear) NOT ("All Metadata":dynamic spectrum leasing) NOT ("All Metadata":digital subscriber line) NOT ("All Metadata":DMT-based) NOT ("All Metadata":DSL-SynRM) NOT ("All Metadata":VoDSL) NOT ("All Metadata":voip) |
| ResearchGate | (“Cloud” OR “aws” OR “azure” OR “cloud computing” OR “gcp” OR “oci”) AND (“DSL” OR “DSL-based” OR “domain specific language” OR “domain-specific language”) |

**Tabela 2 – continuação**

| Base de dados | String de busca |
|---|---|
| Springer Link | Cloud AND domain-specific visual language NOT “Digital Subscriber Loop” NOT “Double Sided Linear” NOT “dynamic spectrum leasing” NOT “digital subscriber line” NOT “DMT-based” NOT “DSL-SynRM” NOT “VoDSL” NOT “voip” |

Fonte: Autor (2022)

### Definição dos Critérios de Inclusão e Exclusão

A etapa de definição dos critérios de inclusão e exclusão foi baseada nas questões de pesquisa, em abordagens relevantes ao projeto e no seu escopo em geral.

#### Critérios de Inclusão

**CI1.** O estudo deve apresentar alguma abordagem sobre a utilização de DSL para provisionar infraestruturas.

**CI2.** Deve poder dar resposta a pelo menos uma questão de pesquisa.

**CI3.** O trabalho deve apresentar uma aplicação de *Model Driven Enginnering* que se aplique no contexto do projeto.

#### Critérios de Exclusão

**CE1.** Ter sido publicado antes do ano de 2000.

**CE2.** Ser duplicado com outro já selecionado.

**CE3.** Trabalhos em qualquer outro idioma que não seja o Inglês.

### Filtragem dos resultados

Com as *strings* de busca e os critérios de inclusão e exclusão definidos, o processo de filtrar os trabalhos acontece quase que por inteiro dentro das áreas de pesquisa das bases de dados. As *strings* de busca retornam poucos resultados, normalmente entre dois e dezesseis trabalhos, são aplicados os CE1 e CE3 diretamente nos filtros das pesquisas, reduzindo mais os resultados disponíveis. Com uma análise rápida dos títulos e palavras-chave dos trabalhos, pode-se verificar mais facilmente os itens de inclusão e selecionar somente os mais viáveis, já que a amostra de trabalhos é muito reduzida.

Conforme é apresentado na Tabela 3, após o processo de filtragem é selecionado oitenta e cinco trabalhos, que em sua grande maioria são da ACM e IEEE (36 e 31 trabalhos respectivamente). A base de dados ResearchGate é utilizada pelo motivo de outras bases de dados retornar poucos resultados, como a *Springer Link* com um trabalho considerado viável.

Tabela 3 – Resultados da busca

| **Base de dados** | **Trabalhos** | **Porcentagem** |
|---|---|---|
| ACM Digital Library | 36 | 42,4% |
| IEEE Digital Library | 31 | 36,5% |
| ResearchGate | 17 | 20% |
| Springer Link | 1 | 1,2% |
| **Total importados** | 85 | 100% |

Fonte: Autor (2022)

### 2.2.2 Condução da revisão

#### Seleção dos trabalhos encontrados

Para classificar os trabalhos que passaram pelos critérios de inclusão, exclusão e filtragem, foi definido um conjunto de Critérios de Qualidade (CQ). Para a definição desses critérios foram levados em consideração três aspectos importantes: o desenvolvimento da DSL, abordagens de provisionamento de infraestruturas em nuvem e sua usabilidade. No que tange a DSL, trabalhos que descrevam como e onde elas são desenvolvidas ou se tiveram um tratamento especifico para resolver um dado problema, favorecem a sua escolha. Quanto ao provisionamento de recursos, o meio que a implantação ocorre e a pluralidade de recursos são levados em consideração, mas o mais relevante é como são solucionados os impedimentos e diferenças entre modelos de serviços e provedores de nuvem diferentes. Outro ponto de atenção é a preocupação que o estudo selecionado tem com o usuário final, seu empenho em simplificar ou abstrair conceitos complexos, guiando seu uso pretendendo diminuir a curva de aprendizagem e obtendo resultados significativos com uma diminuição do esforço.

São definidos sete critérios de qualidade, cada um atendendo a um ponto específico da pesquisa e auxiliando a definir os trabalhos finais considerados viáveis. Cada critério é avaliado com uma pontuação mínima de 0.0 (zero) e máxima de 1.0 (um) e um

estudo pode atingir uma pontuação final máxima de 7.0 (sete).

**Critérios de Qualidade (CQ)**

**CQ1.** O estudo possui alguma demonstração de uso, seja por vídeo ou prática?

Avaliação:

**Sim:** [1.0] Possui uma demonstração.

**Não:** [0.0] Não possui uma demonstração.

**Parcialmente:** [0.5] Possui imagens ou outro tipo de representação que demonstre seu funcionamento.

**CQ2.** O estudo apresentou uma ferramenta que apoie no desenvolvimento de uma DSL?

Avaliação:

**Sim:** [1.0] Apresenta pelo menos uma ferramenta que auxilie no desenvolvimento de uma DSL.

**Não:** [0.0] Não apresenta nenhuma ferramenta para desenvolver DSL.

**Parcialmente:** [0.5] Apenas citou alguma ferramenta, mas não demonstrou seu uso.

**CQ3.** O estudo apresentou alguma notação que auxilie em uma etapa do trabalho?

Avaliação:

**Sim:** [1.0] Apresentou pelo menos uma notação útil ao escopo do trabalho.

**Não:** [0.0] Não apresentou nenhuma notação.

**Parcialmente:** [0.5] Mencionou seu uso, mas não a demonstrou ou relatou sua relevância.

**CQ4.** O estudo demonstrou o desenvolvimento da DSL?

Avaliação:

**Sim:** [1.0] Apresentou as etapas utilizadas na criação de uma DSL.

**Não:** [0.0] Não demonstrou como a DSL foi criada.

**Parcialmente:** [0.5] Apresentou somente um diagrama de relacionamento ou trechos de código da DSL.

**CQ5.** O estudo tem foco em provisionamento de infraestrutura em nuvem?

Avaliação:

**Sim:** [1.0] O estudo é focado em provisionamento de recursos em nuvem.

**Não:** [0.0] Não é voltado para provisionamento de recursos em nuvem.

**Parcialmente:** [0.5] Uma parte do projeto provisiona recursos em nuvem.

**CQ6.** O estudo tem como foco o provisionamento de algum recurso específico de infraestrutura, cobre todos os recursos ou é de uso genérico?

Avaliação:

**Recurso específico de infraestrutura:** [0.5] O trabalho é focado no provisionamento de um único recurso na nuvem.

**Cobre os recursos de infraestrutura:** [1.0] O trabalho abrange no mínimo o provisionamento dos principais recursos em nuvem.

**É de uso genérico:** [0.3] Não tem foco em nenhum recurso, somente na demonstração de como o provisionamento é realizado.

**CQ7.** O estudo tem alguma preocupação com a usabilidade ou utiliza uma interface adaptativa?

Avaliação:

**Sim:** [1.0] O estudo possui relatos de uso ou estudos de usabilidade da ferramenta.

**Não:** [0.0] Não é mencionada em nenhum momento a preocupação de interação do usuário com a ferramenta.

**Parcialmente:** [0.5] Há menções de como é esperado o comportamento do usuário.

### Revisão e análise dos trabalhos

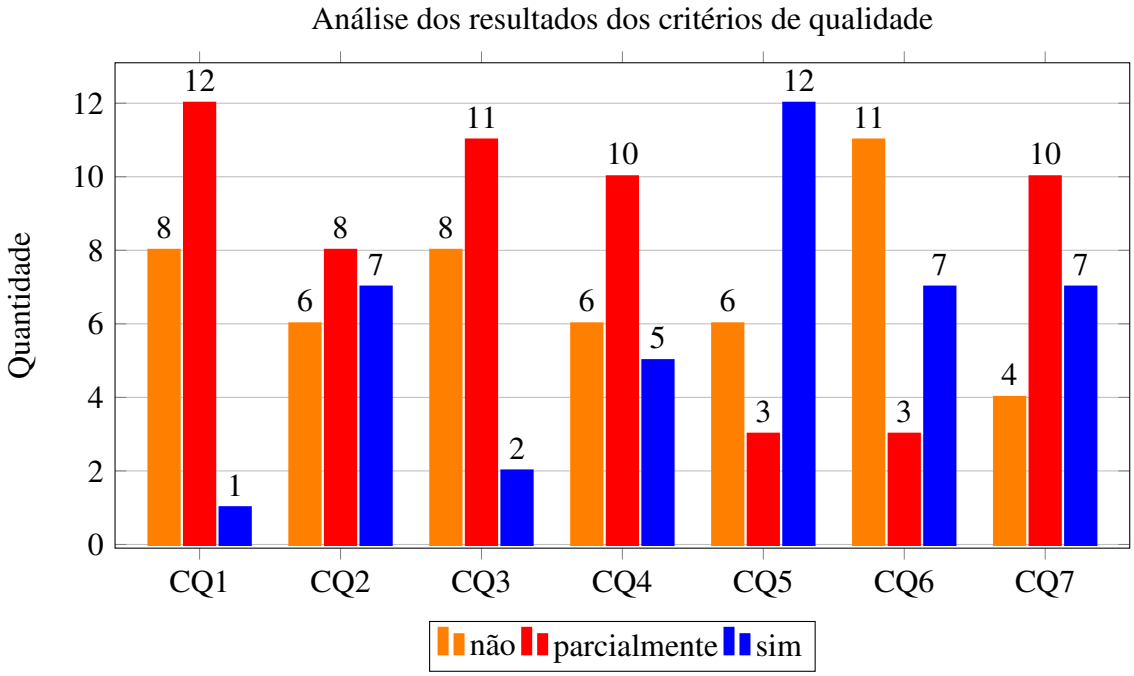
Após concluir os processos de triagem dos trabalhos e submetê-los à critérios de avaliação, foram selecionados vinte e um trabalhos que estão relacionados na Tabela 4. Ao analisar a pontuação dos trabalhos levando em consideração os critérios de avaliação, a nota mais elevada foi 4,8 de um total de 7. Mesmo com os trabalhos retornando notas pouco acima de 50%, eles se tornam relevantes em sua completude, onde alguns solucionam lacunas deixadas por outros.

Para uma compreensão melhor, a Figura 4 apresenta uma análise quantitativa das respostas dos critérios de qualidade. Os primeiros quatro critérios referem-se ao ferramental utilizado nos trabalhos enquanto os três restantes no provisionamento em nuvem e usabilidade. O primeiro critério (CQ1) questiona se o projeto possui uma

demostração de uso da proposta e, em sua grande maioria, os trabalhos não apresentam demonstrações ou somente imagens de suas interfaces. O segundo (CQ2) e o terceiro (CQ3) critérios que indagam o uso de ferramentas de apoio no desenvolvimento da DSL e notação, respectivamente, neste caso a maioria dos trabalhos relatam as ferramentas superficialmente, não informando detalhes de versão, configurações ou limitações de uso. O quarto critério (CQ4) é referente à demonstração da criação dos elementos da DSL, apesar de apresentar fragmentos de sua criação em alguns dos trabalhos, os que apresentam não possuem sua abordagem de provisionamento, mas o comportamento da DSL pode ser identificado.

Nos critérios cinco (CQ5) e seis (CQ6) é avaliado se o trabalho realmente tem foco em provisionamento e, se afirmativo, qual a sua abrangência em relação aos recursos que provisiona. Neste ponto deve-se ter atenção que alguns trabalhos foram relacionados para suprir limitações de outros e seis dos trabalhos do critério cinco não são diretamente relacionados a provisionamento, impactando desta maneira nos onze trabalhos elegidos no critério seis que são considerados genéricos. No último critério (CQ7), é solicitada a preocupação quanto à interação com o usuário e foi percebida tentativas de facilitar seu uso, mas sempre de forma muito custosa para o autor do trabalho.

Figura 4 – Resultado após a aplicação dos critérios de qualidade




Fonte: Autor (2022)

Tabela 4 – Pontuação da qualidade dos trabalhos

| ID | Título | Referência | CQ1 | CQ2 | CQ3 | CQ4 | CQ5 | CQ6 | CQ7 | PF |
|---|---|---|---|---|---|---|---|---|---|---|
| 01 | CloudMF: Model-Driven Management of Multi-Cloud Applications | Ferry *et al.* (2018) | 0.5 | 1.0 | 0.5 | 1.0 | 1.0 | 0.3 | 0.5 | 4.8 |
| 02 | An Infrastructure Modelling Tool for Cloud Provisioning | Sandobalin, Insfran e Abrahao (2017) | 0.5 | 0.5 | 0.5 | 0.5 | 1.0 | 1.0 | 0.5 | 4.5 |
| 03 | A Chrestomathy of DSL Implementations | Schauss *et al.* (2017) | 1.0 | 1.0 | 0.5 | 1.0 | 0.0 | 0.3 | 0.5 | 4.3 |
| 04 | CloudMap: A Visual Notation for Representing and Managing Cloud Resources | Weerasiri Denisand Barukh (2016) | 0.5 | 0.0 | 1.0 | 0.5 | 1.0 | 0.3 | 1.0 | 4.3 |
| 05 | A Modelling Language for Defining Cloud Simulation Scenarios in RECAP Project Context | Morais *et al.* (2018) | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 1.0 | 1.0 | 4.0 |
| 06 | ARGON: A Model-Driven Infrastructure Provisioning Tool | Sandobalin, Insfran e ao (2019) | 0.5 | 0.5 | 0.5 | 0.0 | 1.0 | 1.0 | 0.5 | 4.0 |
| 07 | End-to-End Automation in Cloud Infrastructure Provisioning | Sandobalín, Insfran e Abrahão (2017) | 0.5 | 0.0 | 0.5 | 0.5 | 1.0 | 1.0 | 0.5 | 4.0 |
| 08 | Generating Domain-Specific Visual Language Editors from High-level Tool Specifications | Grundy *et al.* (2006) | 0.5 | 1.0 | 0.5 | 0.5 | 0.0 | 0.3 | 1.0 | 3.8 |
| 09 | An Empirical Study on Visual Programming Docker Compose Configurations | Piedade, Dias e Correia (2020) | 0.5 | 0.5 | 0.5 | 0.0 | 0.5 | 0.5 | 1.0 | 3.5 |
| 10 | MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds | Ardagna *et al.* (2012) | 0.5 | 0.5 | 0.0 | 0.0 | 1.0 | 1.0 | 0.5 | 3.5 |
| 11 | On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric | Sandobalín, Insfran e Abrahão (2020) | 0.5 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 3.5 |
| 12 | Automated Migration of EuGENia Graphical Editors to the Web | Rani *et al.* (2020) | 0.5 | 1.0 | 0.5 | 0.5 | 0.0 | 0.3 | 0.5 | 3.3 |
| 13 | CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications | Ferry *et al.* (2014) | 0.0 | 0.5 | 0.5 | 0.5 | 1.0 | 0.3 | 0.5 | 3.3 |

**Tabela 4 – continuação**

| ID | Título | Referência | CQ1 | CQ2 | CQ3 | CQ4 | CQ5 | CQ6 | CQ7 | PF |
|---|---|---|---|---|---|---|---|---|---|---|
| 14 | Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications | Grundy *et al.* (2013) | 0.5 | 1.0 | 0.0 | 1.0 | 0.0 | 0.3 | 0.5 | 3.3 |
| 15 | HorusCML: Context-aware domain-specific visual languages designer | Almorsy, Grundy e Rüegg (2014) | 0.0 | 1.0 | 0.5 | 0.5 | 0.0 | 0.3 | 1.0 | 3.3 |
| 16 | Model-Based DSL Frameworks | Kurtev *et al.* (2006) | 0.5 | 1.0 | 0.5 | 1.0 | 0.0 | 0.3 | 0.0 | 3.3 |
| 17 | The (5+1) Architectural View Model for Cloud Applications | Hamdaqa e Tahvildari (2014) | 0.0 | 0.5 | 0.0 | 0.5 | 1.0 | 0.3 | 0.5 | 2.8 |
| 18 | DSL Methods for CPS Simulation in the Cloud: Experience Report | Kourzanov (2013) | 0.0 | 0.5 | 0.0 | 1.0 | 0.5 | 0.5 | 0.0 | 2.5 |
| 19 | End-User-Oriented Tool Support for Modeling Data Analytics Requirements | Khalajzadeh *et al.* (2020) | 0.0 | 0.0 | 0.0 | 0.5 | 0.5 | 0.5 | 1.0 | 2.5 |
| 20 | On MODAClouds' Approach for Application Design and Execution on Multi-clouds | Nitto, Solberg e Petcu (2014) | 0.0 | 0.5 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 2.5 |
| 21 | A Model-Driven Approach for Promoting Cloud PaaS Portability | Silva, Fortes e Lucrédio (2013) | 0.0 | 0.0 | 0.0 | 0.5 | 1.0 | 0.3 | 0.0 | 1.8 |

Fonte: Autor (2022)

**Incorporação de referências**

Com a análise dos trabalhos, houve a descoberta de outros estudos e autores, por meio das referências usadas nos artigos lidos, ampliando o conhecimento sobre o tema ou trazendo novas abordagens para a solução dos problemas. Com relação aos sistemas de provisionamento, os projetos *Reliable Capacity Provisioning and Enhanced Remediation for Distributed Cloud Applications* (RECAP) e MODAClouds possuem a relação das suas publicações no *Community Research and Development Information Service* (CORDIS), o principal repositório público de projetos de pesquisa da Comissão Europeia. O projeto MODAClouds produziu um livro (NITTO *et al.*, 2017) apresentando o resultado do estudo trazendo, além do provisionamento, mais detalhes e outras preocupações, tais como análises financeiras e adaptações de projetos.

Com relação à construção de linguagens e, particularmente, sobre DSL, livros e artigos dos autores Martin Fowler, Anneke Kleppe e Ralf Lämmel foram recorrentemente encontrados. Martin Fowler foi bastante citado como referência no desenvolvimento de DSL. No caso, a linguagem de especificação *Unified Modeling Language* (UML), e em relação a como criar estas linguagens obedecendo padrões de arquiteturas. Anneke Kleppe, além da sua abordagem do uso da DSL e de UML, é conhecida pela elaboração de algumas teorias sobre o uso do *Model Driven Architecture* (MDA). Ralf Lämmel tem seus trabalhos referenciados no desenvolvimento de DSL e *Domain Specific Visual Languages* (DSVL), trazendo toda uma categorização do assunto e estudos sobre a criação dos metamodelos de linguagens.

Foi incorporada também a documentação publicada na *Object Management Group* (OMG) sobre a especificação do uso de MDA e formatos necessários para a sua aplicação.

### 2.2.3 Análise da revisão

Ao total foram importados das bases de dados oitenta e cinco trabalhos, quatro deles foram eliminados por já constarem em outra base, outros sessenta foram rejeitados e sendo aceitos vinte e um trabalhos, como mostra na Tabela 5. Os trabalhos aproveitados foram aceitos levando em consideração os seguintes aspectos: projetos correlatos ou com um foco semelhante, abordagens que supram as limitações dos projetos correlatos ou semelhantes, abordagens DSL e DSVL que venham a agregar no desenvolvimento da ferramenta que não foram exemplificadas ou ficaram vagas nos projetos correlatos ou

semelhantes.

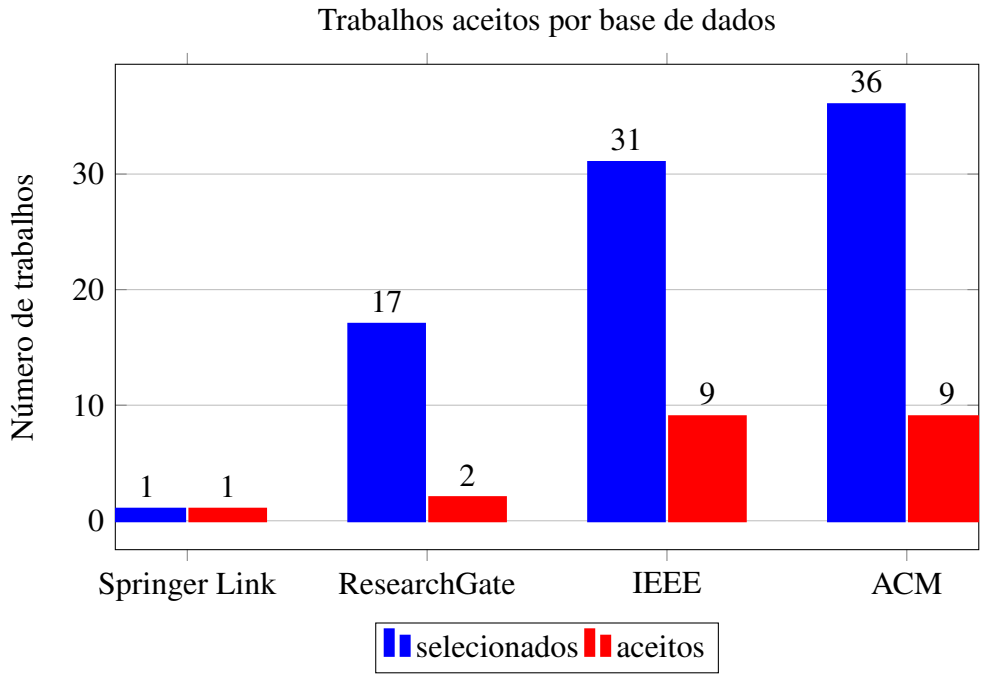
Tabela 5 – Resultados dos trabalhos

| **Trabalhos** | **Quantidade** | **Porcentagem** |
|---|---|---|
| Aceitos | 21 | 24.7% |
| Rejeitados | 60 | 70.6% |
| Duplicados | 4 | 4.7% |
| **Total importados** | 85 | 100% |

Fonte: Autor (2022)

Os fatores que levaram a rejeitar os trabalhos foram diversos: em relação a projetos foram descartados os que possuíam elementos de IoT, Indústria 4.0 ou se em outro projeto já se apresenta a resolução do problema de maneira mais completa e atualizada. Foram descartados também trabalhos onde as construções das DSL não atendessem aos propósitos deste trabalho ou que envolvesse uma complexidade desnecessária para o resultado final pretendido. Projetos de provisionamento que não utilizavam algum tipo de DSL, fossem totalmente focados em *scripts* ou somente em ferramentas de *pipelines*, foram descartados.

Figura 5 – Trabalhos aceitos

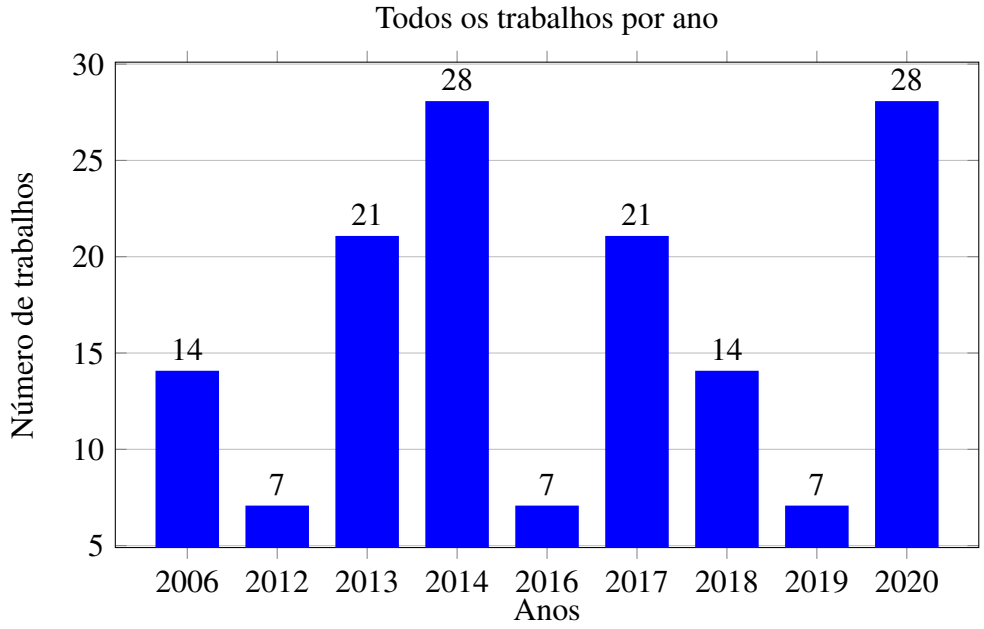



Fonte: Autor (2022)

Com relação as bases de dados (ver Figura 5), foram selecionados da IEEE e ACM nove trabalhos em cada e tiveram trinta e um e trinta e seis trabalhos selecionados

cada. Com números menos expressivos, a ReserarchGate teve dois trabalhos aceitos e a Springer Link somente um. Encontrar projetos relevantes foi difícil. Os poucos trabalhos encontrados foram realizados em sua grande maioria na Europa e dois deles receberam incentivos da própria União Europeia, sendo eles o MODAClouds e o RECAP. Um ponto que chamou a atenção foi que o número de artigos publicados relacionados a provisionamento em provedores de nuvem teve um aumento conforme a divulgação dos resultados dos projetos MODAClouds e RECAP, outubro de 2012 a setembro de 2015 e janeiro de 2017 a dezembro de 2019, respectivamente, apesar de apresentar uma amostra pequena, os trabalhos importados refletem sutilmente este comportamento (ver Figura 6).

Figura 6 – Trabalhos por ano de publicação




Fonte: Autor (2022)

## 2.3 Ferramental tecnológico

Dependendo do foco de atuação em alguma etapa do trabalho, o uso de um conjunto de ferramentas se fez necessário. Algumas tecnologias foram usadas por serem padrão de mercado ou por já estarem sendo utilizadas no contexto de aplicação deste trabalho, outras por dependência de alguma tecnologia que foi utilizada, ou ainda por características que atendem demandas essenciais. No decorrer do desenvolvimento do projeto, outras ferramentas foram necessárias, tendo suas descrições incorporadas a esta seção e estão listadas a seguir.

### Terraform

O Terraform (WINKLER, 2021) é uma ferramenta de provisionamento de infraestrutura como código desenvolvido pela empresa Hashicorp, que pode ser utilizada tanto para infraestruturas em nuvem como nas instalações locais. Seu funcionamento básico prevê a criação de manifestos declarativos baseados na linguagem *HashiCorp Configuration Language* (HCL), onde é realizada a descrição da infraestrutura desejada e logo após pode ser provisionada. Dois pontos são relevantes salientar sobre esta ferramenta: o primeiro é que ao submeter o provisionamento à algum provedor, ele transcreve o arquivo HCL em um grafo no formato *JavaScript Object Notation* (JSON) (HINZE, 2016), a declaração em HCL é utilizada para fins de facilitar a criação por um ser humano e podendo ser descartada em processos de automação mais severos; o segundo ponto é que para cada provedor de nuvem que se for utilizar é necessário usar um *provider* diferente (BRIKMAN, 2019) em que os *providers* funcionam como extensão da linguagem, algo semelhante ao artefato Platform Specific Model (PSM) do MDA.

### JavaScript e Vue

Para o desenvolvimento do *frontend* e do *backend* da plataforma são utilizadas tecnologias baseadas em *JavaScript*, pela sua flexibilidade e suporte a vários *frameworks* e bibliotecas. No *frontend*, o *framework* de interface *Vue* (<https://vuejs.org/>) e no *backend*, o *NodeJS* (<https://nodejs.org/>). Simultaneamente, nos dois ambientes se faz o uso do *TypeScript* (<https://www.typescriptlang.org/>), para garantir uma tipagem fortemente estática, e do JSON, que é um formato de transferência de dados leve e de fácil interpretação por seres humanos.

O *Vue* é um *framework JavaScript* utilizado no desenvolvimento de componentes para interfaces de usuário de forma declarativa, em que suas principais características são a reatividade e a progressividade. A reatividade diz respeito a sua arquitetura que atualiza a camada visual a cada mudança em seu estado e a progressividade remete ao seu projeto ser altamente adaptável a outros *frameworks* e bibliotecas, sendo possível utilizá-lo em uma parte da aplicação ou trabalhar em conjunto com a parte visual de um *framework* de *backend*. Um ponto que o difere dos outros *frameworks* do gênero é a utilização dos *single file components*, em que os seus componentes são criados a partir de um único arquivo com a extensão Vue que contém todas as informações necessárias e divididas

em áreas específicas para o *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e o *JavaScript*. Como os arquivos com extensão .vue não são interpretados pelos navegadores, se faz necessário a utilização de um *bundler* como o *Webpack* para transformá-lo em um arquivo *JavaScript*. Seu foco é na camada *ViewModel* do padrão arquitetural Model View ModelView (MVVM), se conectando as camadas *View* e *Model* por meio de dois *data bindings*, podendo também manipular o virtual *Document Object Model* (DOM) através de filtros e diretivas, trabalhar com modelos e utilizar *JavaScript* XML (JSX) e *TypeScript*.

Em conjunto com o *framework* Vue são explorados o compilador *BabelJS* (<https://babeljs.io/>) e o *bunbler Webpack* (<https://webpack.js.org/>). O *BabelJS* é um compilador que a partir de um código escrito em uma versão mais atual do *JavaScript*, o compila em um outro código *JavaScript* compatível com os a maioria dos navegadores e suas versões. Desta forma, é possível utilizar os recursos mais novos da linguagem mesmo que os navegadores sejam incompatíveis. Resumidamente, a compilação ocorre em três etapas: *parser*, *transformer* e *generator*. O *parser* transforma o código em uma estrutura de dados AST (*Abstract Syntax Tree*) que mapeia todos os elementos do código, sendo manipulada pelo *transformer* de maneira que atenda os requisitos para ser compatível com os navegadores atuais e por fim o *generator* para transformar o AST em um outro código *JavaScript*.

O *Webpack* é um *bundle*, ou seja, um agrupador ou empacotador de arquivos *JavaScript* para ser utilizado em programação modular. Dividir a aplicação em módulos simplifica a realização de *debugs* e testes, em que cada módulo deve ter os seus propósitos bem definidos no contexto geral da aplicação. O *Webpack* faz o gerenciamento das dependências destes módulos a partir do *dependency graph*, uma árvore de dependências responsável pela organização, percorrendo os seus arquivos, os verificando e resolvendo as dependências gerando ao final uma arquivo *JavaScript*. Também são suportados métodos dos sistemas modulares *CommonJS* e ESM (*ECMAScript Modules*) e pode ser estendido para dar suporte a imagens, CSS e fontes.

O *NodeJS* é um interpretador que possibilita o uso do *JavaScript* fora de um navegador, tendo sido desenvolvido com base no V8 *engine* do *Google* e atualmente é utilizado em projetos na nuvem que precisam de um início rápido, baseado em evento e de maneira assíncrona.

**Azure Resource Manager API REST**

O Azure *Resource Manager* API REST (ARM API REST) (<https://docs.microsoft.com/pt-br/azure/azure-resource-manager/>) é a interface de programação de aplicação do provedor de nuvem Azure responsável por receber as solicitações de provisionamento e gerencia o processo de implantação destes recursos. A solicitação pode ser realizada pelo portal da Azure, pelo *PowerShell*, Azure *Command Line Interface* (AZ CLI) ou submetendo um objeto JSON com a descrição do que deve ser provisionado.

# 3 FUNDAMENTAÇÃO TEÓRICA

## 3.1 Provisionamento de infraestruturas em nuvem

Para entender como o provisionamento de infraestruturas em nuvem funciona, inicialmente é necessário compreender quais são os modelos de computação, de serviços e de implantação em nuvem. Os modelos de computação dizem respeito ao local onde o processamento dos dados será realizado; modelos de serviços dizem respeito ao quanto é necessário ter controle sobre os recursos de infraestrutura; e os modelos de implantação dizem onde é decidido se os recursos de *hardware* podem ou não ser compartilhados com outros usuários no provedor de nuvem.

Os modelos de computação em nuvem mais conhecidos (MAHMOOD; RAMACHANDRAN, 2018) são *edge computing* (computação em borda), *cloud computing* (computação em nuvem), *fog computing* (computação em névoa) e *sky computing*. No modelo de *edge computing* o processamento dos dados ocorre nos dispositivos locais, como em projetos de IoT, IIoT, cidades inteligentes, entre outros. Em um outro cenário possui o modelo de *cloud computing*, onde o processamento ocorre em *datacenters* espalhados pelo mundo e podendo ser acessados pela Internet ou por links dedicados; neste caso temos os provedores de nuvem como a Azure e a AWS. Em um meio termo entre os dois modelos anteriores temos o *fog computing*, onde o processamento pode ocorrer nos dispositivos locais, na nuvem ou em dispositivos intermediários chamados de *gateways* que estão entre a borda e a nuvem. Nesse modelo, o local onde o processamento irá ocorrer vai depender dos requisitos do projeto. Não é incomum em projetos desse tipo existir a necessidade de distribuir a carga nos três modelos para diminuir a latência. Por último, temos o modelo *sky computing*, em que o processamento não ocorre somente em um provedor de nuvem, mas em vários provedores simultaneamente, algumas vezes por questões de conformidade, custos, latência ou por causa de algum serviço específico.

Os modelos de serviços mais comuns (FEHLING *et al.*, 2014) são *on premises* (instalações locais), *infrastructure as a service* (IaaS ou infraestrutura como serviço), *platform as a service* (PaaS ou plataforma como serviço) e *software as a service* (SaaS ou software como serviço). No modelo *on premises*, também conhecido como legado, toda a infraestrutura é provisionada nas instalações do cliente e todo o gerenciamento de *hardware* e *software* é de sua responsabilidade. Esse modelo está caindo em desuso

por vários motivos, incluindo custos de adequações às conformidades impostas por lei ou segurança. Em um modelo IaaS, o gerenciamento do *hardware* é realizado pelo provedor em nuvem que consiste em dispositivos de rede, armazenamento, servidores e virtualizadores, restando os sistemas operacionais, configurações de rede, dados e aplicações que o cliente deve gerenciar. Em um modelo PaaS, o cliente somente fica responsável pela segurança e sustentação dos dados e aplicações, o restante fica a cargo do provedor de nuvem. Em um modelo SaaS, o cliente fica responsável somente em alimentar a aplicação, não tendo responsabilidade alguma pelo gerenciamento e sustentação de nenhuma parte do ambiente.

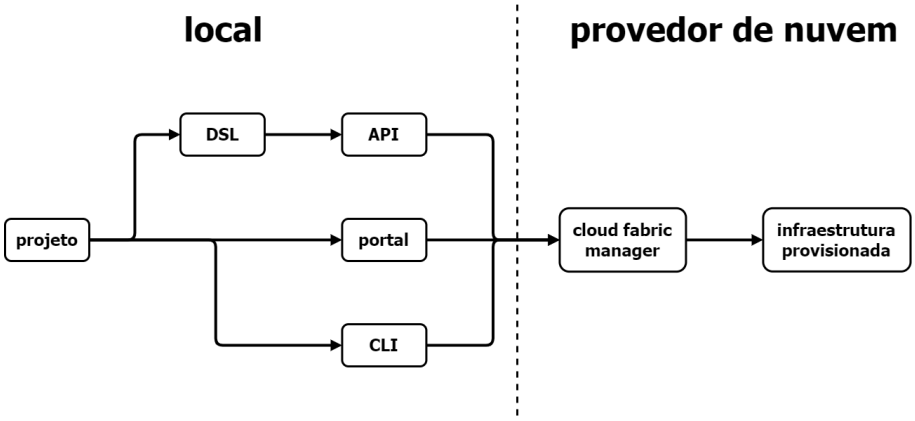
Os modelos de implantação em nuvem mais comuns são *private cloud* (nuvem privada) e *public cloud* (nuvem pública). Em uma nuvem privada os recursos são reservados para um único cliente, ao qual é destinada toda uma área do *datacenter*. Esse modelo garante mais segurança e possibilidades de personalizações, mas tem um custo financeiro mais elevado. No modelo de nuvem pública, a infraestrutura de *hardware* é compartilhada com outros clientes, gerando custos mais baixos e até a possibilidade de serviços gratuitos, mas com uma menor flexibilidade de configurações e um nível de segurança menor quando comparado à nuvem privada.

Um provedor de nuvem é uma empresa que disponibiliza recursos em nuvem sob demanda no modelo de computação *cloud computing*. Em sua grande maioria, essas empresas oferecem serviços de IaaS, PaaS ou SaaS, em nuvens públicas ou privadas. Existem atualmente vários provedores de nuvem atuantes no mercado, que disponibilizam seus serviços globalmente como a Azure e a AWS.

Depois de selecionados os modelos de computação, serviço e implantação, é necessário definir os recursos desejados e escolher como ocorrerá o provisionamento. As três principais formas de provisionamento (MORRIS, 2021) são por *application programming interface* (API ou interface de programação de aplicação), pelo portal do provedor de nuvem ou por *command line interface* (CLI ou interface de linha de comando). Quando se provisiona utilizando API, um objeto é enviado para o provedor de nuvem – normalmente um arquivo no formato da linguagem de descrição de dados JSON – no qual é descrita toda a infraestrutura a ser provisionada. Em alguns casos, para facilitar a criação deste arquivo JSON, são utilizadas linguagens de domínio específico (*domain-specific languages* ou DSL) que recebem parâmetros da infraestrutura e geram o arquivo de descrição no formato correto. Outra maneira de provisionar os recursos é através do portal do provedor em nuvem, onde se pode realizar a configuração

dos recursos um a um, por meio de interfaces dedicadas para cada tipo de recurso e submetê-las ao provisionamento. Em alguns cenários onde são realizadas automações por meio de *scripts*, também podem ser provisionados os recursos por CLI. Nesse caso, antes de submeter o *script* para provisionamento, é necessário realizar uma autenticação de acesso e garantir que o terminal que irá executar o *script* tenha visibilidade ao provedor de nuvem. O processo pode ser visualizado na Figura 7: depois de elegida a forma de provisionamento, as informações são enviadas ao *cloud fabric manager* do provedor de nuvem que interpreta os dados e dispara ações para cada componente responsável por provisionar os recursos.

Figura 7 – Principais formas de provisionamento em nuvem

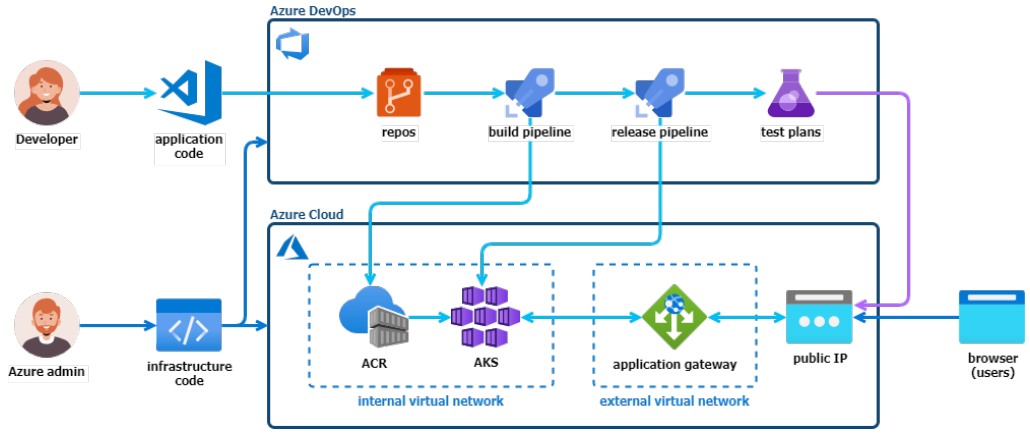



Fonte: Autor (2022)

Definido como e onde será feito o provisionamento, restam estabelecer quais recursos serão provisionados. O processo de submissão de um projeto para provisionamento em nuvem pode ser melhor entendido pelo fluxo simplificado apresentado na Figura 8, usado para disponibilizar uma aplicação na Internet. Nesse caso, temos uma pessoa responsável por desenvolver o código da aplicação em si (*developer*) e um responsável por desenvolver o código que vai provisionar a infraestrutura necessária para executar a aplicação (*Azure admin*). Normalmente, após o *developer* codificar a aplicação, o *Azure admin* provisiona uma esteira para compilar e testar a aplicação (*Azure DevOps*) e também os componentes de rede e servidores responsáveis por executar a aplicação (*Azure Cloud*).

Depois do engenheiro de nuvem ou arquiteto de software definir a estrutura do que será provisionado, evidenciando suas interligações, a equipe responsável necessita transcrever em forma de código o projeto. Depois de codificado, o projeto é convertido em um grafo por um compilador e submetido para a uma plataforma de nuvem no formato

Figura 8 – Fluxo simplificado para disponibilização de uma aplicação na nuvem




Fonte: Adaptado de Wendel (2020)

JSON, na qual a arquitetura é provisionada. Ao projetar a arquitetura, pode-se gerar o grafo diretamente, sem a necessidade de transcrevê-lo em código e, dependendo do grau de complexidade, diminuir a dependência do compilador.

As etapas do projeto de infraestrutura preveem um número considerável de pessoas envolvidas, incluindo equipes de projeto e sustentação de ambiente, equipes de gerenciamento de serviços, vendas e produto. A Tabela 6 apresenta a descrição do processo, incluindo as tarefas e as responsabilidades dos colaboradores que estão diretamente ligados ao foco deste trabalho. A codificação e a documentação são os dois itens do projeto que demandam mais atenção e tempo das equipes.

Tabela 6 – Etapas de um projeto de infraestrutura em nuvem

| Item | Tarefa | Responsável | Sucessor |
|---|---|---|---|
| 1 | Levantamento de requisitos | Equipe de projetos | item 2 |
| 2 | Projetar infraestrutura | Arquiteto ou engenheiro | item 3 |
| 3 | Codificar infraestrutura | Equipe de projetos | item 4 |
| 4 | Provisionamento da infraestrutura | Equipe de projetos | item 5 |
| 5 | Plano de implementação / migração / virada | Equipe de projetos e sustentação | item 6 |
| 6 | Implementação da infraestrutura | Equipe de sustentação | item 7 |
| 7 | Plano de continuidade | Equipe de sustentação | - |

Fonte: Autor (2022)

## 3.2 *Model-Driven Architecture*

*Model-Driven Architecture*, ou simplesmente MDA, é uma metodologia com foco no processo de desenvolvimento de *software*. A ideia fundamental da abordagem é que o ser humano pensa de uma maneira muito mais abstrata do que um programa, que é o elemento concreto do processo de desenvolvimento de software. Modelos são expressos por linguagens e a ideia é que modelos abstratos sejam sucessivamente refinados, com apoio de ferramentas para um processo semiautomatizado, até que o modelo seja tão simples que possa ser traduzido diretamente em código. MDA, dependendo do nível de abstração usado por uma tradução, também pode ser referido como *Model-Driven Engineering* (MDE) ou *Model-Driven Development* (MDD).

A relevância das abordagens orientadas a modelos é encontrada na conversão de processos e fluxos de trabalho abstratos em código. O estado do modelo – usualmente uma máquina de estados – é consultado e modificado a cada alteração realizada, permitindo que alterações em um nível mais abstrato sejam traduzidas nos níveis mais concretos subsequentes, assegurando que o código final seja compatível com o projeto original, o que nem sempre acontece nos processos de desenvolvimento de software. Adicionalmente, qualquer documentação gerada no processo também é alterada, assegurando a consistência interna dos artefatos gerados.

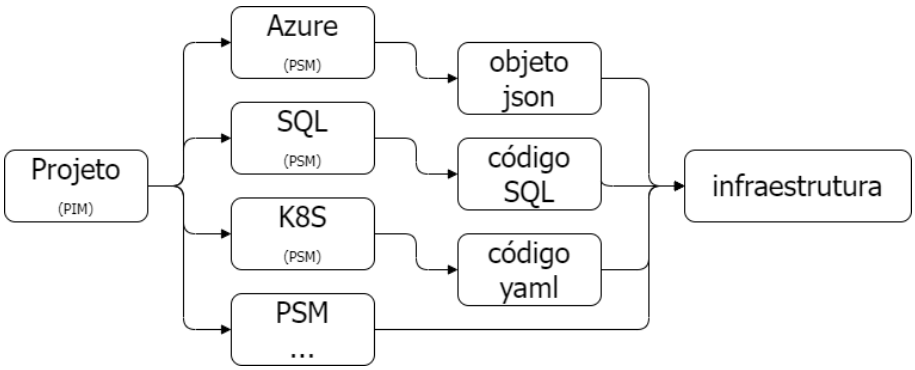
Nessa metodologia, o código é apenas mais um modelo desenvolvido, refinado a partir de um modelo mais abstrato. Dependendo do ponto de vista, não existe um código formal propriamente e sim a marcação das associações internas do modelo (KLEPPE; WARMER; WARMER, 2003). Para compreender como estas associações interagem entre si, transformando o abstrato em algo concreto, este método possui uma estrutura básica composta por artefatos, transformadores, ferramentas e visões.

Os artefatos produzidos pelo processo são camadas específicas de abstração e, conforme o desenvolvimento avança, o processo de refinamento transforma a abstração em algo cada vez mais concreto. Essas camadas são conhecidas como CIM, PIM e PSM (MELLOR *et al.*, 2005) e o avanço entre elas obedece a exatamente essa ordem. O primeiro artefato a ser concebido é o *Computation Independent Model* (CIM) que abrange elementos sobre o modelo do negócio ou algum outro domínio específico, contando com a representação de instâncias do mundo real, como por exemplo pessoas, lugares ou leis e não do sistema. Ao definir o modelo do mundo real, ele deve ser transposto para um modelo lógico mais próximo dos sistemas de informação, sendo esse

modelo denominado *Plataform Independent Model* (PIM). Nesse nível de abstração os componentes são descritos de forma conceitual e não específica. Por exemplo, uma máquina virtual deve ser especificada pelos seus requisitos de *hardware*, mas não um modelo específico de máquina virtual. O modelo nesse nível não pode ser dependente de uma plataforma ou serviço específico e deve poder ser transposto para qualquer um. Nessa etapa também, com a intenção de atingir o seus objetivos, é descrito como os seus componentes influenciam os outros componentes ou até mesmo pessoas e organizações. No último artefato, o *Plataform Specific Model* (PSM), deve ser possível implementar o PIM em uma tecnologia ou plataforma particular. Esse modelo possui uma profunda dependência do PIM, onde para cada tecnologia elegida no projeto é necessário um PSM específico para ela.

A Figura 9 apresenta melhor a dependência entre o PIM e o PSM: ao conceber um modelo PIM que estabelece que será provisionado um ambiente em nuvem, com a criação e alimentação de tabelas em um banco de dados e a implantação de um sistema de orquestração de contêineres, é necessário definir quais tecnologias serão provisionadas pelos PSM. No exemplo, foi definido que o provedor de nuvem será o Azure e terá de possuir um PSM específico para ele, o banco de dados foi definido como SQL (*Structured Query Language*) e o orquestrador será utilizado o *Kubernetes*, para estes dois últimos também será necessário possuir PSM específicos para tratar deles. Após passar pelos PSM resultará nos elementos necessários para provisionar a estrutura, sejam códigos ou objetos, o que a plataforma ou tecnologia final necessitar.

Figura 9 – Exemplo de utilização dos artefatos PIM e PSM




Fonte: Autor (2022)

Para realizar a transição de um artefato para outro é necessário que haja um transformador (ou tradutor) específico. Nesse caso, é necessário identificar os componentes do modelo de origem que serão utilizados no modelo de destino. No

exemplo da Figura 9, são encontrados transformadores do tipo *PIMtoPSM* e *PSMtoCode* (SANDOBALIN; INSFRAN; ABRAHAO, 2017). O primeiro localiza as estruturas necessárias de uma tecnologia no PIM e constrói um novo modelo PSM com elas. Por exemplo, são identificados componentes de provisionamento em nuvem e são gerados modelos de provisionamento para o PSM de Azure, identificam-se modificações em banco de dados e gera-se o modelo do PSM de SQL e assim sucessivamente. Os transformadores de *PSMtoCode* geram os entregáveis finais de todo o processo de abstração em um código ou objeto concreto e pronto para provisionamento vindos de um PSM específico, como é o caso do objeto JSON e os códigos SQL e YAML.

A abordagem MDA exige ferramentas para cada etapa do processo de desenvolvimento, tais como modeladores, editores e repositórios. Os modeladores, também conhecidos como *designers*, são utilizados para projetar os modelos e suas interações por meio de interfaces gráficas ou texto. Os editores são ferramentas para alterar os modelos projetados anteriormente, que podem ser de forma livre ou projetual. Os repositórios são um conjunto de PSM, códigos ou transformadores que a abordagem MDA precisa ter à disposição para atuar. Quanto mais vastos forem os repositórios de PSM, por exemplo, mais tecnologias ou plataformas podem ser elegidas para uso e, consequentemente, mais transformadores e códigos terá em seu repositório para atender estes PSM.

Quanto à organização e orquestração dos artefatos, transformadores e ferramentas, a metodologia MDA utiliza uma abordagem de visões sobre o *software*, que pode ser estática, dinâmica ou ambas. A visão estática está relacionada aos objetos e às instâncias utilizadas no modelo, podendo ser armazenada como um diagrama de classe, por exemplo, mantendo a sua estrutura de associação. A visão dinâmica é um pouco mais complexa, que com um diagrama de estados ou sequência, reproduz como o *software* se comporta, como ele deve atuar em determinadas condições. Por exemplo, ao criar algum novo elemento deve-se verificar se ele já existe e, em caso afirmativo, como deve ser tratada essa situação.

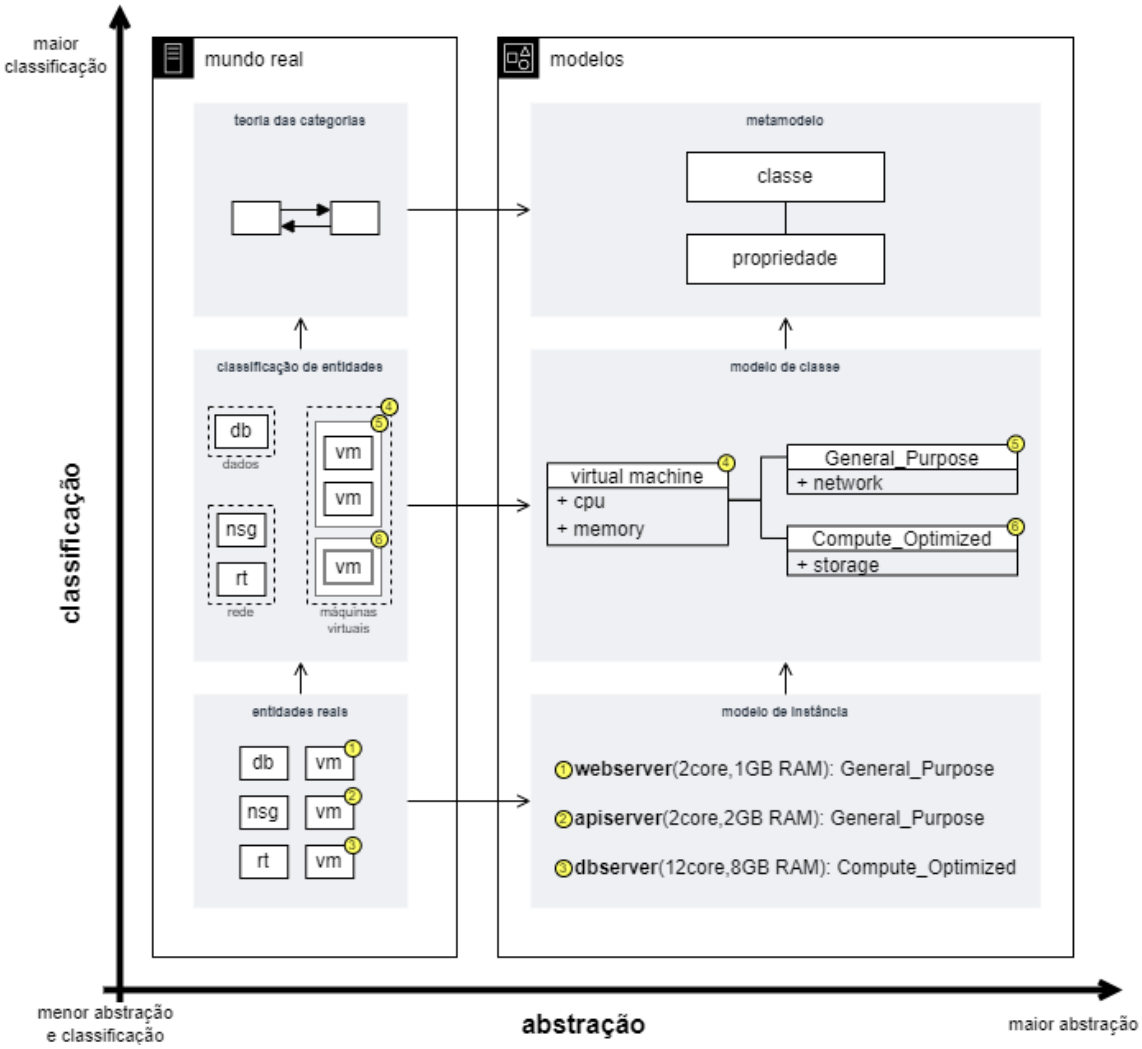
Uma parte importante do processo de modelagem, que auxilia no refino da abstração é a manipulação assertiva dos metamodelos. De uma maneira concisa, um metamodelo é utilizado para especificar de forma simples os conceitos de uma linguagem, suas funções de mapeamento e a comunicação entre outros modelos, sendo resultante de um processo de abstração, classificação e generalização de um domínio específico.

Um metamodelo é um modelo da linguagem de modelagem (MELLOR *et al.*,

2005), em que uma classe é uma subclasse do classificador, capturando os seus conceitos de propriedades e podem ser transpostos para uma linguagem de modelagem, como a UML, por meio de operações, atividades e estados.

A Figura 10 exemplifica um processo que resulta na obtenção de um metamodelo, em que pode ser observado como as entidades do mundo real são classificadas pelas suas características mais relevantes para o domínio e abstraídas ou representadas através de modelos, é obtida uma classe generalizada que pode atender aos mais diversos casos no domínio e simplificando a comunicação com outros modelos.

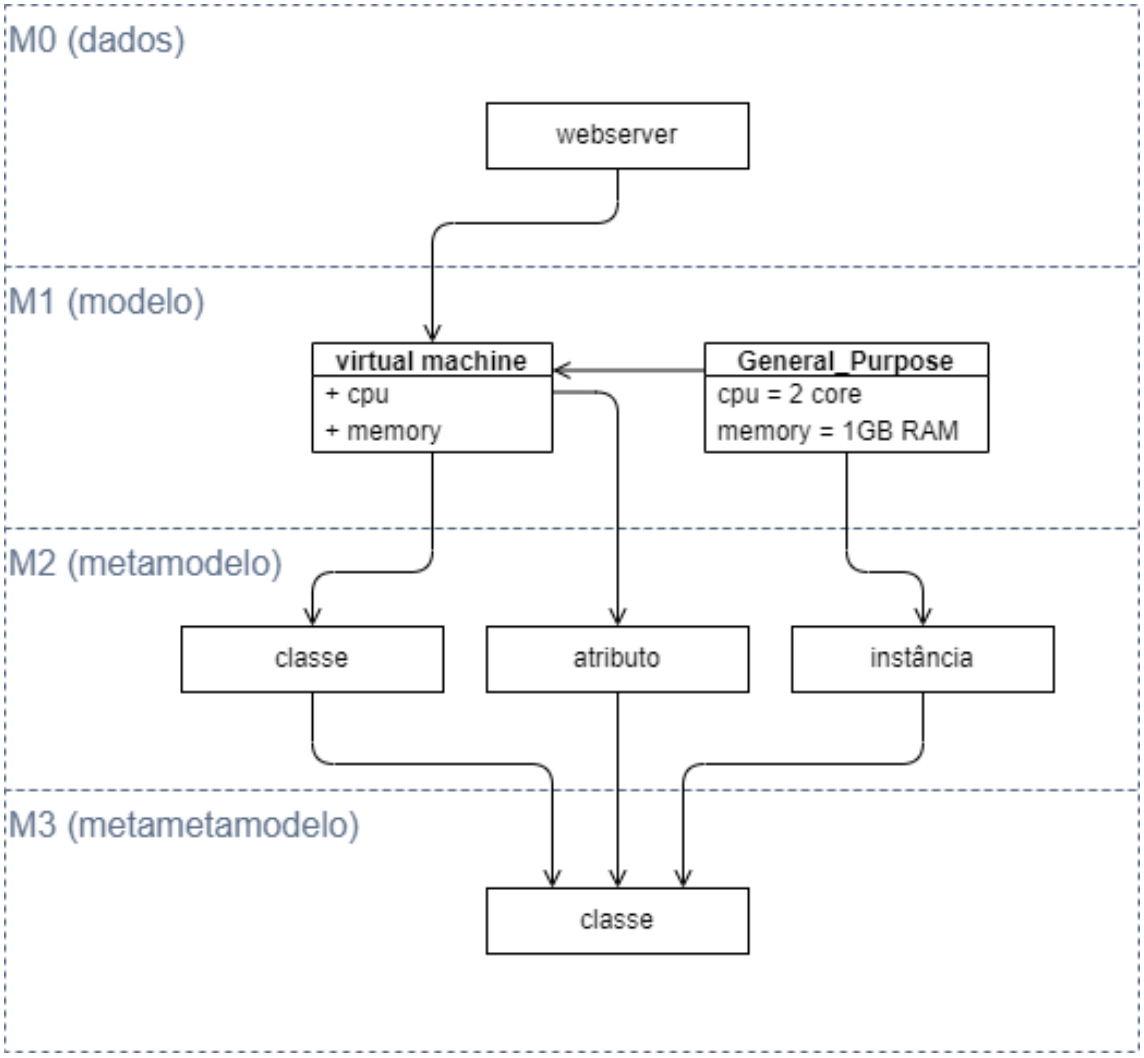
Figura 10 – Metamodelo em relação a modelos de desenvolvimento




Fonte: Adaptado de (MELLOR *et al.*, 2005)

A OMG padronizou esse processo e a relação entre os objetos, modelos e metamodelos, definindo uma arquitetura em quatro níveis meta (M0, M1, M2 e M3) e suas metarelações consiste no relacionamento entre os níveis (ver Figura 11). O nível

Figura 11 – Hierarquia de metamodelo de quatro níveis




Fonte: Adaptado de (MELLOR *et al.*, 2005)

M0 são as instâncias em tempo de execução, ou seja, os dados da aplicação sendo manipulados, o nível M1 é onde a modelagem da aplicação é realizada, contendo o modelo do usuário e a aplicação propriamente, e o nível M2 possui os metadados da linguagem de modelagem e tem a responsabilidade definir a arquitetura da aplicação. O nível M3 possui um conjunto de conceitos mais simples exigidos para identificar os modelos e metamodelos, consistindo em classes cujas instâncias podem estar associadas a instâncias de outras classes, possuir atributos de algum tipo e efetuar operações. Como o nível M3 descreve o paradigma da modelagem, seus modelos também são denominados como metametamodelos.

Uma aplicação primária para metamodelos é habilitar a definição de transformação de modelos, cujas instâncias são tipos em um outro modelo permitindo

manipulá-lo. Para construir essas transformações ou mesmo realizar metarelações, é preciso entender a linguagem de modelagem na qual os modelos de origem e de destino são expressos, levando em consideração a relação entre acoplamento e coesão.

Os termos acoplamento e coesão possuem conceitos bastantes simples, mas de difícil tratamento. Antes de conceituar estes termos, é necessário definir modularização ou componentização e sua relação com seus elementos. O objetivo da modularização é desacelerar a entropia de software, antes que o sistema se torne incontrolável devido a sua complexidade, ou seja, ter sistemas complexos com uma manutenção mais amena possível. A medida que a aplicação cresce, seu desenvolvimento se torna mais lento e a implementação de modelos mentais precisos através de módulos pequenos e independentes são mais viáveis e favoráveis que módulos grandes e interdependentes.

Uma das principais características de um módulo é a capacidade de ocultar de forma abstrata as suas funcionalidades e interfaces, podendo ser importado e exportado para outros projetos sem estar subordinado as suas dependências. Mas os projetos tendem a possuírem muitos módulos e estes necessitam estar interligados, o desafio é evitar ou abrandar o acoplamento entre estes módulos.

O termo acoplamento (PFLEEGER, 2004) está relacionado a quão dependente um módulo ou elemento é de algum outro para executar corretamente. Idealmente, quanto mais desacoplado ou fracamente conectado a outros módulos for sua implementação, mais flexível, portável e manutenível ele se torna. Existem alguns tipos de acoplamentos mais conhecidos, que serão citados pelo seu grau de acoplamento, sendo o primeiro o mais fracamente acoplado até o último considerado o mais fortemente acoplado.

O *data coupling* é utilizado quando uma função envia dados com uma mesma interpretação para outra função para ser utilizado em cálculos, o *stamp coupling* quando a assinatura de uma classe tem outra classe como argumento ou tipo de retorno, o *control coupling* quando uma função controla o fluxo de outra função, o *external coupling* quando o código depende do comportamento de terceiros, o *common coupling* quando compartilha o estado global ou regras de negócio com outros componentes e o *content coupling* quando uma classe modifica o conteúdo de outra classe.

Enquanto o acoplamento pode ser definido pelas suas conexões entre diferentes módulos, a coesão é sobre as conexões dentro dos limites de um módulo. Idealmente, a coesão ou o direcionamento do módulo deve ser definido antes de qualquer acoplamento com outro módulo e este sendo definido no decorrer ou no final do desenvolvimento do módulo. Um código ou módulo é definido com alta coesão, quando seus elementos

formam um todo funcional, com seus métodos focados na classe atendendo o problema do domínio proposto. É definido com baixa coesão, quando a classe e seus métodos realizam uma grande variedades de ações e geralmente sem um foco comum ou atendendo um problema específico do domínio. Funcionalidades dentro de um módulo que são incoerentes a sua proposta ou lógica, devem ser transferidas para outros módulos, desta forma, se for necessário realizar alguma alteração neste módulo, não será necessário modificações em outros módulos e suas interfaces.

Como no acoplamento, existem alguns tipos de coesões que são citadas iniciando pela de mais alta coesão indo até a de mais baixa coesão (PFLEEGER, 2004). A *function cohesion* é quando cada elemento essencial para uma única computação esta contido no componente, executando as tarefas e as funções, a *sequence cohesion* tem relação ao elemento que emite dados que se tornam a entrada para outro elemento, a *communicational cohesion* onde dois elementos operam nos mesmos dados de entrada e contribuem para os mesmos na saída, a *procedural cohesion* garante a ordem de execução de ações que estão fracamente conectadas e improváveis de serem utilizadas, a *temporal cohesion* onde os elementos estão relacionados pelo tempo envolvido e todas as tarefas são executadas no mesmo intervalo de tempo, a *logical cohesion* onde os elementos estão relacionados logicamente e não funcionalmente e a *coincidental cohesion* onde os elementos não estão relacionados conceitualmente e sendo a pior forma de coesão.

Sintetizando, ao aumentar a coesão se reduz o acoplamento, pois tudo que se precisa está dentro do próprio módulo e reduz a necessidade de conexões com outros módulos.

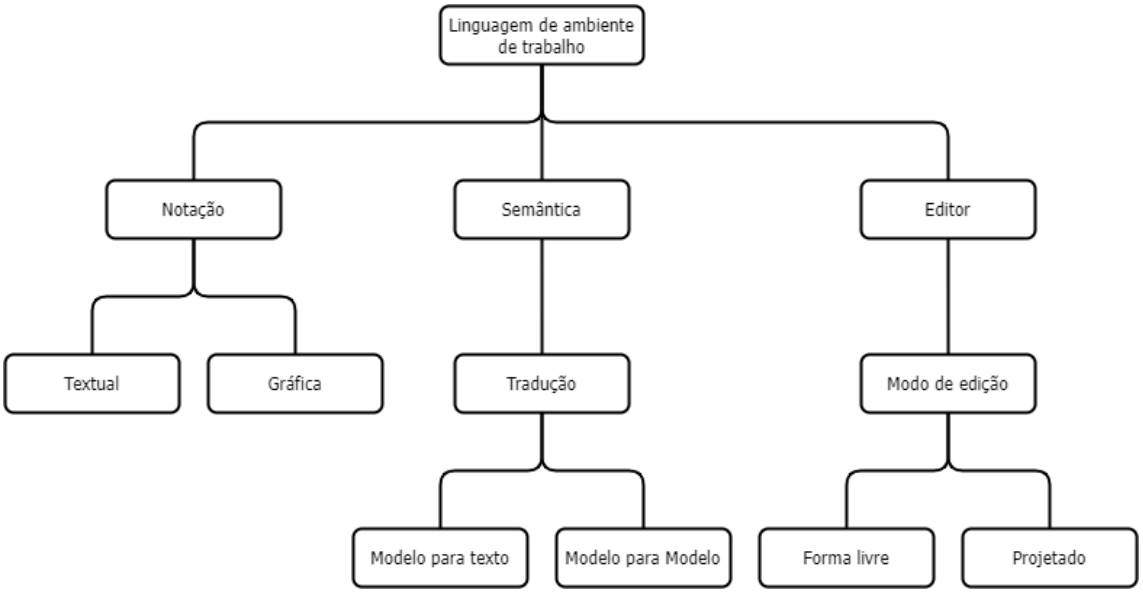
## 3.3 *Domain-specific languages*

Linguagens orientadas a um domínio de aplicação, conhecidas como DSL, são linguagens de modelagem ou programação que visam atender a um tipo específico de problema, ao contrário das chamadas linguagens de propósito geral, que possuem uma sintaxe genérica capaz de representar a solução de qualquer problema computável (FOWLER, 2010). Esse tipo de linguagem é muito mais comum do que o nome aparenta: CSS e HTML são dois exemplos de DSL muito utilizadas para formatação de páginas em navegadores de Internet, SQL é uma DSL para consultas a banco de dados relacionais, *Graphviz* é uma DSL para construção e visualização de grafos e LaTeX é uma DSL para formatação de texto.

Abordagens orientadas a modelos, em qualquer dos seus níveis de abstração, frequentemente fazem uso de DSL para especificações. As razões para isso são várias, mas a principal é que uma DSL pode capturar exatamente o que aquele nível de abstração necessita, sem faltas ou excessos. Como DSL não necessitam ser Turing-completas, verificação de propriedades de modelos podem ser feitas por algoritmos exatos e eficientes, minimizando ou mesmo eliminando a necessidade de testes posteriores no código gerado.

Um domínio específico pode ser compreendido como um conjunto de conceitos e regras que são associados à descrição de uma solução para um certo tipo e problema. O papel de uma DSL é abstrair as soluções desse tipo de problema em um formato de objetos e classes para utilizá-los da maneira mais apropriada possível. Em um contexto mais amplo, todo o conjunto de ferramentas para o desenvolvimento dessa nova linguagem é conhecido como *language workbench* (ERDWEGD *et al.*, 2015), em que cada ferramenta ou grupo de ferramentas é responsável por uma parte da linguagem. Os três principais e obrigatórios recursos da *language workbench* são notação, semântica e edição, conforme a Figura 12.

Figura 12 – Estrutura básica de uma *language workbench*




Fonte: Adaptado de (LÄMMEL, 2018)

A notação define como será a interação do usuário com a linguagem. Essa interação pode ser, por exemplo, de maneira textual ou gráfica. Quando a notação for textual, o usuário envia um conjunto de *strings* com os comandos necessários para o *parser* da linguagem, que o interpreta, ativa as funções apropriadas e executa o solicitado.

Quando a notação é gráfica, normalmente o usuário manipula um grafo ou diagrama onde, dependendo das alterações realizadas, são ativadas as funções apropriadas para executar o solicitado. DSL visuais são chamadas de DSVL (sigla para *Domain-Specific Visual Language*). Note-se que, em qualquer dos casos, a notação precisa ser gerada a partir de um gerador formal, que usualmente é uma gramática. Gramáticas livres de contexto (LEWIS; PAPADIMITRIOU, 1998) são comumente usadas para definição de linguagens textuais enquanto gramáticas de grafos (ZHANG; ZHANG, 2003; EHRIG *et al.*, 1997) são usadas para a definição sintática de linguagens visuais.

O usuário manipula a linguagem pela notação, mas é a definição da semântica da linguagem que assegura que a tradução é realizada de forma correta. Dependendo do contexto, pode-se ter como resultado dessa tradução um texto ou um outro modelo. De maneira resumida, toda a base da linguagem é modelada com objetos e classes, como mencionado anteriormente, gerando um modelo que pode ser estruturado em forma de grafo, árvore ou outra estrutura apropriada (LÄMMEL, 2018). A tradução vai ocorrer utilizando esse modelo, acrescida de parâmetros de entrada. Se a tradução for do tipo modelo-para-texto, o seu resultado é uma concatenação de *strings*; se for do tipo modelo-para-modelo, o resultado é um novo grafo, árvore ou outra estrutura pertinente. Note-se que as linguagens-alvo da tradução também devem ser especificadas formalmente, podendo ser outras DSL ou não.

No centro da utilização da linguagem está o suporte à edição, onde de fato as notações são manipuladas e pode ocorrer principalmente de forma livre ou projetada. Quando editado de forma livre, o usuário pode modificar o modelo da maneira que achar apropriado, mas com o ônus de precisar garantir que alterações que prejudiquem o funcionamento do modelo não sejam realizadas. Para evitar danificar o modelo pode-se utilizar a edição projetual, na qual o usuário manipula um modelo dentro das restrições sintáticas estabelecidas no editor. Dessa forma, é possível evitar erros tanto sintáticos que inviabilizarão a tradução quanto erros semânticos, como ligações entre objetos que não podem ser ligados, por exemplo, antecipando erros que somente serão detectados na fase de testes da aplicação.

## 3.4 Trabalhos relacionados

Antes da pandemia de 2020, poucos trabalhos propondo o provisionamento de infraestruturas em múltiplos provedores de nuvem foram publicados e, em escala

ainda menor, os que abordam o problema de maneira visual. A partir do método delineado na Seção 2.2, foram selecionados quatro projetos e algumas abordagens sobre provisionamento em nuvem que possuem aspectos que refletem a proposta deste trabalho.

O *Reliable Capacity Provisioning and Enhanced Remediation for Distributed Cloud Applications* (RECAP) (MORAIS *et al.*, 2018) é um projeto financiado pela União Europeia que visa desenvolver a próxima geração de provisionamento de recursos *cloud/fog/edge* e informa de maneira generalizada os desafios técnicos e financeiros deste tipo de abordagem. Esse estudo propõe uma linguagem específica de modelagem baseada no paradigma de *Model-Driven Development* (MDD), denominada SML (*Simulation Modelling Language*), para auxiliar no provisionamento avançado de recursos e simulação por meio de uma interface gráfica. A interface permite caracterizar como as conexões entre os elementos são realizadas, utilizando uma notação própria.

Outro projeto financiado pela União Europeia é o *MOdel-Driven Approach* (MODAClouds) (ARDAGNA *et al.*, 2012) que, por meio de sua IDE, visa auxiliar desenvolvedores e operadores no ciclo de vida de projetos baseados em nuvem. Ele integra projetos secundários, como o sistema de tomada de decisão, para selecionar qual provedor de nuvem é mais adequado para o projeto e o MODACloudML em que relaciona os modelos e metamodelos para serem utilizados. O núcleo do seu processo de provisionamento acontece em três níveis: um nível inicial, detalhando o fluxo de trabalho a ser provisionado, seguindo por um detalhamento dos recursos e finalizando com a codificação do projeto. O projeto aborda os principais desafios considerados para a migração entre IaaS/PaaS/SaaS: os bloqueios tecnológicos, o gerenciamento de riscos e a garantia da qualidade. Também define pontos que devem ser avaliados em termos de negócio durante o provisionamento, como a economia de custos, a avaliação de risco, o *quality-of-service* (QoS) e a flexibilidade no processo de desenvolvimento.

O *Cloud Modelling Framework* (CloudMF) (FERRY *et al.*, 2014) é um projeto que serviu como base para vários outros, incluindo o MODAClouds. Ele consiste em uma DSL para provisionamento em múltiplos provedores de nuvem chamada *Cloud Modelling Language* (CloudML) e uma ferramenta denominada *models@runtime* responsável pelo provisionamento, implantação e adaptação da infraestrutura em nuvem. O *framework* suporta serviços IaaS e PaaS, pode ser utilizado por uma interface gráfica ou de forma textual, além de oferecer suporte remoto para outras ferramentas. O motivo de servir de base para outros trabalhos se deve principalmente por três pontos: traz suporte aos formatos *XML Metadata Interchange* (XMI) e JSON, é estruturado em modelos da OMG

e sua abordagem é baseada em requisitos.

Com uma visão diferente, o projeto *An infRastructure modellinG tool for clOud provisioNing* (ARGON) (SANDOBALIN; INSFRAN; ABRAHAO, 2017) realiza o provisionamento utilizando conceitos de DevOps e não de processos, como os três projetos anteriores. Sua DSL tem como resultado gerar *scripts* que atendem a três estágios utilizados em Infrastructure as Code (IaC): integração contínua, implantação contínua e entrega contínua. Possui uma interface gráfica que tem como base o projeto EuGENia e onde a infraestrutura é visualizada por meio de um diagrama ou por uma estrutura hierárquica de árvore. Outro aspecto relevante é a fácil identificação do uso do *Model-Driven Architecture* através dos artefatos PIM e PSM, e seus transformadores M2M (*Model to Model*) e M2T (*Model to Text*).

Além dos quatro projetos supracitados, outros trabalhos encontrados na revisão visam resolver o mesmo problema de provisionamento em nuvem. Silva, Fortes e Lucrédio (2013) foca na portabilidade dos serviços de IaaS, PaaS e SaaS entre os diferentes provedores de nuvem e nas dificuldades enfrentadas nas suas migrações. Estas diferenças mapeadas auxiliam na criação de uma linguagem intermediária mais adaptativa para diferentes contextos ou fluxos de trabalho.

Com a meta de produzir evidências do impacto que uma abordagem visual pode ter no desenvolvimento de infraestrutura como código, Piedade, Dias e Correia (2020) destacam dois tópicos em seu trabalho: como as técnicas de *model-driven engineering* são utilizadas para identificar os problemas envolvendo o domínio da infraestrutura em nuvem, e entender quais abordagens de programação visual podem ser utilizadas no serviço de orquestração. É definida uma DSL que possa atuar independentemente do provedor e seu resultado produz um *script* para a ferramenta alvo. Para atender ao que se propõe, sua arquitetura tem três componentes: o *Docker Model*, o *Executing Environment* e o *Connector*. O *Docker Model* é utilizado para modelar os *containers* e suas dependências, o *Execute Environment* tem como alvo a infraestrutura que o *container* irá necessitar e o *Connector* trabalha como uma ponte e provê o sincronismo entre os dois componentes anteriores. Para validar seu trabalho foi realizado um estudo experimental que mensura a real redução de tempo que uma ferramenta visual de IaC tenha durante a utilização. Ao final foi relatado que indivíduos utilizando a ferramenta visual, mas sem nenhum treinamento prévio, e indivíduos com um certo conhecimento na linguagem e implementando a mesma infraestrutura proposta através de codificação textual, tiveram tempos de entrega praticamente iguais. O diferencial constatado foi que ao utilizar a

ferramenta visual observou-se uma redução do tempo utilizado em pesquisa, na consulta a documentações e validações pelo terminal, frente à codificação textual.

Com um foco maior em análise de dados e na automação do processo de aprendizagem de máquina, o trabalho de Khalajzadeh *et al.* (2020) apresenta relevância na abordagem das lacunas apresentadas na rastreabilidade de projetos de implementação e a comunicação deficiente entre os *stakeholders*. Segundo o estudo, os dois problemas citados anteriormente são causados pela característica das equipes atuais serem multidisciplinares, acarretando em uma diferença de conhecimento e experiência entre os profissionais da área de negócio, gerenciamento, cientistas de dados, engenheiros, arquitetos e desenvolvedores. A solução encontrada é a criação de uma linguagem comum para descrever as soluções, de forma visual e descrita através de cinco tipos de diagramas diferentes: dois diagramas de alto nível (*brainstorm* e diagrama de processos) e três de baixo nível (dados, técnicos e diagramas de provisionamento). Cada diagrama inclui uma coleção de notações para modelar diferentes objetos, suas relações e regras. Cada usuário, dependendo da sua responsabilidade, terá um diagrama adequado ao seu contexto de trabalho, mas que se comunica os demais diagramas, mantendo uma comunicação mais fluida e precisa.

Almorsy, Grundy e Rüegg (2014) compartilham também do problema de comunicação entre equipes multidisciplinares e vai um pouco mais além em sua abordagem, propondo o uso de DSVL para alterar os contextos computacionais por meio de diagramas. Contudo, ressalta que as ferramentas para criação de DSVL atuais não capturam informações como permissões de usuário, preferências, localização e dispositivos alvos durante a edição do diagrama e alerta para a necessidade de manipulá-las utilizando outras abordagens.

No estudo para simulação de redes, Kourzanov (2013) utiliza DSL para inserção de dados e para resolver lacunas durante o processo, neste último caso fazendo uso de EDSL (*Embedded DSL*). São desenvolvidas DSL mais simples e integradas em alguns pontos específicos, que tratam o problema de forma individual e não externando impedimentos. Ainda com o uso de EDSL, é defendida a criação de identificadores de padrões, para identificar se as especificações foram construídas corretamente.

Os nove trabalhos relacionados nesta seção visam responder diretamente à questão de pesquisa 1 da revisão bibliográfica, que visava encontrar trabalhos correlatos ao que se pretende desenvolver neste trabalho. Os demais trabalhos não citados atendem às questões referentes à construção e ao uso de DSL, que também eram foco da revisão. A Tabela 4

(pág. 28) contém todos os 21 trabalhos selecionados, nos quais se incluem os descritos nesta seção.

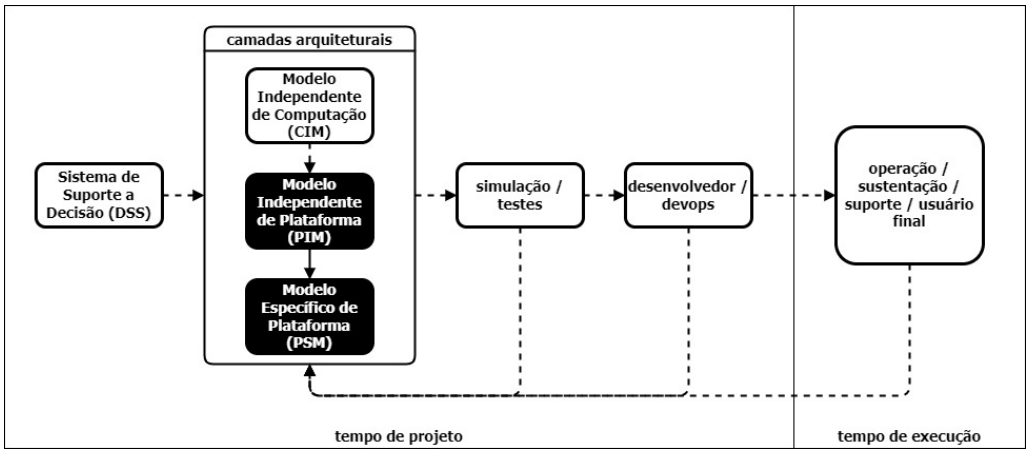
# 4 BLENDMESH

## 4.1 Elementos e escopo do sistema de provisionamento

O processo para provisionamento de sistemas na nuvem tem suas macroatividades apresentadas na Tabela 6. Neste capítulo, é apresentada a estrutura geral do sistema desenvolvido, o qual foi denominado Projeto Blendmesh. É um projeto porque vai além do escopo deste trabalho de conclusão de curso, que implementa parte do processo.

A Figura 13 apresenta o ciclo de funcionamento do sistema de software do projeto. O processo de provisionamento pode ser aproximadamente dividido em tempo de projeto (*project-time*) e tempo de execução (*run-time*). Durante o tempo de projeto são realizados os levantamentos de requisitos da aplicação e de recursos financeiros necessários, são desenvolvidos e atualizados os projetos das arquiteturas, são realizadas simulações, testes de aplicativos em ambientes de desenvolvimento e homologação e os ajustes das esteiras. Durante o tempo de execução o ambiente está em estado produtivo, onde equipes de sustentação, operação e suporte estão garantindo o seu funcionamento para o usuário final.

Figura 13 – Contexto que o projeto Blendmesh e em destaque o que este trabalho atende




Fonte: Autor (2022)

O *Decision Support System* (DSS) é utilizado para determinar qual provedor ou tecnologia utilizar para apoiar a solução, comparando custos, riscos e as características funcionais de cada opção. Esta é uma etapa de difícil realização, a obtenção das informações para o levantamento de requisitos pode variar de caso para caso. Por questões de conformidade, segurança ou legislação, muitas organizações não podem divulgar

como seus fluxos de trabalhos funcionam e a mensuração das informações necessita ser realizada por coletores e sintetizado o possível fluxo que o cliente possui. Em outros casos é assinado um acordo de não-divulgação, tratado em reuniões fechadas e todo esse processo ocorre em um ambiente criptografado. A compreensão de todos os fatores para atender o objetivo do cliente é essencial, desta maneira é possível dimensionar corretamente cada recurso e elaborar um fluxo de trabalho ideal. Ao contrário do cenário ideal, na maioria dos casos, o cliente idealiza o planejamento do que tem que ser provisionado e não considera a falta de informação um impedimento. O Blendmesh pode automatizar esta etapa utilizando uma base de modelos de fluxos de trabalho já testados e os adaptando com o uso das informações de coletores e da resposta de um formulário para definir a melhor abordagem, elencando as necessidades de melhorias e listando os *data lock-in* a serem transpostos.

Com as decisões tomadas e objetivos definidos, começa o processo de modelagem da infraestrutura, por meio das camadas arquiteturais CIM, PIM e PSM. Na camada CIM é projetado o fluxo de trabalho da infraestrutura tomando como base o mundo real, logo após conceitos de tecnologia são acrescentados na camada PIM e por último são especificadas as tecnologias na camada de PSM. O modelo provisionado nas camadas de arquitetura é simulado e testado, gerando novas propostas de mudanças para o modelo. Em uma próxima fase, os desenvolvedores submetem suas criações à infraestrutura e são coletadas métricas que auxiliarão a refinar o modelo. Quando está em tempo de execução, as métricas do ambiente produtivo, relatos de falhas e sugestões de melhorias são submetidas também às camadas de arquitetura para refinar o modelo e continuar o ciclo. A aplicação das camadas de arquitetura simplifica o processo de desenvolvimento do projeto a ser implementado, evidenciando problemas no fluxo de trabalho antes da infraestrutura ser provisionada ou uma tecnologia ser empregada. Erros de projeto são comuns de acontecer por falta de informações, equipes atuando em paralelo no mesmo projeto com abordagens de trabalho diferentes ou quando se faz necessário acelerar etapas dos processos ágeis, ocasionando atrasos nas entregas e diminuição das margens de lucro.

Este trabalho focou na construção dos artefatos relacionados a duas camadas da arquitetura: o PIM, responsável pelo projeto em mais alto nível e o PSM onde é especificada e provisionada a infraestrutura. O produto final deste trabalho não é uma plataforma completa, mas uma base sólida para receber as outras partes de maneira sustentável. A redução do escopo do trabalho foi necessária por questões de tempo, mas é a peça-chave para o desenvolvimento dos artefatos restantes do projeto. Para validar

as etapas que possuem dependências de estruturas que não foram implementadas, foram utilizados modelos prontos com todas as informações necessárias.

Tomando como base o processo de provisionamento convencional, definido na Tabela 6, duas tarefas foram removidas: a codificação e o provisionamento da infraestrutura, que são agora realizadas pelo sistema. Na Tabela 7, as cinco etapas resultantes mostram a diminuição dos esforços das equipes de sustentação e projetos, os deixando encarregados somente de acompanhar o processo de validação e entrega do que foi provisionado e não mais por codificar manualmente a infraestrutura para o Terraform.

Outro fator que foi alterado com a retirada dessas duas tarefas é a documentação, que deveria ser entregue após a conclusão das mesmas. Na Tabela 8 é visualizado que a documentação de uso da infraestrutura foi redirecionada para a tarefa de projeto da infraestrutura e os códigos são entregues diretamente pela plataforma, sem envolvimento de um membro de equipe. Mesmo atuando unicamente nas camadas PIM e PSM durante a execução deste trabalho, as duas tarefas que demandam mais tempo e envolvimento de pessoas foram removidas (ver Tabela 8), resultando em uma redução significativa no tempo de entrega.

Tabela 7 – Nova distribuição das tarefas para o provisionamento

| Item | Tarefa | Responsável | Sucessor |
|---|---|---|---|
| 1 | Levantamento de requisitos | Equipe de projetos | item 2 |
| 2 | Projeto da infraestrutura | Arquiteto ou engenheiro | item 3 |
| 3 | Plano de implementação / migração / virada | Equipe de projetos e sustentação | item 4 |
| 4 | Implementação da infraestrutura | Equipe de sustentação | item 5 |
| 5 | Plano de continuidade | Equipe de sustentação | - |

Fonte: Autor (2022)

## 4.2 Projeto da DSVL

O provisionamento de recursos atende a certos requisitos em um domínio de conhecimento bem específico, garantindo que a infraestrutura suporte o fluxo de trabalho necessário. Esse domínio leva em consideração as diferentes tecnologias e suas aplicações, o modelo de negócio que a organização ou projeto utiliza e como os *stakeholders*, internos e externos, agem nesse ambiente levando em consideração seus contextos de atuação. Neste último aspecto, a comunicação entre os envolvidos afeta diretamente a qualidade da infraestrutura, podendo ocorrer dimensionamentos

Tabela 8 – Documentação resultante após a conclusão das tarefas

| **Tarefa** | **atual** | **proposto** |
|---|---|---|
| Levantamento de requisitos | documento de descoberta | documento de descoberta |
| Projetar infraestrutura | projeto da infraestrutura, estimativas de custos (durante e após provisionamento) | projeto da infraestrutura, estimativas de custos (durante e após provisionamento), documentação de uso da infraestrutura |
| Codificar infraestrutura | códigos da infraestrutura em formato terraform ou json | - |
| Provisionamento da infraestrutura | documentação de uso da infraestrutura | - |
| Plano de implementação / migração / virada | plano de execução de migração ou recuperação de desastre, GMUD | plano de execução de migração ou recuperação de desastre, GMUD |
| Implementação da infraestrutura | comprovante de GMUD executada | comprovante de GMUD executada |
| Plano de continuidade | Plano de Continuidade do Negócio | Plano de Continuidade do Negócio |

Fonte: Autor (2022)

equivocados e falhas no campo da segurança, ocasionando retrabalho e um aumento significativo nos custos. O problema de comunicação ocorre normalmente em projetos realizados com prazos reduzidos e em equipes multidisciplinares com habilidades e experiências em níveis diferentes, contendo de profissionais da área de negócios a cientistas de dados, por exemplo.

Nesse cenário com uma ampla gama de variáveis, uma linguagem comum que una o modelo de negócios com as diferentes perspectivas dos *stakeholders* e as tecnologias necessárias para o seu funcionamento é necessária. Para auxiliar nas diferentes expertises dos envolvidos, é abordado neste trabalho o uso de visões do modelo de negócio. Cada ponto de visão tem o seu próprio contexto, notação e regras, que devem ser contempladas e ao mesmo tempo interagir com as outras visões. Alguns exemplos de visões do modelo de negócios:

**Analista de negócios:** normalmente são representados por um *Product Onwer* (PO), *Product Manager* (PM), *Service Level Manager* (SLM) ou um membro da área de FinOps (gestão financeira do consumo em nuvem). Sua visão deve atender itens

como o progresso das atividades, mapeamento dos processos e custos operacionais e de implantação.

**Projetista:** utiliza ferramentas que auxiliam na descrição dos requisitos funcionais e não funcionais, *workloads*, custos operacionais, recursos utilizados, necessidade de melhorias e listagem dos *data lock-in*.

**Engenheiro ou arquiteto:** sua visão contem itens como topologia de rede, diagramas de recursos ou descrição dos recursos.

**Desenvolvedor:** é exibida a descrição do comportamento das aplicações e suas relações entre si.

**DevOps:** visualiza itens de provisionamento de aplicações e CI/CD.

**Operação:** utiliza artefatos que contenham as relações entre os componentes e seu dimensionamento.

**Simulação:** deve conter itens que auxiliem no ajuste do modelo que será utilizado em processos de aprendizado de máquina.

**Teste:** descritores com os tipos de testes que serão realizados e como eles afetam o ambiente.

**Suporte ou sustentação:** necessita visualizar itens de monitoramento, documentações, visualização da infraestrutura e recursos que auxiliem na resolução de problemas.

Cada usuário, baseado em sua responsabilidade (visão do modelo de negócio), trabalha com um tipo de diagrama diferente, mas utilizando a mesma linguagem. Cada diagrama inclui uma coleção de notações para modelar diferentes objetos, suas relações e regras. Eles são divididos em dois tipos de diagramas: diagramas de alto nível e de baixo nível. Os diagramas de alto nível são mais abstratos, fazendo referência ao comportamento do fluxo de trabalho e suas relações com o mundo real, podendo ser descritos em mapas mentais ou diagramas de processos. Os diagramas de baixo nível trazem representações e especificações do uso das tecnologias, técnicas e dados utilizados no provisionamento, podendo ser descritos em diagramas de provisionamento e topologias.

Para poder atender os aspectos de comunicação e visões do modelo de negócios, simultaneamente, simplificando o uso das tecnologias envolvidas, foi empregada a utilização de um conjunto de DSL para formar uma DSVL mais completa. O desenvolvimento desta linguagem foi apoiado na metodologia MDA que possui como base a modelagem dos processos no desenvolvimento de *software*.

O emprego da construção da DSVL utilizando o MDA se dá pela natureza do projeto: o processo de provisionamento de recursos acontece com uma constante de entregas sucessivas e tem que estar apto a mudanças pontuais, repentinas e, frequentemente, de grande impacto. As camadas arquiteturais sendo representadas pelos artefatos que, além de auxiliar na abstração dos conceitos para todos os membros da equipe delimitam as etapas do projeto e refletem o estado atual de desenvolvimento. Nesse ponto, a aplicação de métodos ágeis e seu acompanhamento dentro da plataforma é um ponto estabelecido no escopo macro do projeto, mas que está além do escopo deste trabalho.

Outro fator que a aplicação de *model-driven architecture* beneficia o estudo é a utilização dos transformadores entre cada etapa, sendo responsáveis de fato pela tradução da comunicação entre os diferentes contextos de interação e, conforme a linha do tempo do projeto vai sendo percorrida, a abstração vai dando lugar a uma infraestrutura cada vez mais confiável. Ao necessitar de novas visões do modelo de negócios, não é necessária uma intervenção na estrutura da arquitetura da plataforma, basta desenvolver o diagrama apropriado e seus transformadores, acrescentando aos seus respectivos repositórios e tornando a plataforma bastante flexível e modular.

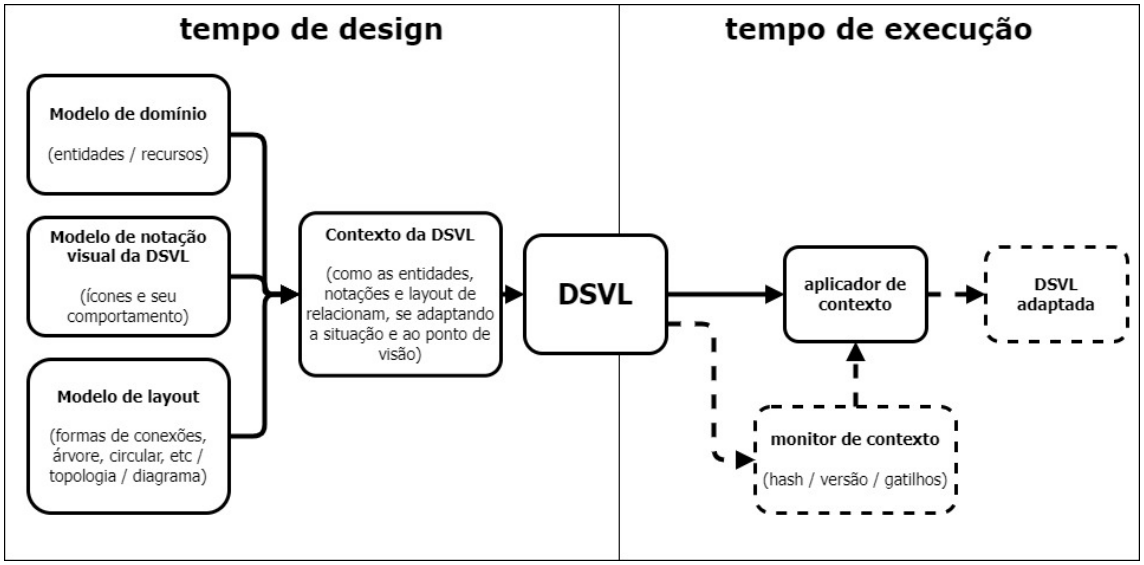
Todo este processo, desde a concepção dos fluxos de trabalho até seu provisionamento e execução, deve ter a garantia de que toda informação é coesa e que qualquer mudança seja refletida para todos os envolvidos. As visões de MDA atuam neste quesito, onde a visão estática mantém o provisionamento dos recursos e suas associações que seguem modelos pré-definidos na plataforma. Quando o projeto assume uma maturidade maior e estas associações já precisam enfrentar tomadas de decisões de como se comportar em determinados eventos a visão dinâmica é empregada, funcionando como uma máquina de estados. O emprego destas duas estruturas é um dos pilares da plataforma e sendo transparente para o usuário final, somente sendo percebida na consistência das atualizações dos diagramas e códigos.

Para o desenvolvimento dos transformadores e a manipulação dos diagramas foi empregado o uso de algumas DSL que atendem contextos diferentes, sendo para a transição entre as camadas arquiteturais ou na manipulação das estruturas das visões estáticas e dinâmicas do MDA. No escopo geral do projeto, todos os metamodelos das DSL derivam de metametamodelos que garantem a consistência do funcionamento da plataforma e uma padronização no desenvolvimento de novos componentes, por questão do tempo de execução deste estudo, são criados os metamodelos das DSL diretamente mas

com um foco futuro na sua subordinação a um metametamodelo. De acordo com Sebesta (2018), por poder representar a si mesmo, um metametamodelo tem um grande poder de abstração e é utilizado para descrever outras linguagens, sendo uma metalinguagem.

Na manipulação dos diagramas foi utilizada uma DSVL que possui dois tempos de trabalho, no tempo de *design* toda a estrutura e modelos são estabelecidos e devem interagir entre si garantindo que as modificações ocorridas sejam entregues aos transformadores corretos, enquanto no tempo de execução ele aplica os diagramas de acordo com a visão solicitada através do aplicador de contexto. O tempo de *design* é dedicado ao desenvolvimentos dos modelos, sendo independente do tempo de projeto, o primeiro ocorre durante o desenvolvimento da linguagem enquanto o segundo ocorre durante o uso da plataforma no desenvolvimento das arquiteturas que serão provisionadas. Estes dois tempos podem ocorrer ou não em paralelo, pode-se desenvolver novas atualizações para a linguagem durante o tempo de design, sem afetar o uso da plataforma durante o tempo de projeto, mas como compartilham o mesmo tempo de execução, pode-se fazer uso do aplicador de contexto presente na interface da plataforma para aplicar as atualizações desenvolvidas sem a necessidade de parar a plataforma como um todo e adaptando a DSVL utilizada na interface.

Figura 14 – Design dos contextos das DSVL




Fonte: Autor (2022)

No tempo de *design*, quatro estruturas são responsáveis pelo resultado final, o modelo de domínio, de notação visual, de layout e de contexto (ver figura 14). No modelo de domínio da DSVL são definidos os comportamentos das entidades e recursos, que são

compreendidos pelos itens de *hardware* e *software* nesta etapa do projeto. O modelo de notação visual é responsável pela representação gráfica das entidades e recursos, além da definição de um ícone também define como as mensagens e informações são apresentadas no diagrama, assim como explicitar os pontos de associações entre as entidades. No modelo de layout da DSVL são determinadas como as conexões podem ocorrer, as possibilidades de como as entidades ou recursos são agrupados (árvore, circular, etc) e o layout padrão do diagrama. Para consolidar os três modelos anteriores é utilizado o modelo de contexto da DSVL, o qual relaciona os elementos já definidos os adaptando para o contexto ou visão do modelo de negócio apropriado.

No tempo de execução, duas estruturas são possíveis, o aplicador de contexto e o monitor de contexto. O aplicador de contexto recebe as solicitações da plataforma conforme o usuário e devolve o seu diagrama correspondente, por exemplo, o engenheiro pode solicitar a topologia de rede da infraestrutura e os recursos de rede são obtidos pelo modelo de domínio, a representação gráfica pela notação visual e a estrutura hierárquica pelo modelo de layout e retorna o diagrama de topologia acrescido de informações para este contexto, se for necessário. O monitor de contexto fica monitorando mudanças na estrutura da DSVL ou atualizações, como em qualquer etapa pode solicitar uma alteração na estrutura do projeto ou até mesmo da plataforma, garantindo que não ocorram perdas de informações e solicitar a renderização atualizada dos diagramas ativos. Como o monitor de contexto está diretamente ligado às outras camadas da arquitetura, resultados de simulações, monitoramento e testes, e também deve ser modelado com base em um metametamodelo, ele não foi implementado neste trabalho.

## 4.3 Projeto da interface gráfica

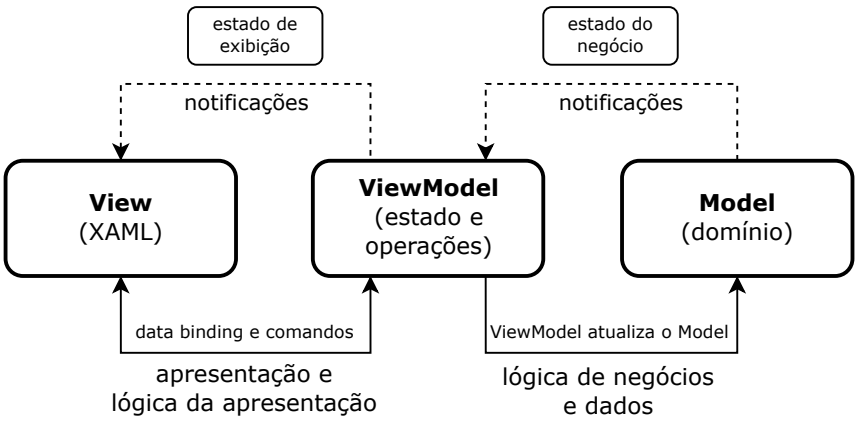
A interface gráfica é o meio pelo qual os vários perfis de usuários interagem com a plataforma, desenvolvendo e provisionando a infraestrutura em um provedor em nuvem. Pela característica destas equipes serem multidisciplinares e trabalharem em áreas físicas distantes ou até mesmo no formato remoto, e também garantir uma rápida atualização em todos os pontos de acesso, a forma de entrega da plataforma avaliada mais viável foi provisioná-la em um ambiente em nuvem acessível pela Internet.

A modelagem da arquitetura (*frontend*) foi realizada utilizando o *framework JavaScript VueJs* para criação de interfaces com suporte a *Typescript*. A escolha pelo uso do *Vue* ocorreu pelo seu comportamento padrão ser baseado em componentes, podendo

isolar os elementos individualmente e monitorar o seu estado, por poder renderizar áreas específicas da página de maneira mais simplificada e pela sua abordagem de linguagem declarativa. O *Typescript* foi utilizado para garantir uma tipagem estática ao *Javascript*, pois dependendo da tecnologia empregada no projeto da infraestrutura no momento do provisionamento, o uso de uma tipagem dinâmica pode ser um impeditivo, assim é garantida a compatibilidade das entradas de ponta a ponta em qualquer contexto.

O padrão de projeto arquitetural escolhido para o *frontend* foi o MVVM (Model View ViewModel), que foi criado pela *Microsoft* e possui um funcionamento semelhante a outros padrões como *Model View Presenter* (MVP) e o *Model View Controller* (MVC). Foi elegido por dividir o software em três partes interconectadas (ver Figura 15), possuindo seu foco em projetos com uso intenso de interfaces gráficas a partir do desenvolvimento das regras do domínio e, dessa forma, dissipando a lógica do domínio entre as camadas de apresentação, evitando a duplicidade de código e forçando a reutilização. Por ter uma camada de apresentação com fraco acoplamento, simplifica a manutenção, extensões da interface de usuário e a realização de testes unitários em todas as camadas sem a utilização de uma interface.

Figura 15 – Funcionamento do MVVM




Fonte: Adaptado de (OSMANI, 2012)

Sua camada *model* tem como característica um conjunto de classes que descrevem as regras ou lógica do domínio, ditando como os dados serão manipulados. Ela também pode ser utilizada como um repositório, criando uma subcamada para atender aos mais diversos tipos de acessos de dados, como API, bancos de dados, arquivos, entre outros. A camada *View* representa os componentes da interface do usuário, possuindo o mínimo possível de regras do domínio e utilizando um Extensible Application Markup Language (XAML) declarativo para construir a interface com as informações vindas da camada

*ViewModel*.

A camada *ViewModel* é responsável por manter o estado e a camada *View* atualizada por meio de comandos e *data binding*, realizando a associação entre a camada *model*, que possui o domínio e os dados, com a camada *View*, que define quais dados são relevantes para envio à interface do usuário. Essa camada pode, ainda, ser utilizada como ponto de integração com outros serviços, como banco de dados, e em projetos pequenos pode absorver a camada *model*. Para manter a sincronização dos dados com a camada *View*, utiliza uma interface declarativa no *data binding* e utiliza comandos para manter as dependências mais difíceis entre diferentes tipos de códigos. A utilização de comandos permite que métodos, classes, controles e componentes sejam alterados mais facilmente, criando um desacoplamento com separação de conceitos.

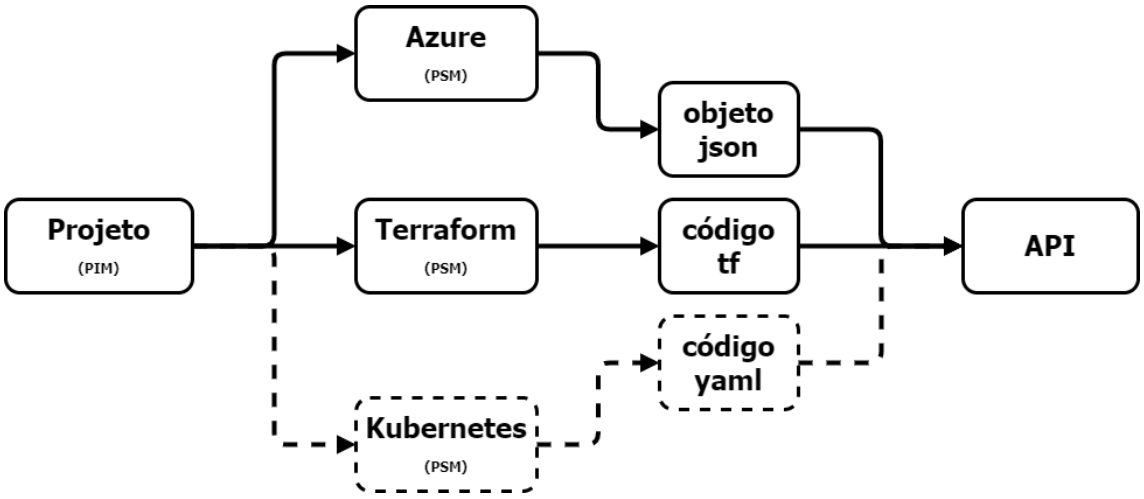
Na análise e aplicação das regras de criação da arquitetura (*backend*), para o desenvolvimento e suporte para as DSL e a DSVL foi utilizado o *runtime JavaScript nodeJS*, pela sua característica assíncrona orientada a eventos, escalabilidade e por manter uma unicidade com a linguagem do *frontend*, diminuindo os custos de manutenção, suporte e contratação de mão de obra futura. Foi cogitado o uso de C++ ou *Python* no *backend* através da linguagem textual *WebAssembly*, mas esta tecnologia, apesar de apresentar bons resultados, está em estágios iniciais de desenvolvimento, o que poderia acarretar em refatorações constantes do código ou instabilidades devido à incompatibilidade com algum navegador ou recurso.

Para o desenvolvimento da interface gráfica e do estudo como um todo foi estabelecido um conjunto mínimo de recursos e tecnologias suportadas, mas que garantiram a demonstração do funcionamento da plataforma. O provedor de nuvem que recebeu suporte foi o Azure da *Microsoft* pela sua maior transparência na descrição de como os recursos devem ser entregues a sua API. Para validação do provisionamento e manter compatibilidade com o ferramental utilizado atualmente no mercado é possível exportar o provisionamento no formato do Terraform. Quanto aos recursos foram elegidos alguns de IaaS e PaaS, como recursos de rede, máquinas virtuais, bancos de dados e similares, esta limitação de recursos iniciais se deve ao prazo de conclusão do trabalho.

Na figura 16 são apresentados os PSM presentes no estudo, onde depois de realizado um projeto independente de tecnologia, pode ser utilizado tanto o PSM de Azure para especificar os recursos que serão provisionados e resultando na criação de um objeto JSON com a infraestrutura, como o PSM de Terraform, que para este estudo, tem como resultado a codificação dos recursos especificados em Azure para um arquivo

com a extensão do Terraform. Também é exibido a possibilidade de PSM futuros, sendo exemplificado pelo orquestrador de *conteiners Kubernetes*, que pode gerar manifestos yamls para sua implantação. Em ambos os casos, os resultados dos PSM são submetidos a API do provedor de nuvem para provisionamento.

Figura 16 – PSM para este estudo




Fonte: Autor (2022)

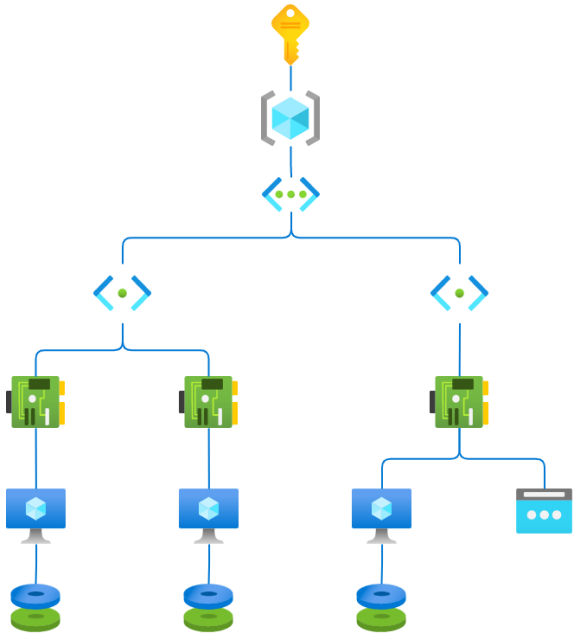
Para o desenvolvimento dos projetos, o usuário encontra na interface os mesmos ícones dos recursos presentes nos próprios serviços de nuvem e podendo alterá-los conforme a necessidade da representação. Na figura 17 são apresentados exemplos de alguns ícones de recursos utilizados na plataforma Azure da *Microsoft*.

Figura 17 – Alguns dos ícones de recursos distribuídos pela Azure


Fonte: Microsoft Docs (2020)

Ao acessar o portal da Azure e solicitar a geração da topologia de recursos é retornado um diagrama como o mostrado na figura 18, onde cada recurso só pode ser conectado a um certo escopo de recursos. Usualmente para representar esta topologia em projetos é utilizada uma visão a partir da rede e não dos recursos (ver figura 19), assim ocultando vários componentes que serão atribuídos no momento da codificação.

Figura 18 – Modelo de topologia gerada por ferramentas da Azure



Fonte: Autor (2022)

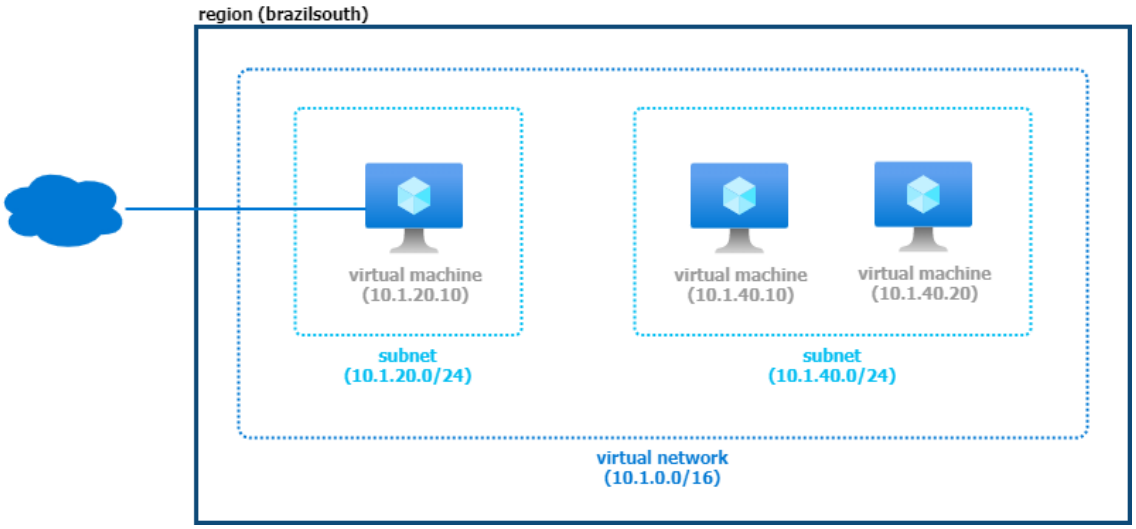
A plataforma mescla estes dois modelos de topologia gerando um diagrama dinâmico (ver figura 20), contemplando todas as informações necessárias, autocompletando informações e apontando os erros na declaração dos recursos.

Após passar por todas as etapas do processo de provisionamento dentro da plataforma, o PSM responsável pelo provisionamento na Azure, fornece um arquivo contendo a representação de um grafo com as dependências entre os recursos e sua ordem de implantação. Para apurar a qualidade do provisionamento e validar seu uso foi utilizada a ferramenta de infraestrutura Terraform, usada para compilar o grafo, onde a verificação foi feita pela análise da equivalência entre o arquivo de saída da plataforma e o arquivo gerado pelo Terraform. No provedor de nuvem o grafo gerado foi submetido para a ARM API onde foi validado pelo provedor e provisionado. As funcionalidades foram desenvolvidas de forma incremental, facilitando os testes unitários e de regressão, garantindo que o incremento de funcionalidades não altere o que foi anteriormente implementado.

## 4.4 Gerenciador de estados

Uma primeira abordagem do projeto previa a utilização de um objeto JSON para armazenar o estado de forma estática, sendo utilizado como o arquivo de trabalho padrão

Figura 19 – Modelo de topologia utilizada em projetos



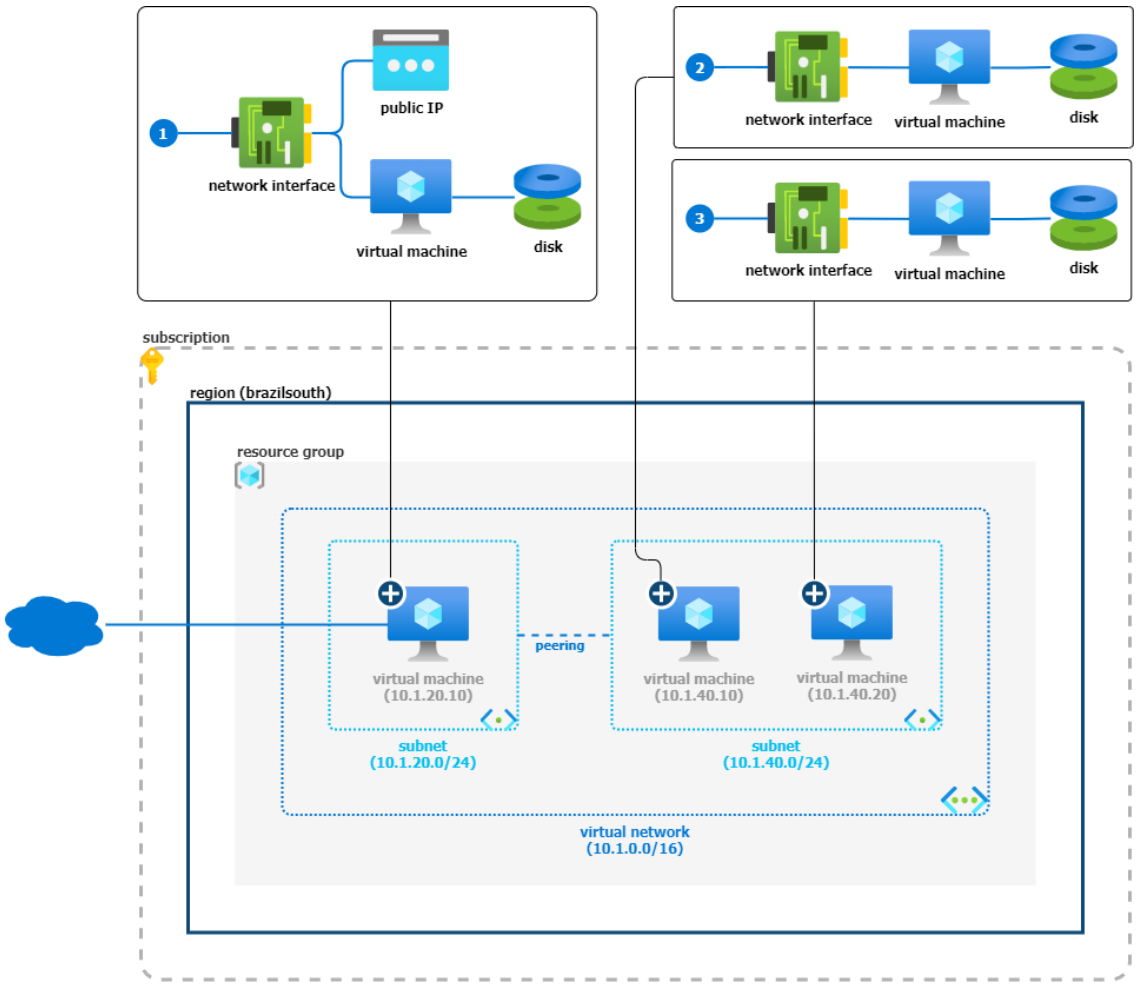

Fonte: Autor (2022)

da plataforma. À medida que a aplicação foi crescendo, sua complexidade em manter o estado atualizado aumentou proporcionalmente, sendo necessário encontrar uma solução para abrandar o acoplamento entre os componentes. A incorporação do gerenciador de estados Pinia (<https://pinia.vuejs.org/>) à plataforma trouxe uma maior flexibilidade na manipulação dos estados do domínio da aplicação e evitando conflitos com o estado da interface gráfica.

Os dados utilizados na aplicação são distribuídos entre três estados, denominados baseStore, projectStore e transformerStore. O baseStore é utilizado para armazenar as informações padrão da aplicação, por exemplo, especificações dos componentes, sistemas operacionais suportados ou regiões disponíveis nos provedores, e deve permanecer imutável até que a plataforma receba novas atualizações. Na Figura 21 pode ser observada a organização deste estado, mantendo as informações das camadas de arquitetura independentes e onde cada uma delas são importadas individualmente de arquivos JSON específicos. Para atribuir as informações do PIM é importado o arquivo pim.json contendo os valores padrões da plataforma e cada PSM possui um arquivo JSON que é atribuído ao estado conforme a necessidade. Esta abordagem foi utilizada visando atualizar o estado mais facilmente, agilizar o processo de regressão caso ocorra algum problema na atualização e futuramente poder receber estas atualizações de um banco não relacional.

Diferente do baseStore que possui um comportamento estático, o projectStore possui um comportamento dinâmico e é o estado principal da plataforma. Ele é primariamente dividido em cinco grupos de informações: *accounts*, *architecturalLayers*, *informations*, *representations* e *schema* (ver Figura 22). O elemento *schema* contém

Figura 20 – Proposta de diagrama dinâmico para o projeto

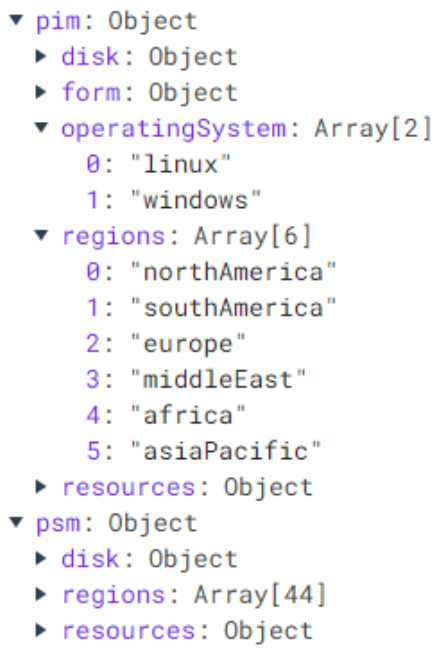



Fonte: Autor (2022)

a versão da construção do estado, determinando como cada elemento estará disposto e como deve ser tratados quando carregados pela plataforma. O grupo *accounts* armazena os acessos aos ambientes de provisionamento em nuvem para realizar as implantações ou consultar o estado atual no provedor, e o grupo *informations* atua monitorando o andamento e responsabilidades contidas no projeto, dentro do escopo geral do que será provisionado. Neste trabalho o grupo *informations* recebe apenas dados básicos, este campo tende a ser expandido para receber os dados de gerenciamento do projeto como um todo, alimentando linhas de tempo, históricos de modificações e quadros de atividades.

Os dois principais grupos deste estado são o *representations* e o *architecturalLayers*. Quando é acrescentado um novo recurso de nuvem em um diagrama, as informações de contexto da DSVL, como por exemplo a posição do elemento, são armazenadas no grupo *representations* no seu respectivo tipo e camada de arquitetura. As informações de domínio específico, como o tipo de sistema operacional,

Figura 21 – Estrutura do estado baseStore



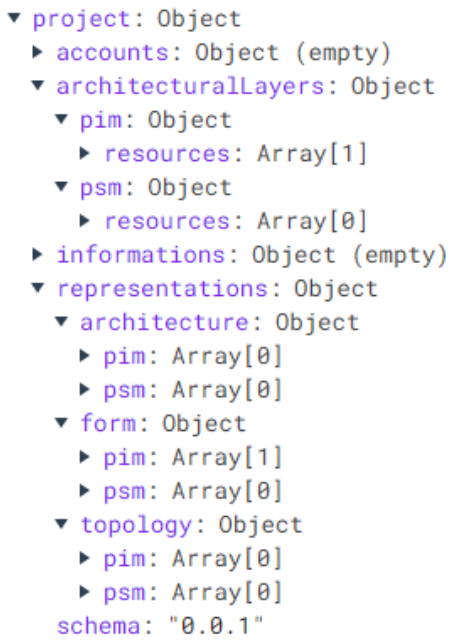

Fonte: Autor (2022)

são armazenadas no grupo *architecturalLayers* em sua respectiva camada de arquitetura. Durante a criação dos elementos, esta divisão é bem definida, onde tudo que estiver contido no objeto data do elemento é considerado dado de domínio específico e o restante é considerado dado de manipulação de representação. Assim como o baseStore, este estado é provisionado através de um arquivo JSON denominado default.json, contendo a estrutura básica esperada pelos componentes da aplicação e seu versionamento sendo identificado pelo elemento *schema*.

O último estado acrescentado, o transformerStore, é considerado volátil por ser utilizado somente quando um transformador do tipo PIMtoPSM (entre camadas arquiteturais) ou PSMtoCode (modelo para código) for acionado. Na raiz do projeto existe uma pasta denominada *Models* contendo os arquivos que servem de modelo para os estados, e além de possuir os arquivos default.json e pim.json, citados anteriormente, possui arquivos para os transformadores implementados e que serão atribuídos conforme a necessidade. Na Figura 23, o transformador pim4azure (entre camadas arquiteturais) foi atribuído e garantindo o seu estado esperado, assim como o transformador azure4code (modelo para código).

Conforme pode ser observado na figura 24, cada recurso passa a ser mapeado e recebe uma identificação única que é utilizada na construção dos diagramas, independente do contexto utilizado. Para tentar garantir que os recursos provisionados não possuam

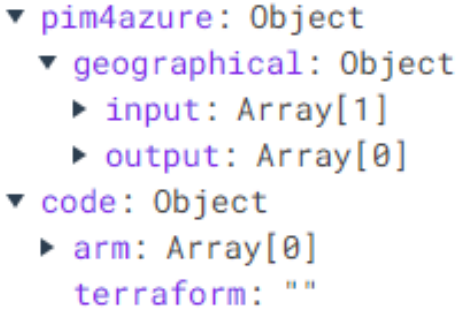
Figura 22 – Estrutura do estado projectStore




Fonte: Autor (2022)

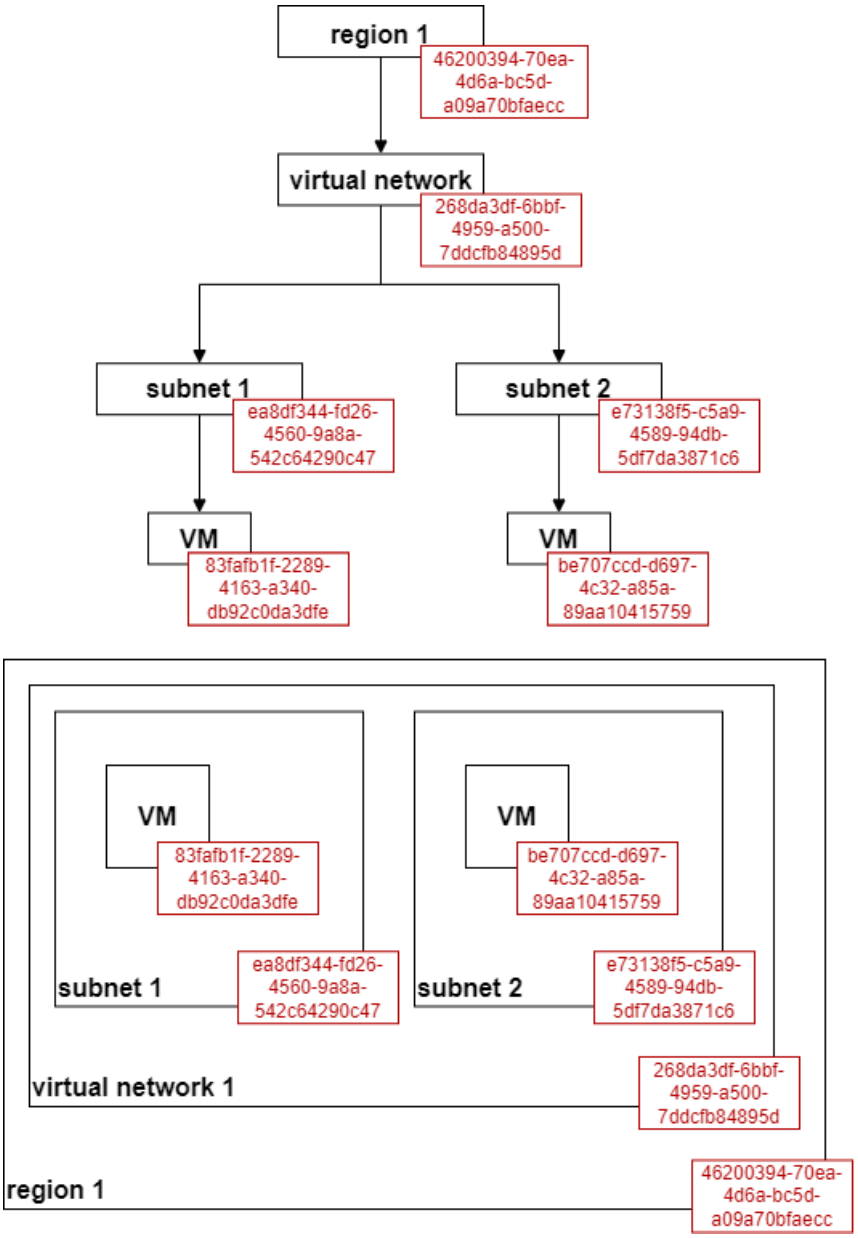
uma mesma identificação, é utilizado um número de 128 bits para identificar informações em sistemas computacionais denominado identificador único universal (UUID). Em versões futuras, o uso do UUID irá garantir que dois ou mais projetos possam ser mesclados e criar ambientes mais complexos.

Figura 23 – Estrutura do estado transformerStore




Fonte: Autor (2022)

Figura 24 – Mapeamento dos recursos entre os diagramas




Fonte: Autor (2022)

## 4.5 Construção da interface gráfica

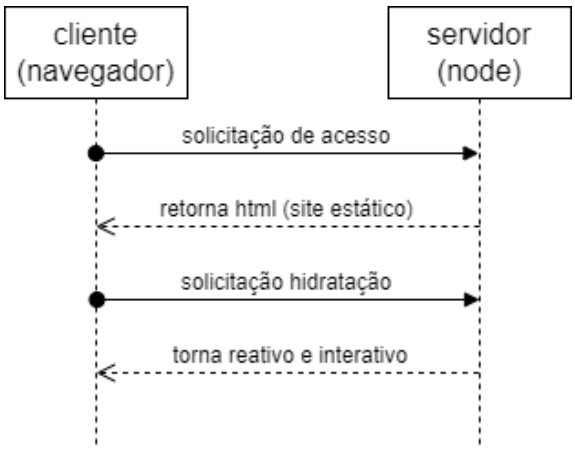
Após o planejamento dos elementos que compõem a interface com o usuário, o seu processo de criação se torna mais intrincado e impedimentos não previstos são esperados. Nos primeiros protótipos da interface, para avaliar a viabilidade de implementação de alguns componentes, alguma lentidão na renderização da interface foi percebida e conforme o acréscimo de funcionalidades, mais instabilidades foram observadas.

O processo de renderização estava sendo realizado no lado do cliente (navegador), assim como a interface e o domínio compartilhavam um mesmo estado, ocasionando em requisições constantes ao servidor para atualizar o estado e reconstruir a interface. A primeira decisão foi separar os estados, possuindo um estado para observar a interface e outro para compartilhar o domínio com outros usuários (projectStore), ambos os estados permanecem no servidor e hidratam o cliente conforme a necessidade utilizando pacotes menores.

A outra decisão foi gerar o HTML no lado do servidor antes dele ser entregue ao cliente no navegador utilizando o processo de *Server-Side Rendering*. Neste processo, o HTML dinâmico é pré-renderizado no servidor e a página é entregue ao cliente, podendo ser considerada isomórfica ou universal, pois o código da aplicação pode ser executado tanto no lado do servidor quanto do lado cliente. Conforme demonstrado na Figura 25, o cliente envia uma solicitação ao servidor, que gera uma página HTML estática e não reativa e reenvia ao cliente. Depois de recebido e exibido pelo cliente, a camada *view* vai sendo hidratada gradualmente e se tornando reativa e interativa, sem a necessidade de novos acessos ao servidor enquanto o estado não for criticamente modificado ou a página atualizada.

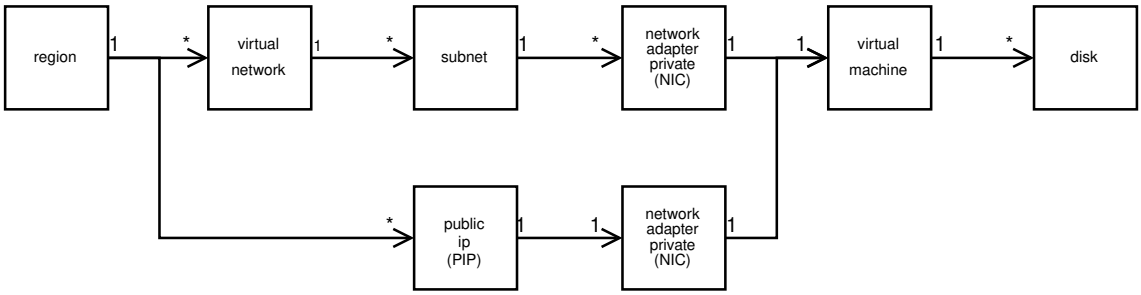
Com as decisões sobre renderização da interface e organização dos estados redefinidos, o congelamento dos recursos que foram utilizados pelo PIM foi realizado e seus relacionamentos estabelecidos (ver Figura 26). Foi definido o recurso de localidade *region* para obter um isolamento geográfico para um conjunto de recursos, os recursos de rede *virtual network*, *subnet*, *public ip* e *network adapter private* para validar acessos privados e públicos a rede, o recurso de serviço *virtual machine* como terminal de execução e o recurso de armazenamento *disk* para validar associações entre recursos que podem estar situados em diferentes locais em uma mesma região geográfica. O recurso de disco, comumente, vem pré estabelecido quando uma *virtual machine* é criada, mas podem ser adicionados outros conforme a necessidade.

Figura 25 – Renderização SSR




Fonte: Autor (2022)

Figura 26 – Relacionamento entre os recursos que são utilizados




Fonte: Autor (2022)

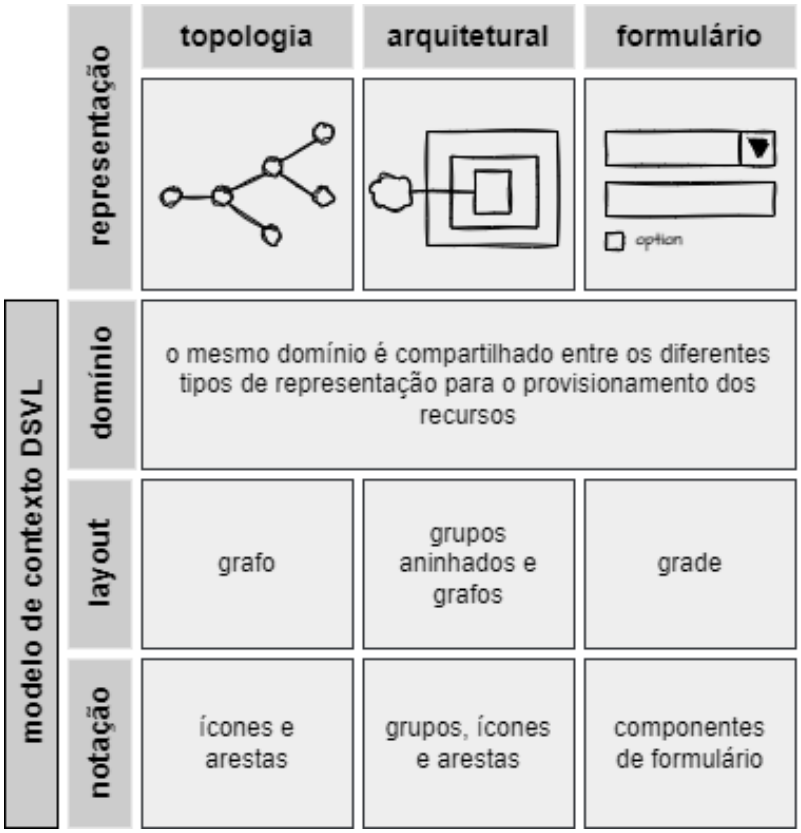
Sendo necessário manipular os recursos do PIM, foram utilizados conceitos do contexto da DSVL (ver Figura 14) estabelecidos neste trabalho e foram desenvolvidos três diagramas de baixo nível, agora sendo denominados tipos de representações de infraestrutura, sendo eles: a representação topológica, o diagrama arquitetural e por formulários. A representação por topologia relaciona os recursos individualmente utilizando arestas, podendo atender profissionais de rede ou quando é necessário realizar a verificação de um problema com maior detalhes. A representação de diagrama arquitetural é a forma mais usual para descrever infraestruturas em nuvem, agrupando os componentes que compartilham recursos e abstraindo outros desnecessários para o entendimento do ambiente. A representação por formulário lista as opções de configuração dos componentes em caixas de textos, caixas de seleção, marcadores, entre outras entradas se assemelhando aos encontrados nos portais dos provedores de nuvem. O contexto da DSVL depende e é composto pelos modelos de notação, layout e domínio.

Para o modelo de notação foram desenvolvidos os componentes de ícone, aresta, grupo e componente de formulário. O componente de ícone possui pontos de conexão de

entrada e saída que são interligados aos outros componentes utilizando arestas. O grupo compreende em um componente que isola dentro da sua representação seus descendentes e herdam alguns de seus atributos, enquanto os componentes de formulário são *inputs* de *forms* HTML exibidos conforme o tipo do dado requisitado.

Os modelos de layout possuem três formas de orquestrar a interação entre os componentes do modelo de notação, que são o grafo, grupos aninhados e grade. O grafo trata os componentes de notação como nós e os conecta utilizando arestas, podendo ser direcionado ou não, dependendo do domínio. O grupo aninhado, atribui a capacidade de um componente de notação grupo além de ter descendentes ícones, também possuir descendentes aninhados do tipo grupo até satisfazer alguma regra do domínio. Os dois tipos de layout anteriores podem coexistir em um mesmo tipo de representação, mas o layout do tipo grade somente é utilizado para componentes de formulário, os distribuindo em linhas e colunas. A relação entre os tipos de representação e os modelos de contextos pode ser observado na Figura 27.

Figura 27 – Tipos de representações distribuídos nos modelos de contexto da DSVL

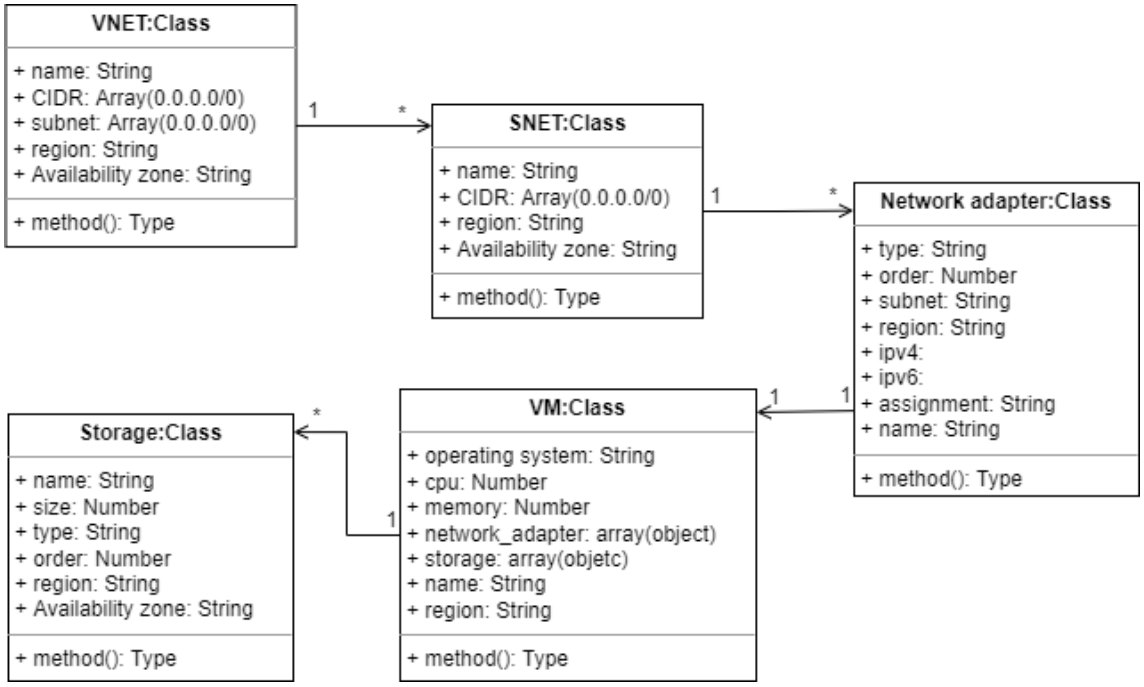



Fonte: Autor (2022)

O relacionamento entre os modelos durante o uso da ferramenta são declarados no estado projectStore, em uma área específica para o seu tipo de representação contendo

as relações entre os recursos, assim como seus atributos e satisfazendo o uso de um estado dinâmico previsto pelo MDA. As informações do domínio de cada recurso são armazenadas em outra área do estado projectStore, reservada para as camadas arquiteturais, no formato de um objeto nomeado de *data*, contendo dados relevantes para o provisionamento e que hidratam atributos dos modelos de layout e notação. Na Figura 28 podemos observar um fragmento do diagrama de classe que compõem a DSL do modelo de domínio, onde o conjunto dos atributos de cada classe formam o objeto *data* e cada atributo representa um elemento chave-valor. Em alguns casos específicos o elemento pode ser um *array* ou objeto, identificando uma relação com outro recurso. Um exemplo de hidratação no modelo de notação utilizando um atributo do domínio é o *name*, para o domínio ele representa o nome que será atribuído quando for provisionado, mas será repassado para o atributo *label* do componente de notação. Outros atributos de notação também são afetados (ver Figura 29), como os manipuladores utilizados para conexão *input* e *output*. Por exemplo, quando é representado um recurso SNET existe somente uma VNET que pode ser conectada em sua entrada e várias placas de redes podem ser conectadas em sua saída, esta tomada de decisão é realizada atendendo a regras de mapeamento entre os componentes observando seus CIDR e região por exemplo.

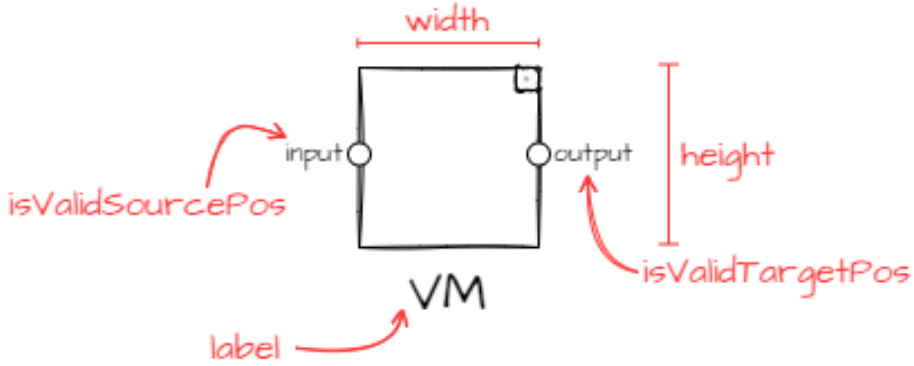
Figura 28 – Parte do diagrama de classes do PIM




Fonte: Autor (2022)

A disposição dos recursos durante a construção da arquitetura é um fator que foi elaborado para orientar usuários menos experientes a construir arquiteturas de mais

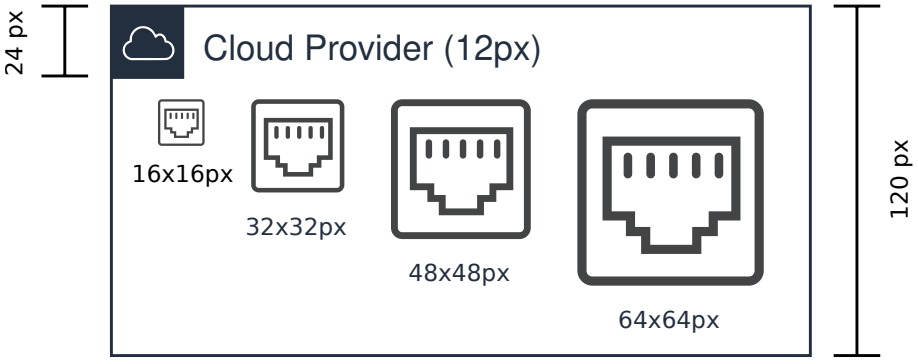
Figura 29 – Relacionamento das variáveis com o renderização dos ícones




Fonte: Autor (2022)

fácil leitura, sugerindo o uso das proporções mais aceitas pelo mercado e que possa ser utilizada em documentações sem muitas adaptações. Os ícones podem ser utilizados em quatro tamanhos 16x16px, 32x32px, 48x48px e 64x64px, mantendo a proporção e um espaço entre eles de 16px (ver Figura 30), podendo ser descendente de um componente grupo que deve possuir uma altura mínima de 120px, sua label possuindo fonte com tamanho de 12px alinhada a esquerda e seu ícone de rótulo de 24x24px. Estes valores podem ser modificados nos parâmetros de configuração visual de cada recurso, mas não é algo que será estimulado pela interface. Para o tipo de representação diagrama arquitetural é bastante utilizado o layout de grupo aninhado e a criação dos seus descendentes também é controlado. Na Figura 31 foi utilizado um ícone de 32x32px como base, a largura do seu ascendente é de 80px e ao solicitar a criação de um novo ícone neste grupo sua largura passara para 160px e aumentará mais 80px para cada novo ícone adicionado. O ascendente deste grupo possui a largura de 128px, aumentando 80px em cada novo ícone adicionado em seu filho e 128px a cada novo grupo adicionado a si.

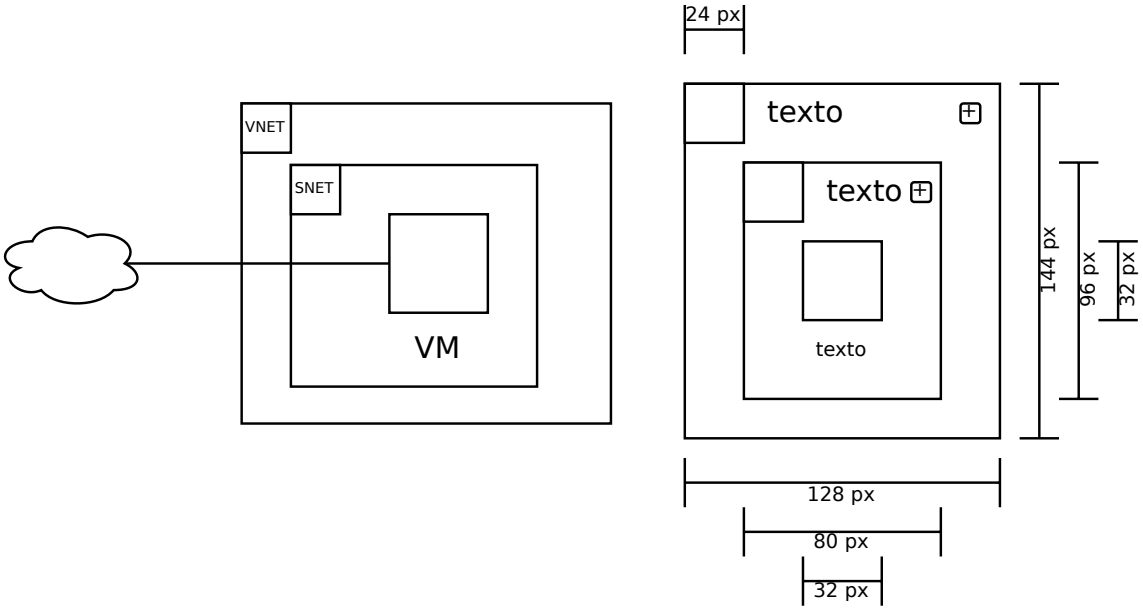
Figura 30 – Escala dos ícones utilizados




Fonte: Autor (2022)

Para projetar a arquitetura a interface gráfica possui além da área de edição, barras

Figura 31 – Proporções utilizadas no diagrama arquitetural



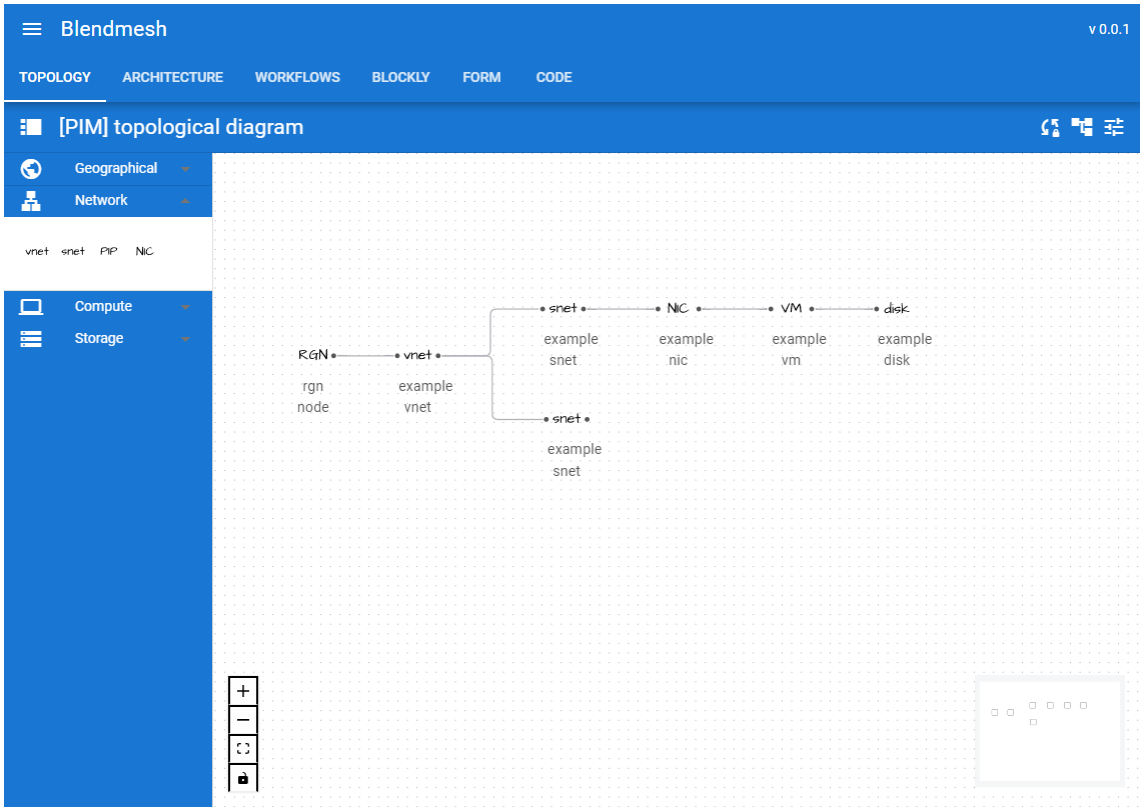

Fonte: Autor (2022)

horizontais de contexto e colunas responsáveis pelos recursos (ver Figura 32). A barra de título está localizada na parte superior da interface, sendo responsável por conter o nome da aplicação, sua versão, abrigar as notificações, além de recursos para importar, exportar, modelos prontos e configurações gerais da plataforma no menu. A barra de tipos de representações está localizada abaixo da barra de títulos e permite alterar entre os tipos de representações, a barra de configurações do tipo de representação ativa fica abaixo da anterior e além da possibilidade de ocultar as colunas da área de edição, permite visualizar o nome do projeto e outras informações relacionadas a esta representação. Ao lado esquerdo encontra-se a coluna de recursos, onde os mesmos são distribuídos em categorias e podem ser facilmente arrastados para a área de edição para compor a arquitetura. Ao lado direito está localizada a coluna de edição de recursos, que altera o seu conteúdo conforme cada recurso é selecionado para exibir seus parâmetros de configuração. Ao centro está a área de edição, possuindo um minimapa no lado inferior direito para localizar os recursos mais facilmente quando está com um zoom elevado, no lado inferior esquerdo possui botões para aumentar e diminuir o zoom, centralizar a visualização do projeto e bloquear a edição e possui uma grade para facilitar o alinhamento dos recursos.

Cada recurso adicionado na área de edição pode ser movimentado para qualquer área do projeto, desde que seja permitido pelas regras do domínio, trazendo informações de contexto do recurso em um *tooltip* ao deixar o mouse sobre o mesmo e a abertura de um menu de contexto ao clicar com o botão direito sobre algum recurso. Para realizar a

configuração das propriedades do recurso e alterar as informações dedicadas ao domínio específico, é necessário clicar com o botão esquerdo do mouse sobre ele (ver Figura 33) e suas informações podem ser alteradas utilizando cinco tipos diferentes de entradas de dados, dependendo do seu contexto.

Figura 32 – Interface gráfica da plataforma Blendmesh

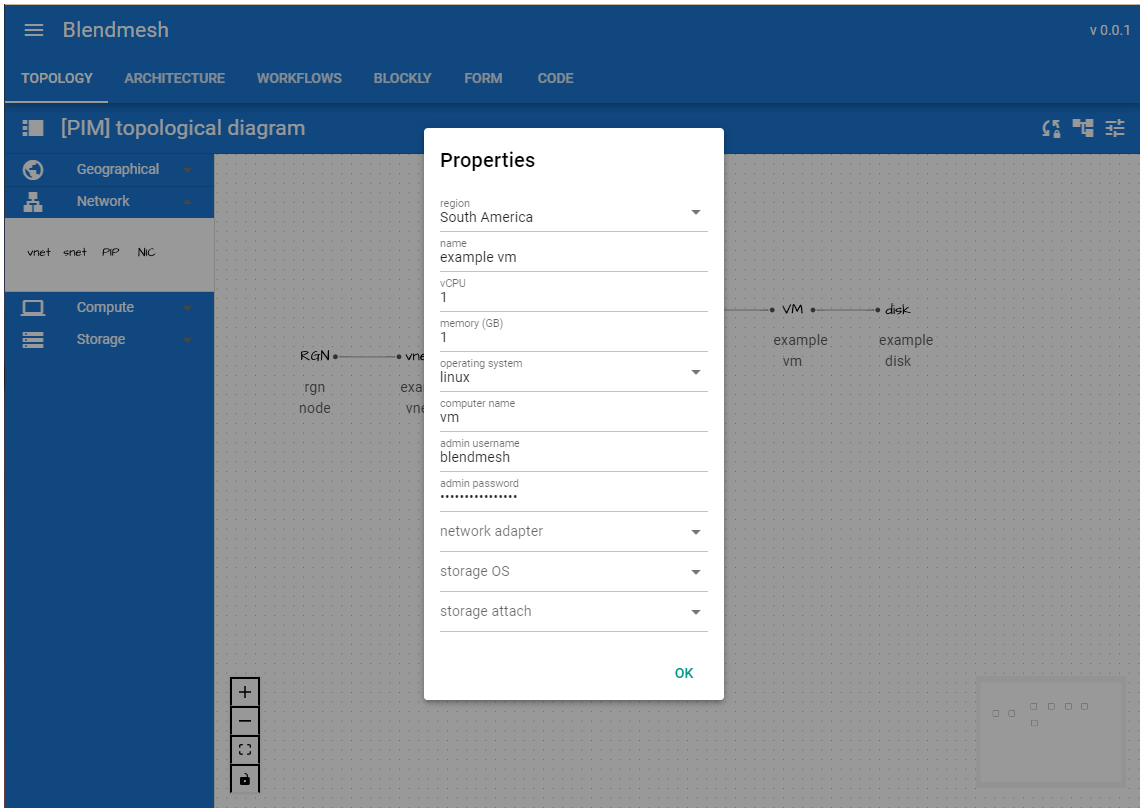



Fonte: Autor (2022)

Para atender profissionais que não tem familiaridade com codificação de infraestrutura como código, com diagramas topológicos ou arquiteturais, o tipo de representação de formulário pode ser utilizado (ver Figura 34). Esta representação é dividida em duas partes distintas, uma área a esquerda disponibilizando uma árvore de recursos e outra a direita para a manipulação do domínio utilizando as entradas de dados. Conforme novos recursos vão sendo adicionados ao projeto a árvore é atualizada, exibindo as respectivas informações de cada elemento e sua relação de parentalidade. Ao selecionar um recurso, suas propriedades se tornam editáveis nas entradas de dados a direita da tela e aplicadas ao pressionar o botão *apply*.

Outra característica incluída na interface é a possibilidade de visualizar o comportamento atual do estado do tipo de representação utilizada (ver Figura 35). Este recurso, apesar de não ser editável, apresenta como a ferramenta está tratando a relação

Figura 33 – Configuração das propriedades do recurso



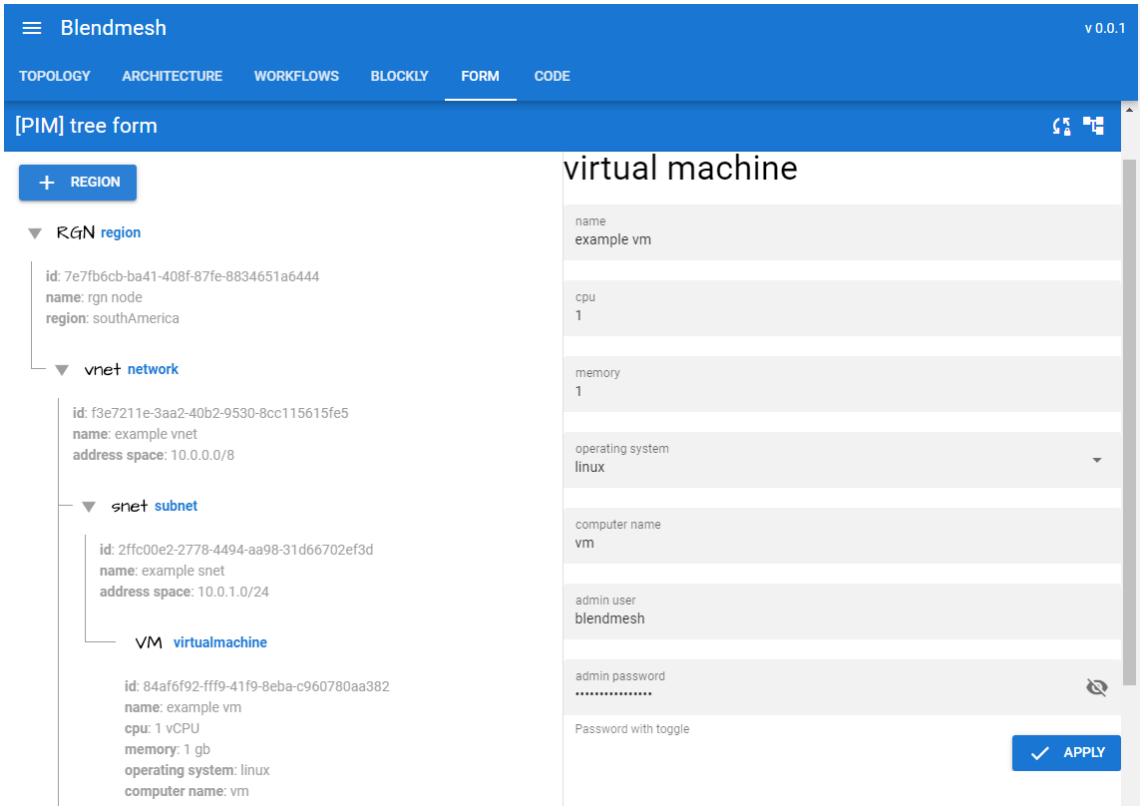

Fonte: Autor (2022)

entre os recursos provisionados e futuramente a organização do estado se assemelhará a de ferramentas análogas de padrão aberto, para facilitar sua exportação. Outros tipos de representações já estão definidos e com *frameworks* selecionados (ver Figura 36), como o tipo de representação de *workflow* abordando uma perspectiva de esteiras e o tipo de representação *blockly* que utilizará o *framework* do Google de mesmo nome (<https://developers.google.com/blockly>) abordando uma construção utilizando blocos.

## 4.6 Processo de codificação na plataforma

Para obter o código elegível para provisionamento, é necessário realizar três etapas na plataforma: elaborar o PIM, especificar o PSM e utilizar um transformador de código. Na primeira etapa é definida de forma genérica a arquitetura e são validados conceitualmente os recursos empregados, podendo utilizar uma representação topológica, arquitetural ou de formulário. Um exemplo de utilização desta etapa, em reuniões de projeto ou com o cliente, é comum criar esboços da arquitetura pretendida em softwares de diagramas ou de apresentação. Ao realizar o mesmo esboço desta etapa do PIM, mas

Figura 34 – Interface do tipo de representação de formulário



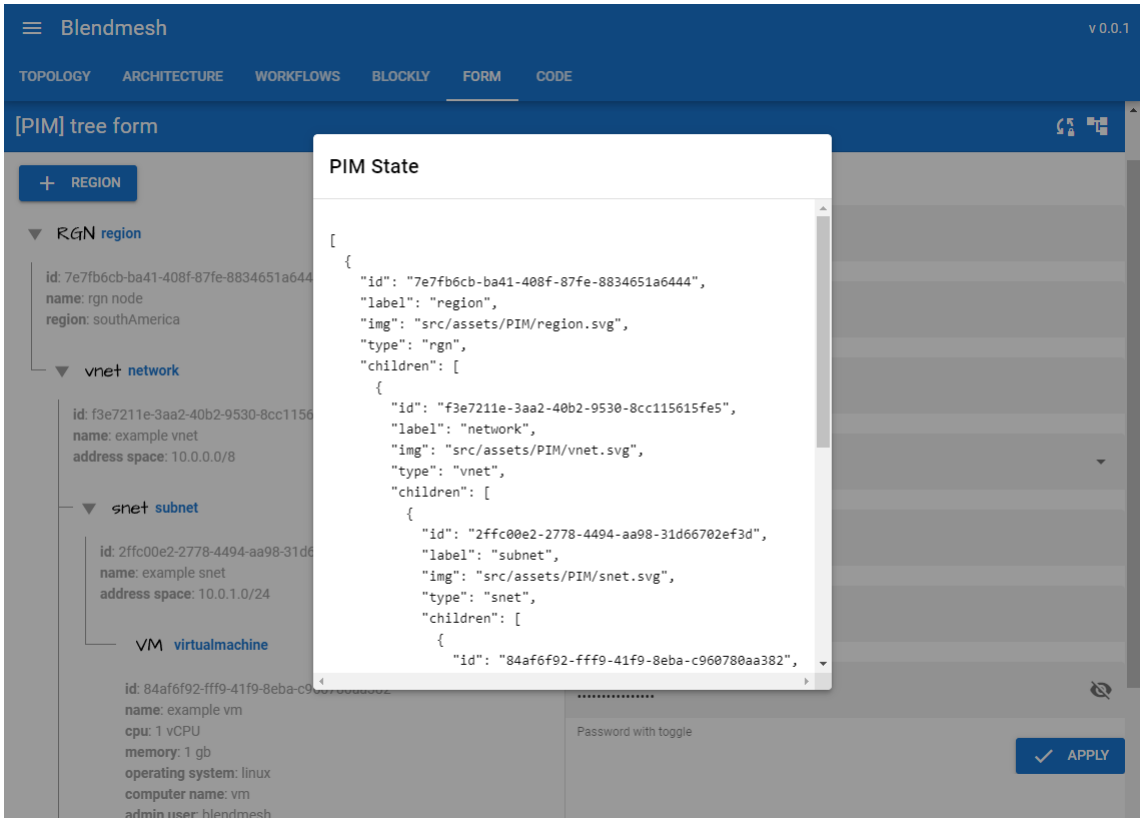

Fonte: Autor (2022)

utilizando a plataforma, no término da reunião é plausível possuir a arquitetura pronta ou grande parte dela e habilitada para ser especificada.

Para iniciar o processo de especificação da infraestrutura é necessário utilizar um transformador do tipo PIMtoPSM, que adéqua o projeto PIM para as estruturas PSM elencadas para uso no projeto. Para este trabalho foi desenvolvido o transformador pim4azure, que encontra o recurso mais adequado para ser provisionado na Azure conforme o que foi proposto no PIM. Ao acionar o transformador a sua interface sobrepõem a tela, relacionando os recursos, adequando ao seu correspondente na Azure, e em casos específicos pode solicitar alguma intervenção do usuário em temas como a definição de uma região mais adequada. Na Figura 37 pode ser observado o comportamento da interface, relacionando os recursos nas categorias *geographical*, *network*, *storage* e *computer*, onde cada uma possui dependência da anterior. Desta forma, para ter acesso a relação dos recursos de rede durante o processo de transformação é necessário sanar os impedimentos da categoria *geographical*.

Após finalizar o processo de transformação e estar utilizando a interface em modo PSM, o projeto pode receber mais algumas adições de recursos ou ser submetido para avaliação dos setores de segurança ou financeiro. Nesta etapa o funcionamento da

Figura 35 – Visualização do estado do tipo de representação

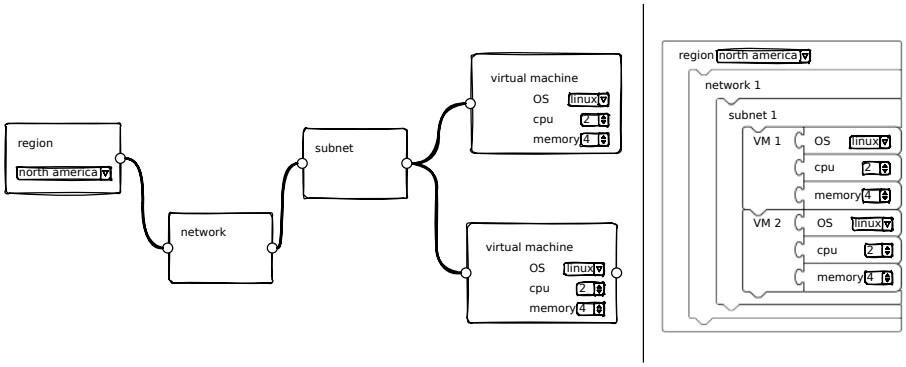



Fonte: Autor (2022)

interface é semelhante ao utilizado no PIM, mas com os recursos específicos do PSM escolhido e suas respectivas propriedades. Com a aprovação dos setores responsáveis, pode ser realizada a última etapa, a codificação. Para dar inicio a este processo é necessário clicar no botão que aciona o transformador do tipo PSMtoCode e o código é apresentado na tela e pronto para ser provisionado (ver Figura 38). Em versões futuras será possível acompanhar as modificações do código enquanto os diagramas são manipulados.

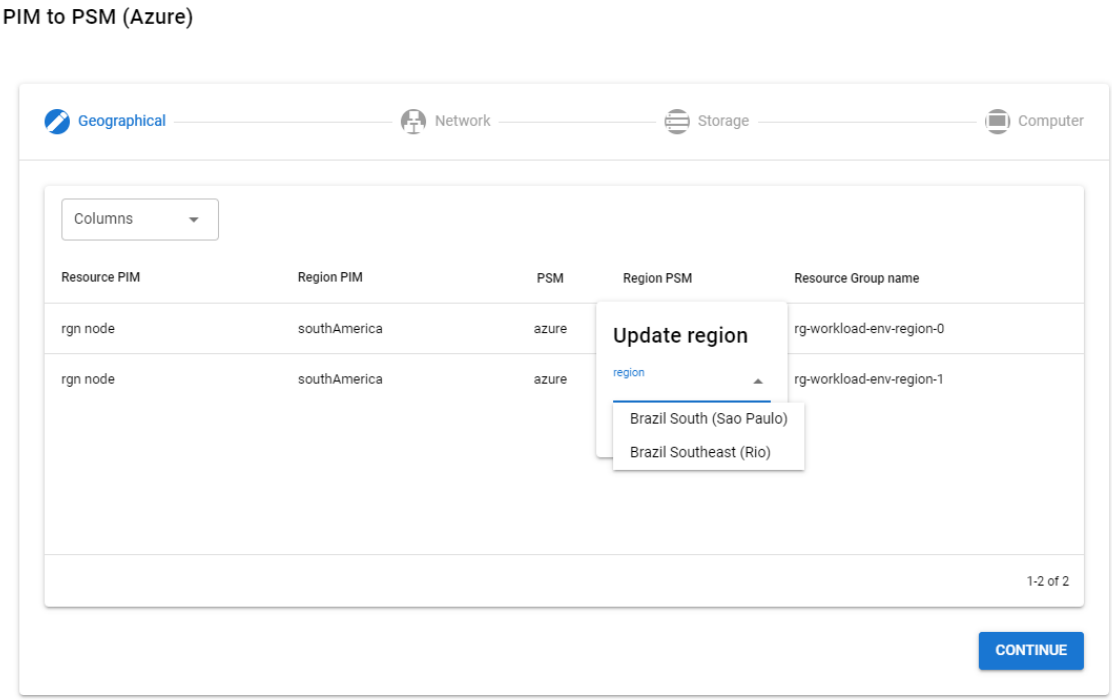
Em projetos pequenos ou treinamentos, durante a mesma reunião ou encontro, as infraestruturas já podem ser elaboradas, especificadas e provisionadas. Em casos de projetos de porte médio, grande ou complexos, é esperado que a submissão das arquiteturas para aprovação ou readequação seja mais ágil e assertiva.

Figura 36 – Representações futuras do tipo workflow (esquerda) e blockly (direita)



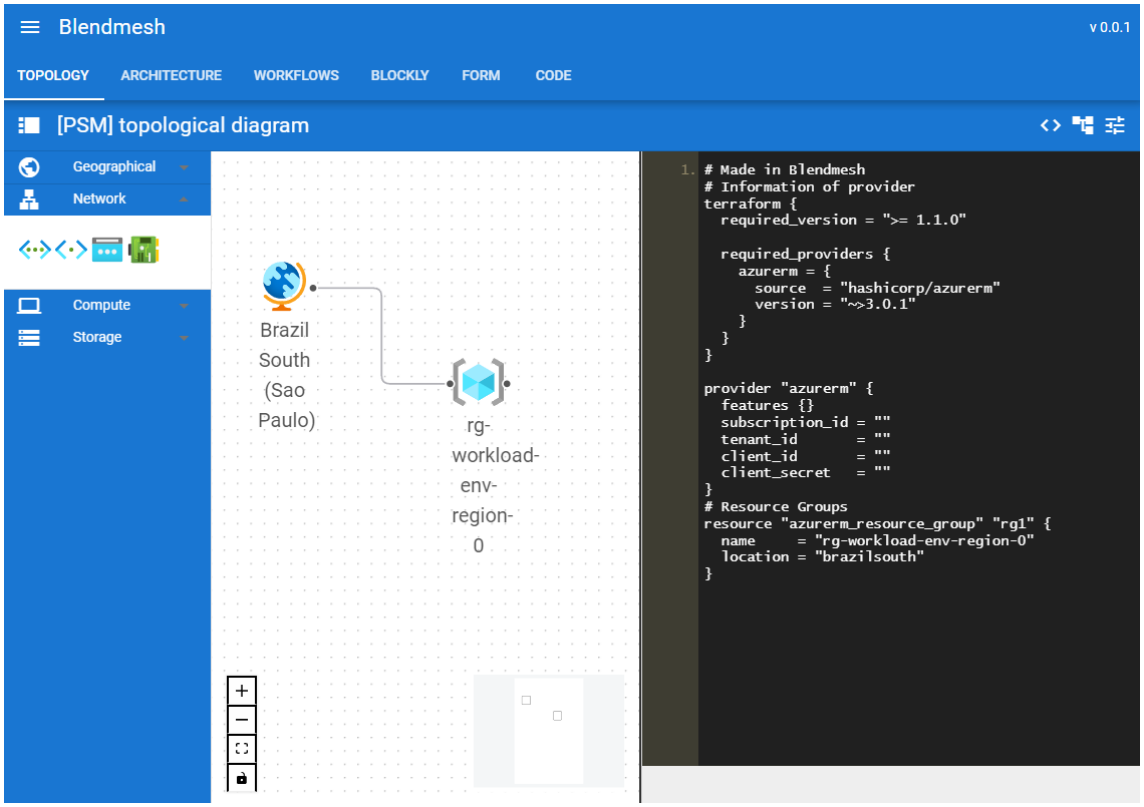

Fonte: Autor (2022)

Figura 37 – Utilização do transformador de PIM para PSM




Fonte: Autor (2022)

Figura 38 – Utilização do transformador de PSM para código




Fonte: Autor (2022)

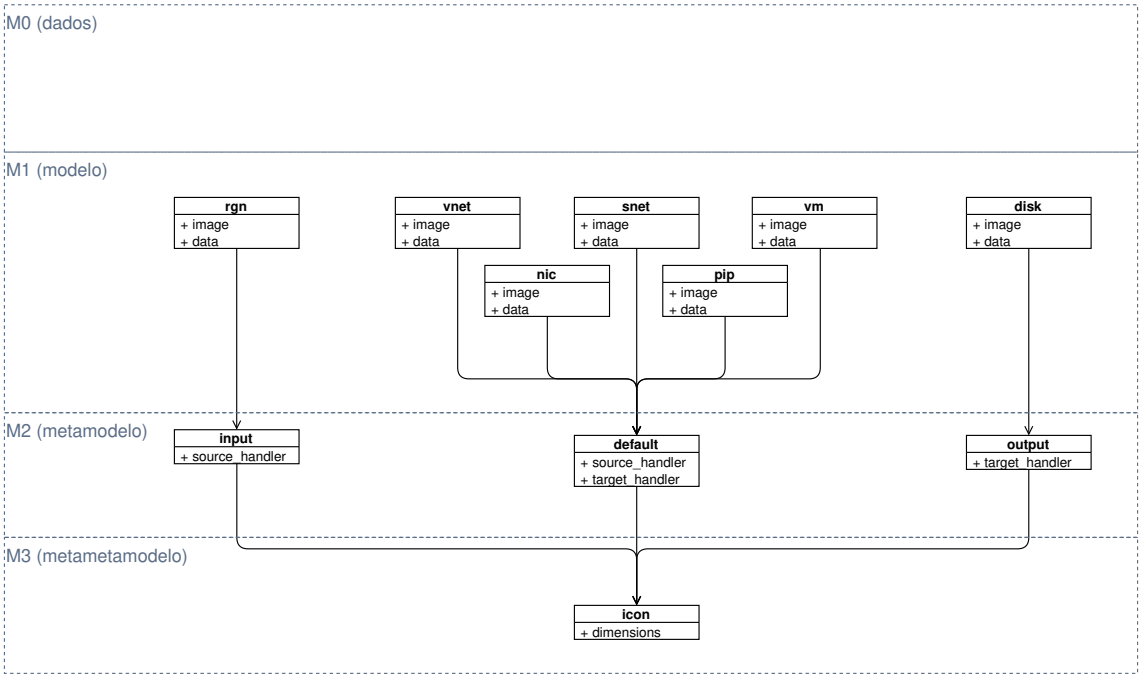
# 5 DISCUSSÃO DOS RESULTADOS

## 5.1 Estrutura e desenvolvimento do sistema

Com o escopo de diminuir o esforço necessário para provisionar infraestruturas e facilitar o uso de entrantes profissionais na área, foi necessário condensar o conhecimento para atuar e transformá-lo em uma linguagem. Mesmo tendo compilado somente uma fração do necessário para desempenhar a função de forma completa, a percepção do montante de informações requeridas chamou a atenção, criando uma maior empatia com os entrantes e seu desafio em assimilar o máximo possível em um curto espaço de tempo.

A percepção de como os recursos de infraestrutura se relacionam foi ampliada: configurações simples como de uma interface de rede podem escalar em complexidade muito rapidamente, gerar inúmeros novos cenários que escapam do escopo inicial e que precisam ser mapeados e analisados. Algo que a princípio pode parecer trivial, como a adição de um novo recurso à aplicação se demonstra desafiadora pelo número de relações necessárias que, e conforme são acrescentadas variações deste mesmo recurso, podem fazer o número de relações crescer drasticamente.

Figura 39 – Modelagem de um grupo de componentes




Fonte: Autor (2022)

A experiência do uso de orientação a modelos durante o trabalho agregou como uma ferramenta fundamental na tratativa de problemas de relacionamentos e mapeamento

de recursos. Com sua aplicação correta, a codificação se torna bastante simples e de fácil aplicação de testes. Na figura 39 pode ser observada a modelagem de componentes baseados no metametamodelo *icon*, seus três filhos herdaram seus atributos e seus testes, assim como seus netos. A mesma estrutura de código encontrada no *icon* foi replicada para seus sete netos, que possuem funções e comportamentos distintos, somente informando a sua imagem e um objeto *data* contendo as informações do domínio.

Além do MDA, outras metodologias e processos foram aplicados de formas indiretas como o *Domain-driven Design* que auxiliou em alguns processos de refatoração dos modelos, abordagens de arquitetura orientada a eventos para elaboração dos PSM pensando na utilização futura de API, e testes unitários dos recursos e estados inspirados em *Test-Driven Development*.

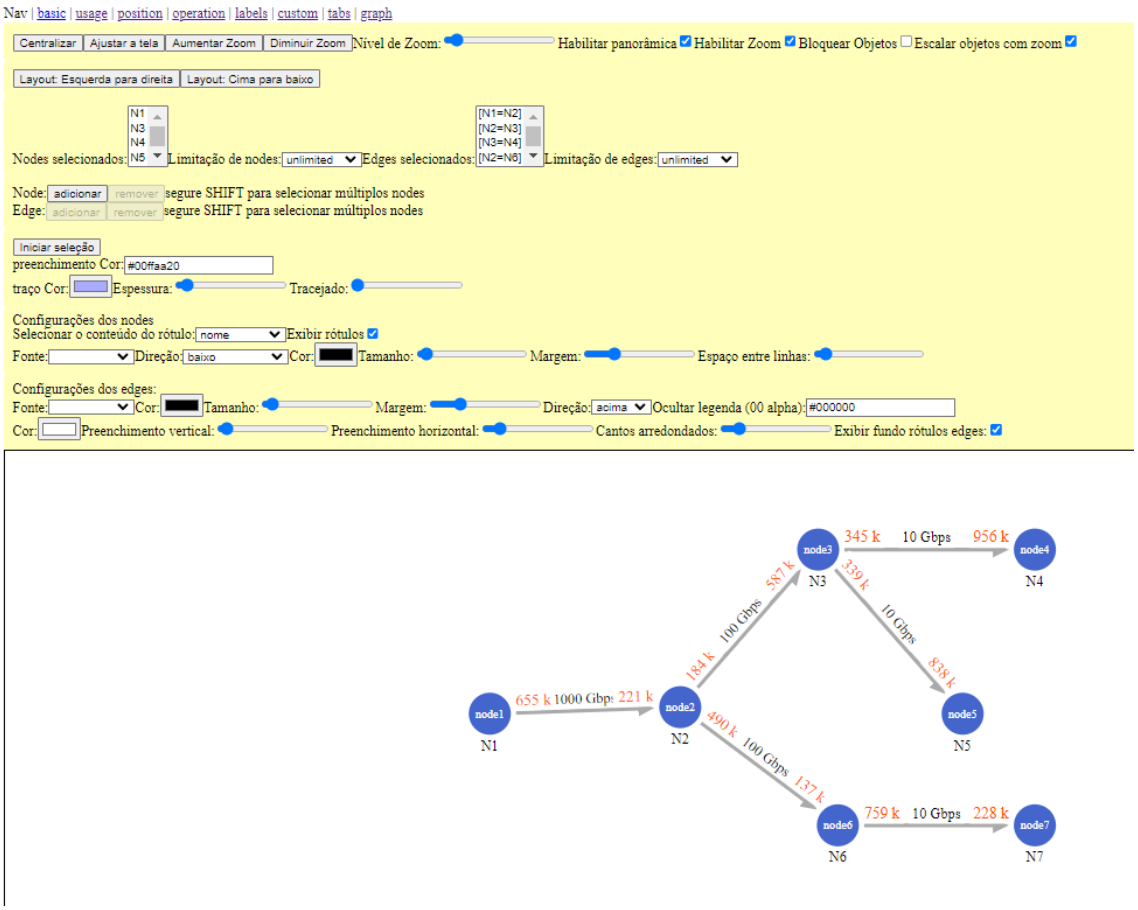
O mapeamento interno das classes e modelos também tiveram influências de *Object Constraint Language* (OCL), *Unified Modeling Language* (UML) e *Query/View/Transformation* (QVT), assim como a utilização de estruturas baseadas em *Design Patterns* e os pouco documentados padrões de arquiteturas em nuvem. Muitos destes possuem mais de vinte anos e não são atualizados há pelo menos dez anos. No inicio deste estudo ocorreu um receio de trabalhar com padrões tão antigos de desenvolvimento, mas ao compreender seus propósitos e realizando as devidas atualizações, se demonstraram eficientes na resolução dos problemas.

Além do desafio de trabalhar com diferentes metodologias para a modelagem da arquitetura, a interface gráfica e seu conjunto de dependências e versionamentos também requereram atenção. Não foi incomum componentes e *frameworks* se mostrarem incompatíveis em certas circunstâncias, desempenhando corretamente em compilações de testes e conflitando em compilações produtivas. Em versões futuras, para diminuir os impedimentos será adotado o emprego de versões *Long-term support* (LTS) das bibliotecas e *frameworks* e os que não possuírem terão uma de suas versões congeladas, mas para evitar problemas de incompatibilidades com navegadores e problemas de segurança também será adotado um calendário de atualização de ferramental base.

No decorrer da construção da interface mais recursos foram removidos que adicionados, motivados pelos impedimentos relacionados anteriormente, pela necessidade de refatorações e pelas sugestões de voluntários que testaram a ferramenta. Em sua versão inicial (ver figura 40), existia uma disponibilidade maior de customização dos elementos gráficos, como seus tamanhos, cores, rótulos ou tipos de fontes, por exemplo. Os recursos de rede também mantinham valores do tráfego possível, conforme

mais recursos iam sendo atribuídos na arquitetura. Em uma segunda versão (ver figura 41), a atualização de ponta a ponta do tráfego de rede foi removida pela complexidade de controle e consumo de recursos computacionais conforme o diagrama era incrementado. Grande parte de suas customizações foram removidas para dar lugar a itens mais estáticos, como ícones para facilitar a identificação e diminuir o número de variáveis controladas dentro do estado. Na versão da entrega deste trabalho, as informações contidas nas arestas foram removidas e também a maioria dos recursos de customização, centrando o foco na modelagem do domínio e seus transformadores.

Figura 40 – Primeira versão da interface gráfica

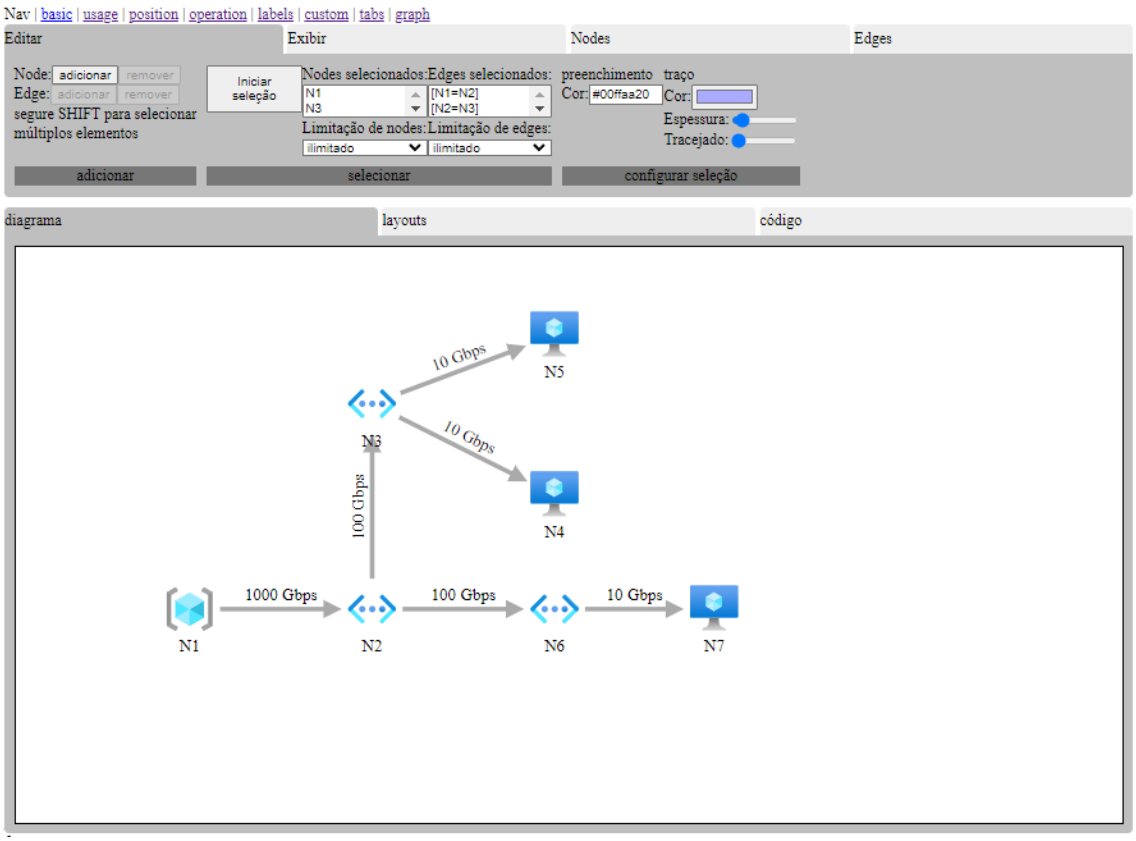



Fonte: Autor (2022)

## 5.2 Percepções dos usuários

Após o amadurecimento da interface gráfica, contendo tanto o PIM quanto o PSM parcialmente implementados, seu uso foi submetido a testes de usuários de maneira informal. O processo consistiu na apresentação da plataforma, instigando o seu uso e

Figura 41 – Segunda versão da interface gráfica




Fonte: Autor (2022)

coletando comentários sobre possíveis melhorias. As abordagens ocorreram em eventos, reuniões e treinamentos sobre o tema, verificando se a interface era intuitiva o suficiente para que entendessem a proposta de uso e chegassem ao fim do processo sem dúvidas.

Alguns impedimentos no decorrer do desenvolvimento da interface e do gerenciamento de estados ocasionaram mudanças no processo de aplicação da pesquisa com usuários, alterando a ideia inicial de um grupo de controle, com consultas e coleta de dados regulares, para consultas espontâneas de indivíduos, seguido da demonstração de uso da ferramenta. Nesse novo processo, pessoas que desempenham funções como analista, engenheiro ou arquiteto em empresas de serviços de transformação digital eram abordadas e convidadas a testar um software para provisionamento de infraestrutura em nuvem. Ao todo, participaram do processo dezessete (17) pessoas com diferentes níveis de experiência, maturidade profissional e formação, em sua grande maioria profissionais júnior e plenos, em formação acadêmica e com certificações na área.

O tempo médio da abordagem foi de quinze minutos, iniciando com o teste cego, seguido de uma demonstração de uso, novamente solicitado para realizar um projeto e o finalizando com as críticas sobre a ferramenta. O teste cego consumiu de três a cinco

minutos, onde sem nenhuma instrução prévia, foi solicitado abrir a ferramenta e arquitetar uma rede com uma máquina virtual. Quando percebida dificuldades no uso, era realizada uma demonstração do fluxo da ferramenta, arquitetando no PIM, transformando para o PSM e devolvendo o código, o que consumiu em média cinco minutos. Munidos das instruções de uso, foi pedido para repetir a arquitetura anterior e solicitada as críticas, levando de cinco a dez minutos as duas últimas etapas.

Os conceitos de PIM e PSM não foram identificados em um primeiro momento e somente após a demostração do seu uso foram reconhecidos. Quando questionados sobre a não percepção dos dois modelos e sua transição, a resposta mais recorrente foi que não conheciam este tipo de abordagem sobre o assunto. Quanto ao uso de transformadores, tanto entre camadas, quanto para código, também foi relatado não esperar que a ferramenta realizasse este tipo de processo, somente tinham conhecimento de ferramentas com focos mais específicos e não como esta que aborde um número significativo de contextos.

Sintetizando as críticas recebidas, uma grande parte dos testadores não reconheceram a proposta da plataforma em um primeiro momento, em parte por presumir que seria uma ferramenta de diagramas ou por considerar que era uma interface gráfica para converter códigos Terraform. Ferramentas visuais para arquitetura em nuvem não foram encontradas no mercado durante o período de escrita deste trabalho, somente estudos ou algumas com focos bastante específicos, neste contexto o usuário não consegue encontrar experiências familiares para serem associadas. Uma nova rodada de testes com enfase na experiência de usuário pode validar esta hipótese e obter um melhor embasamento para a solucionar este item. Com estas observações dois pontos de atenção precisarão ser trabalhados em atualizações futuras: primeiro, deixar mais aparente a proposta da plataforma como suas possibilidades de uso, e o segundo ponto tornar mais familiar termos vindo da modelagem orientada a arquitetura, como modelos independente ou específicos de plataforma e seus benefícios ao empregá-los.

## 5.3 Estrutura de pastas, *branches* e versionamento

Durante a realização de testes de integração em ambiente online e disponibilização da plataforma para testadores, o seu custo operacional se tornou um obstáculo a ser transposto. Três pontos foram os grandes problemas: o uso elevado de CPU, um consumo acima do esperado de memória e o tamanho da imagem final da aplicação.

Em ambientes de nuvem, a alocação de recursos computacionais como memória RAM, processamento e armazenamento é precificada dependendo do recursos de nuvem, pela sua quantidade empregada, por tempo de uso ou ambas as situações. No caso de processamento, a unidade mais comum é o vCPU, sendo um núcleo físico do processador onde o recurso de nuvem está sendo executado e, em algumas situações, envolvendo o uso de *containers* esse valor pode ser fracionado. O uso de memória RAM vai depender do tipo de recurso de nuvem que será empregado, podendo estar na casa dos megabytes quando se utilizar *containers* simples e gigabytes quando se utiliza máquinas virtuais, por exemplo. O armazenamento pode impactar em duas áreas em relação a custos, no próprio espaço ocupado e dependendo da situação no tráfego de rede quando os dados armazenados forem utilizados.

A plataforma desenvolvida neste trabalho foi projetada para ser executada em containers Linux executando nodeJS, um servidor HTTP e a aplicação em si. Neste cenário, os recursos computacionais deveriam ser relativamente baixos e seus custos proporcionais a eles. O principal motivo mapeado para este incidente foi a disponibilidade de recursos computacionais superiores na estação de desenvolvimento, comparado aos disponíveis no recurso de nuvem contratado. Ao realizar os testes na estação de trabalho, o consumo de recursos acima do esperado não foi percebido, fato que poderia ter sido evitado monitorando ou limitando o uso de recursos no containers de testes. Um motivo secundário observado foi que o processo de codificação estava direcionado em obter o resultado esperado dos modelos e não em manter o seu código otimizado.

Para diminuir o consumo de CPU foi necessário realizar refatorações em algumas partes da aplicação, principalmente nos componentes, procurando atingir o menor acoplamento possível entre eles, fazendo uso intensivo das regras e funções de mapeamento propostas no MDA. Empiricamente, o consumo de CPU foi diminuindo gradativamente conforme os conceitos de mapeamento foram aplicados, mas não foram realizados testes suficientes para sustentar esta hipótese. Um outro ponto que foi mapeado como ofensor no consumo de CPU, mas não foi tratado, é o emprego de algumas rotinas de repetição como o *loop*. Para amenizar o uso de rotinas de repetição, para esta aplicação, será necessário rever o uso de mutações entre os componentes e os estados em refatorações futuras. Depois das intervenções realizadas, o requisito mínimo foi de 3 para 1 vCPU.

O consumo de memória foi amenizado com o emprego do gerenciador de estado e uma maior atenção nos elementos responsáveis pela renderização da página no navegador

quando o *Document Object Model* (DOM) era manipulado. Com o gerenciador de estado, a orquestração do uso da memória no que tange à manipulação do estado foi delegado para ele, desempenhando de maneira mais eficiente que o processo manual. O processo de renderização tem como seu principal ofensor a atualização da posição dos elementos na tela, outros itens que impactavam foram sanados com as aplicações dos conceitos de mapeamentos do MDA. Existe espaço para melhoria nos itens de renderização, mas será necessário reescrever uma parte do *framework* utilizado, o que não é escopo deste trabalho. Ao final o consumo de memória que antes estava na casa dos gigabytes, agora já opera na casa dos megabytes.

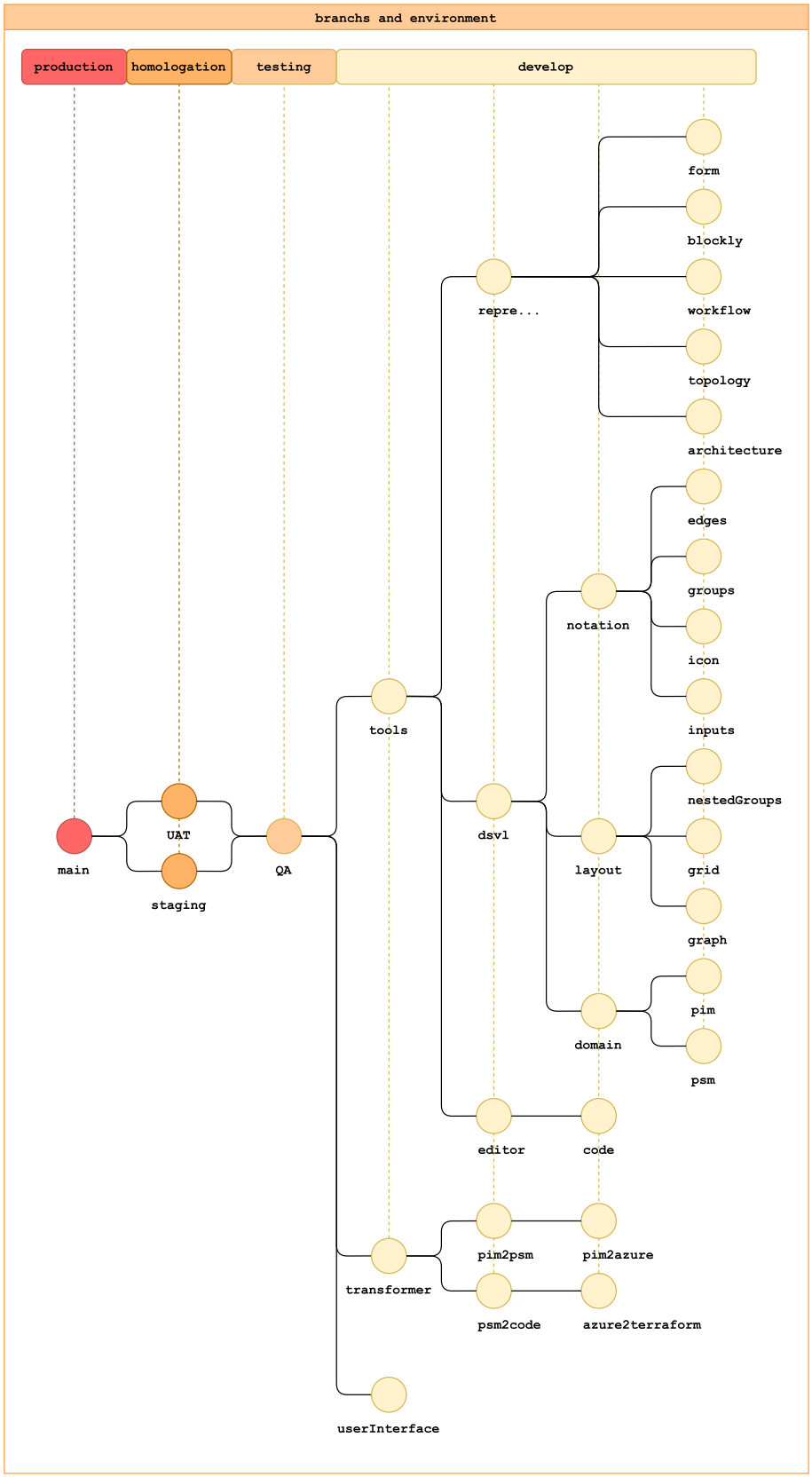
Para a redução do tamanho da imagem de trabalho foi necessário atuar na configuração do compilador da linguagem, no processo de montagem da imagem, na estrutura de pastas do repositório (ver figura 42) e suas *branches*. Durante de processo de compilação foi solicitado que o conteúdo que for estático seja pré-compilado e utilizando somente os binários necessários para a construção da imagem. Foi adotado o processo de estágios para montar a imagem, apesar de ser mais demorado e custoso, onde no primeiro estágio realiza a compilação do código (*build-stage*) e no segundo estágio (*production-stage*) anexa o resultado da compilação a um servidor HTTP. Como etapa final, em conjunto com as medidas aplicadas de refatoração mencionadas anteriormente, foi realizada uma reorganização da estrutura de pastas do projeto, obedecendo uma estrutura modular e facilitando a atualização. Com estas medidas o tamanho da imagem diminuiu de 1,3GB para 34MB.

Para manter o tamanho da imagem reduzida e organizar o fluxo das futuras atualizações, o repositório do projeto foi dividido em *branches* para cada módulo da aplicação, podendo realizar testes sem afetar a versão estável (ver figura 43). Para ter um controle do que está sendo testado e o que será incluído nas versões produtivas, está sendo empregado um sistema de versionamento adaptado do *Semantic Versioning* 2.0.0 (<https://semver.org/>).

Figura 42 – Reestruturação das pastas do repositório (esquerda) e detalhe da pasta componentes (direita)


Fonte: Autor (2022)

Figura 43 – Organização dos ambientes e *branches* dos módulos do projeto




Fonte: Autor (2022)

# 6 CONSIDERAÇÕES FINAIS

## 6.1 Conclusões

As dificuldades para o preenchimento de postos de trabalho enfrentadas pelo mercado de tecnologia têm se acentuado nos últimos anos, com uma alta taxa de rotatividade de pessoas e carência de pessoal qualificado. A dificuldade no preenchimento de vagas tem resultado em dois problemas pontuais: falta de experiência das equipes e falhas de comunicação entre clientes e equipe e entre os membros das próprias equipes. Com a necessidade de preencher as vagas para atender as demandas, a contratação de pessoas sem a experiência necessária tem aumentado, fazendo com que, além do treinamento padrão para ocupação do cargo, tenham que ser realizados treinamentos extras para aprendizado dos conceitos relacionados às tecnologias utilizadas nas empresas.

Com um número crescente de pessoas sem experiência atendendo demandas que necessitam de um conhecimento prévio para executá-las, profissionais mais experientes são mais exigidos, tendo que monitorar o processo de aprendizado dos novos colaboradores enquanto que continuam a atender suas demandas usuais. O acúmulo de trabalho acarreta estresse e *burn-out* desses profissionais. Outro fator é a difícil comunicação com pessoas de níveis e áreas do conhecimento diferentes, com foco e expectativas singulares. Estes dois pontos ocasionam divergências entre o resultado das entregas, especificações errôneas e o não cumprimento dos prazos dos projetos, gerando retrabalho, insatisfação dos clientes e prejuízo para a empresa.

O objetivo desse trabalho foi intervir nessa situação, desenvolvendo uma ferramenta para projetar arquiteturas em nuvem de forma simplificada que possa atender aos novos colaboradores, os guiando no processo. Pretende-se, assim, diminuir erros de projeto e sobrecarga de trabalho dos colaboradores mais sêniores. Também atende os mais experientes excluindo as etapas de codificação e documentação, além de auxiliar na comunicação entre profissionais de áreas diferentes, por meio de geração automática de código e documentação de sistemas, fazendo uso de desenvolvimento orientado a modelos e linguagens de domínio específico.

Para atender aos objetivos, o trabalho foi dividido em três etapas: a revisão da literatura, a modelagem do problema com aprendizado dos conceitos necessários e a escrita do trabalho. Após a revisão da literatura, onde o foco recaiu em encontrar

projetos correlatos e o emprego de DSL, um novo leque de opções e abordagens foram encontradas para tratar do tema de provisionamento de infraestruturas. As perguntas iniciais respondidas deram abertura para novas questões de desafios a serem solucionados e com a utilização de uma metodologia de desenvolvimento de software apoiando o processo, todo o modelo inicial teve de ser revisitado. O projeto assumiu um formato mais sólido com a divisão em etapas bem distintas, para cadenciar a maneira que a abstração do fluxo de trabalho vai se transformando em uma infraestrutura funcional e apontando mais claramente onde as associações devem ser realizadas a cada etapa. Mas esse novo modelo de trabalho veio com o custo de reduzir o escopo de entrega, pois sua implementação requer mais atenção e, principalmente, um tempo maior de aprendizado.

O escopo pretendido inicialmente foi referenciado durante o texto, evidenciando o que de fato foi implementado no transcorrer deste trabalho e o que é esperado para versões futuras dele. Ao decorrer do desenvolvimento da estrutura base também foi verificada a necessidade de atenção de diversos outros pontos, como exposição de informações sensíveis, estratégias que têm que acompanhar os modelos de negócios, estar em conformidade com algumas leis e normas, análises financeiras que impactam a viabilidade do provisionamento, impacto que os diferentes modelos de planejamento estratégico trazem, os custos deste próprio trabalho, entre vários tópicos que não são abordados no texto para não desviar do tema central mas que impactam no resultado final.

No desenvolvimento da DSVL, mais especificamente, a criação de metamodelos que satisfizessem tanto o seu domínio de atuação quanto a mudança de contexto, foi algo singular durante o desenvolvimento deste estudo. Requerendo um amalgama interdisciplinar, como a aplicação dos conceitos de disciplinas como estrutura de dados, programação orientada a objetos, engenharia de software, lógica para computação, linguagens formais, teoria da computação, análise e projetos de algoritmos e engenharia de software, para mapear os modelos, definir suas estruturas, possibilidades de reuso, heranças, dependências e suas classes de complexidades. Assim como as disciplinas de cálculo, probabilidade e estatística empregadas para determinar qual impacto do comportamento crescente das relações entre os recursos. Para sustentar toda a construção do domínio para infraestruturas em nuvem definido no escopo do trabalho, fazendo uso de disciplinas como arquitetura e organização de computadores, sistemas operacionais, banco de dados e redes de computadores.

Na data de escrita deste trabalho, o desenvolvimento *frontend* tem a disposição

uma miríade de opções e necessitando ser resiliente na criação de seu projeto. O mapeamento de dependências entre *frameworks*, bibliotecas, *plugins* e linguagens motivaram a definições de regras para seus usos e a um versionamento antecipado no projeto, reduzindo a necessidade de refatorações e atrasos. O emprego de um gerenciador de estados trouxe uma maior flexibilidade ao trabalhar com os diferentes tipos de representações, separando em estados diferentes o que o domínio do estudo exigia e o que um grupo de bibliotecas necessitavam, podendo intercambiar entre os diferentes tipos de contextos sem maiores impactos.

Com a aplicação adaptada da pesquisa-ação foi disponibilizando pouco tempo para o usuário testar a interface e explorar os recursos apropriadamente. Foram obtidos comentários espontâneos relevantes, mas sem uma reflexão mais detalhada do problema que a plataforma aborda, este fato impactou no resultado dos testes, não o invalidando, mas não oportunizando o usuário relatar uma experiência de uso mais consistente.

Algumas etapas durante o decorrer do estudo necessitaram ser readequadas por imprevistos no desenvolvimento, como a forma de abordagem da pesquisa-ação, redefinição dos estados da plataforma e problemas de otimizações. Mas soluções foram encontradas em tempo hábil para não requerer alterações no cronograma determinado em seu inicio, fazendo uso de metodologias para tomada de decisões apresentadas em disciplinas como engenharia de software, controle estatístico do processo e estratégia organizacional.

Todo um conjunto de modelos, componentes e regras que resultaram deste trabalho, atenderam a expectativa de ter uma base sólida para a plataforma e o desenvolvimento de uma linguagem visual específica para o domínio, ambos baseados em métodos amplamente conhecidos e validados. Como foi um processo experimental itens de atenção eram esperados, vários pontos de melhorias ou que necessitam de reconstrução foram mapeados após seu término, assim como suas soluções.

## 6.2 Trabalhos futuros

A plataforma idealizada tem um escopo maior que o resultado deste trabalho, sendo consideradas cinco grandes áreas de atuação: sistema de tomada de decisão, camadas arquiteturais, simulações, *DevOps* e operação. Neste estudo foi abordado uma fração da área de camadas arquiteturais e que mantém um grande espaço para melhorias.

Para a área de atuação do sistema de tomada de decisões, existem três demandas

mapeadas. A primeira é o desenvolvimento de uma DSL direcionada para o levantamento de requisitos e cargas de trabalho junto aos ambientes das empresas, assim como agentes para inventariar seus recursos e poder computacional. Também a construção de um modelo de *machine learning* para determinar as melhores cargas de trabalho possíveis dado um conjunto de limitações, levando em consideração as melhorias necessárias e eventuais *data lock-ins*. E por último, a definição de uma análise preditiva que envolva *machine learning*, modelagem estatística e mineração de dados para avaliar os riscos e a viabilidade da implantação das cargas de trabalho definidas.

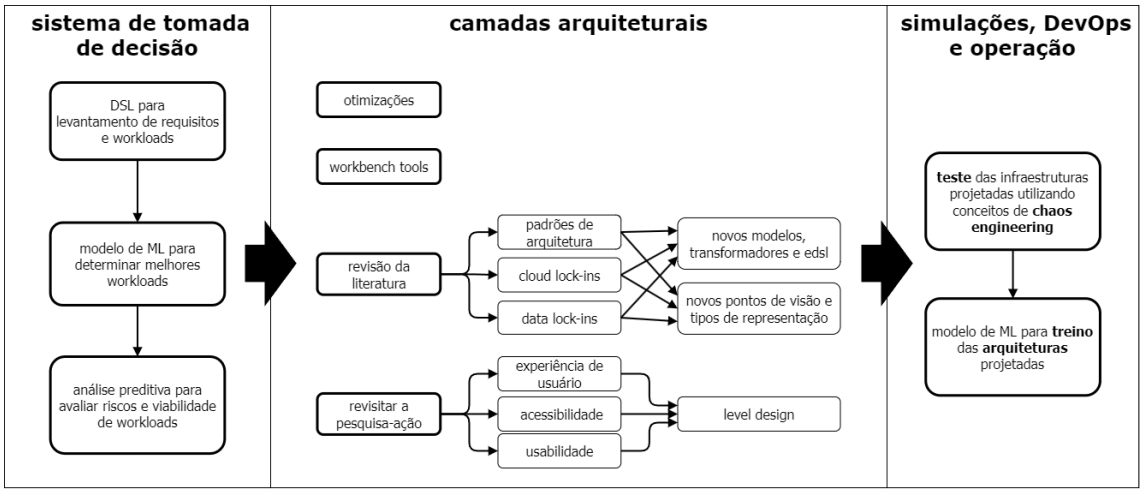
Para a área de camadas arquiteturais, que parte dela foi o foco deste estudo, temos os itens citados durante o trabalho, como a realização de otimizações nos modelos. Para este fim, uma aplicação mais intensa dos conceitos de mapeamento, fazendo uso de suas funções, regras e dos modelos de marcação. Neste ponto o uso de algumas EDSL, para tratamento de estruturas e regras que fogem do escopo do modelo é desejado. Com modelos mais estruturados e menos genéricos, o desenvolvimento de uma analisador léxico e *parser* próprio para a DSVL auxiliará na confiabilidade de seus resultados e desempenho. Outros trabalhos cogitados são a realização de revisões de literatura sobre os padrões de arquitetura em nuvem disponíveis, sobre os *cloud lock-ins* entre os provedores de nuvem e sobre os *data lock-ins* de ferramentas destinadas a um mesmo fim. Estas revisões auxiliarão no desenvolvimento de transformadores mais robustos e assertivos.

Dentro das camadas arquiteturais existe escopo para o desenvolvimento de monitores de projeto, que validarão se a arquitetura está atendendo os requisitos levantados, cumprindo as diretrizes do *well-architected* dos provedores, os custos e possíveis problemas de projeto. Também necessita da expansão do PSM da Azure, como a criação para outros provedores e ferramentas de bancos de dados ou orquestração de *containers*, como também o desenvolvimento de transformadores do tipo módulo-para-documento. Este último transformador atuará na confecção da documentação de suporte, na criação de planos de arquitetura, gestões de mudança ou relatórios financeiros. Também necessitam implementar a rastreabilidade dos artefatos entre as camadas arquiteturais, importar arquivos Terraform e integrá-los ao projeto, integrar duas ou mais arquiteturas projetadas, mapeamento das cargas de trabalho de projetos em BPMN, gráfico sinalizando a evolução da abstração para o concreto, criação de API para conexão com outras ferramentas e versões futuras *mobile* e *desktop*, implementação dos outros pontos de visão, desenvolvimentos dos outros tipos de representação, criação de ferramentas de *workbench* para o desenvolvimento de

transformadores e novos modelos, implementação do monitor de contexto em tempo de execução e realizar a pesquisa-ação de maneira formal com processos de aplicação e métricas.

Nas áreas de atuações simulações, *DevOps* e operação, foi mapeada somente a implantação de modelos *machine learning* para treino das arquiteturas projetadas e a implementação de conceitos de *chaos engineering* nos testes de infraestrutura, mas mantém um escopo aberto para integração com os diversos padrões de utilizados no *DevOps*, monitoramento e observabilidade. Na figura 44 é apresentado de forma condensada as propostas futuras.

Figura 44 – Propostas de trabalhos futuros




Fonte: Autor (2023)

Após atender aos objetivos acadêmicos da plataforma, foi definido um roteiro para iniciar seu uso em ambientes corporativos e para tal, algumas partes do projeto precisam ser congeladas e outras ter seu uso estabilizado antes de receber novos recursos. A interface de usuário atual cumpre o seu objetivo de uso da ferramenta, mas necessita ser revisitada considerando conceitos de experiencia de usuário, acessibilidade e usabilidade. Uma abordagem pretendida futuramente é o emprego de conceitos de *level design*, visando projetar a interface de modo a conduzir o seu uso e se adaptando ao grau de experiência do usuário e sua área de atuação. Também é desejado contribuir com a atualização de tópicos de algumas metodologias que se encontram desatualizada para os paradigmas atuais, como o MDA. O projeto tem um escopo bastante amplo, podendo explorar este tema no futuro em mestrado e doutorado, como em projetos encubados em ambientes corporativos.

# REFERÊNCIAS

ALMORSY, M.; GRUNDY, J.; RÜEGG, U. Horuscml: Context-aware domain-specific visual languages designer. **Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC**, p. 133–136, 2014. Doi:10.1109/VLHCC.2014.6883035.

ARDAGNA, D. *et al.* Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. **ICSE Workshop on Modeling in Software Engineering (MISE 2012)**, p. 50–56, 2012. Doi:10.1109/MISE.2012.6226014.

BRIKMAN, Y. **Terraform - up running - writing infrastructure as code**. Sebastopol: O'Reilly Media, 2019.

COUGHLAN, P.; COGHLAN, D. Action research for operations management. **International Journal of Operations & Production Management**, MCB UP Ltd, v. 22, n. 2, p. 220–240, Jan 2002. ISSN 0144-3577. Disponível em: <https://doi.org/10.1108/0144357021041751.>

DEED/INEP. **Notas Estatísticas do Censo da Educação Superior 2019**. Brasília, DF, 2020. 32 p.

DERMEVAL, D.; BITTENCOURT, J. A. P. M. C. I. I. Mapeamento sistemático e revisão sistemática da literatura em informática na educação. In: ____. **Metodologia de Pesquisa Científica em Informática na Educação: Abordagem Quantitativa**. [S.l.]: CEIE/SBC, 2020. v. 2, cap. 3, p. 1–26.

EHRIG, H. *et al.* **Handbook of Graph Grammars and Computing by Graph Transformation**. Singapore: World Scientific, 1997. v. 2 (Applications, Languages and Tools). 698p.

ERDWEGD, S. *et al.* Evaluating and comparing language workbenches - existing results and benchmarks for the future. Elsevier, 2015.

FEHLING, C. *et al.* **Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications**. Springer Vienna, 2014. ISBN 9783709115688. Disponível em: <https://books.google.com.br/books?id=aum9BAAAQBA.>

FERRY, N. *et al.* Cloudmf: Model-driven management of multi-cloud applications. **ACM Trans. Internet Technol.**, Association for Computing Machinery, v. 18, 2018.

FERRY, N. *et al.* Cloudmf: Applying mde to tame the complexity of managing multi-cloud applications. p. 269–277, 2014.

FOWLER, M. **Domain Specific Languages**. Boston: Addison-Wesley Professional, 2010. 640 p.

GERHARDT, T. E.; SILVEIRA, D. T. **Métodos de Pesquisa**. Porto Alegre: Editora da UFRGS, 2009. 115 p. Disponível em: <https://www.lume.ufrgs.br/handle/10183/5280.>

GRUNDY, J. *et al.* Generating domain-specific visual language editors from high-level tool specifications. p. 25–36, 2006.

GRUNDY, J. C. *et al.* Generating domain-specific visual language tools from abstract visual specifications. **IEEE Transactions on Software Engineering**, v. 39, p. 487–515, 2013.

HAMDAQA, M.; TAHVILDARI, L. The (5+1) architectural view model for cloud applications. IBM Corp., p. 46–60, 2014.

HINZE, P. **Applying Graph Theory to Infrastructure**. 2016. Disponível em: Disponível em: <ttps://www.youtube.com/watch?v=4Pd9NrZSbGU>. Acesso em: 26 abr de 2022.

JAVASCRIPT Object Notation. Disponível em: Disponível em: <https://json.org/json-pt.html>. Acesso em: 26 de abr de 2022.

KHALAJZADEH, H. *et al.* End-user-oriented tool support for modeling data analytics requirements. p. 1–4, 2020.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews**. Keele, 2004.

KLEPPE, A. G. **Software Language Engineering : Creating Domain-Specific Languages Using Metamodels**. London: Addison-Wesley Professional, 2008. 240 p.

KLEPPE, A. G.; WARMER, J. B.; WARMER, W. B. J. **MDA Explained The Model Driven Architecture : Practice and Promise**. Boston: Addison-Wesley Professional, 2003. 170 p.

KOURZANOV, P. Dsl methods for cps simulation in the cloud: Experience report. p. 25–32, 2013.

KURTEV, I. *et al.* Model-based dsl frameworks. Association for Computing Machinery, p. 602–616, 2006.

LÄMMEL, R. **Software Languages - Syntax, Semantics, and Metaprogramming**. Cham: Springer International Publishing AG, 2018. 424 p.

LEWIS, H. R.; PAPADIMITRIOU, C. H. **Elements of the Theory of Computation**. 2. ed. Upper Saddle River: Prentice Hall, 1998. 361 p.

MAHMOOD, Z.; RAMACHANDRAN, M. Fog computing: Concepts, principles and related paradigms. In: ____. **Fog Computing: Concepts, Frameworks and Technologies**. Cham: Springer International Publishing, 2018. p. 3–21. ISBN 978-3-319-94890-4. Disponível em: <https://doi.org/10.1007/978-3-319-94890-4_.>

MELLOR, S. J. *et al.* **MDA Destilada: Princípios de Arquitetura Orientada por Modelos**. Rio de Janeiro: Ciencia Moderna, 2005.

MODEL-DRIVEN Approach for design and execution of applications on multiple Clouds. CORDIS. Disponível em: Disponível em: <https://cordis.europa.eu/project/id/318484>. Acesso em: 26 de abr de 2022.

MORAIS, C. M. de *et al.* A modelling language for defining cloud simulation scenarios in recap project context. p. 301–302, 2018.

MORRIS, K. **Infrastructure as Code: Dynamic Systems for the Cloud Age**. 2. ed. Sebastopol: O'Reilly Media, 2021.

MORRIS, K. **Infrastructure as Code: Dynamic Systems for the Cloud Age**. Sebastopol: O'Reilly Media, 2021.

MUNN, Z. *et al.* Systematic review or scoping review? Guidance for authors when choosing between a systematic or scoping review approach. **BMC Medical Research Methodology**, v. 18, 2018. <https://doi.org/10.1186/s12874-018-0611-x>.

NITTO, E. D. *et al.* **Model-Driven Development and Operation of Multi-Cloud Applications**. [S.l.]: Springer Cham, 2017. 149 p. (SpringerBriefs in Applied Sciences and Technology).

NITTO, E. D.; SOLBERG, A.; PETCU, D. On MODAClouds' Approach for Application Design and Execution on Multi-clouds. p. 49–60, 2014.

NUNES, D. J. **Educação Superior em Computação – Estatísticas 2019**. Porto Alegre, RS, 2019. 67 p.

OBJECT Management Group. OMG. Disponível em: Disponível em: <https://www.omg.org/>. Acesso em: 26 de abr de 2022.

OSMANI, A. **Learning JavaScript Design Patterns**. Sebastopol: O'Reilly Media, Inc., 2012.

PARSIFAL. **Parsifal**. Disponível em: <https://parsif.al/>. Acesso em: 26 de abril de 2022 .

PFLEEGER, S. L. **Engenharia de software : teoria e prática**. 2. ed. São Paulo: Prentice Hall, 2004.

PIEDADE, B.; DIAS, J. a. P.; CORREIA, F. F. An empirical study on visual programming docker compose configurations. 2020.

RANI, F. *et al.* Automated migration of eugenia graphical editors to the web. Association for Computing Machinery, 2020.

RELIABLE Capacity Provisioning and Enhanced Remediation for Distributed Cloud Applications. CORDIS. Disponível em: Disponível em: <https://cordis.europa.eu/project/id/732667>. Acesso em: 26 de abr de 2022.

SANDOBALIN, J.; INSFRAN, E.; ABRAHAO, S. An infrastructure modelling tool for cloud provisioning. p. 354–361, 2017.

SANDOBALIN, J.; INSFRAN, E.; aO, S. A. Argon: A model-driven infrastructure provisioning tool. IEEE Press, p. 738–742, 2019.

SANDOBALíN, J.; INSFRAN, E.; ABRAHãO, S. End-to-end automation in cloud infrastructure provisioning. 2017.

SANDOBALíN, J.; INSFRAN, E.; ABRAHãO, S. On the effectiveness of tools to support infrastructure as code: Model-driven versus code-centric. **IEEE Access**, v. 8, p. 17734–17761, 2020.

SCHAUSS, S. *et al.* A chrestomathy of dsl implementations. Association for Computing Machinery, p. 103–114, 2017.

SEBESTA, R. **Conceitos de Linguagens de Programação**. 11. ed. Porto Alegre: Bookman, 2018.

SILVA, E. A. N. da; FORTES, R. P. M.; LUCRÉDIO, D. A model-driven approach for promoting cloud paas portability. p. 92–105, 2013.

TERRAFORM. HashiCorp. Disponível em: Disponível em: <https://www.terraform.io/>. Acesso em: 26 de abr de 2022.

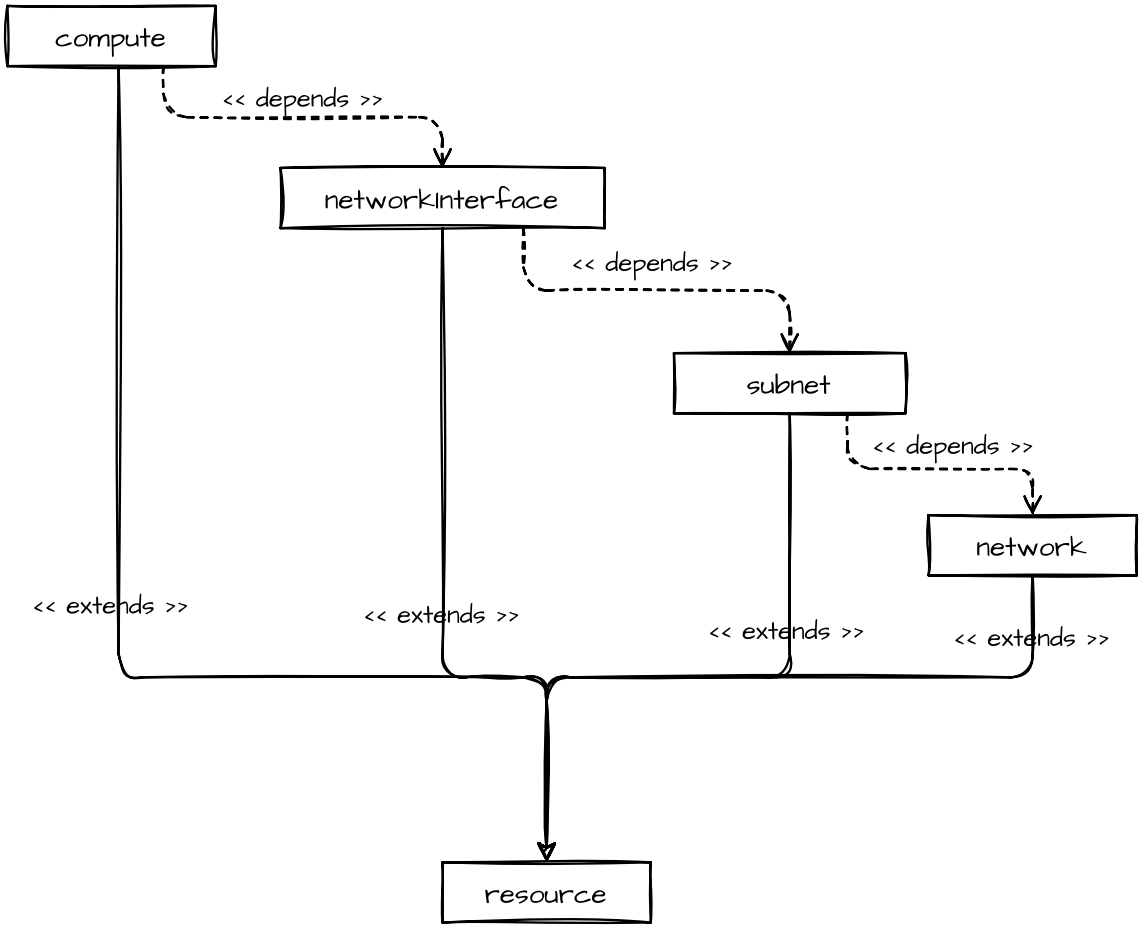
WEERASIRI DENISAND BARUKH, M. C. B. B. J. C. Cloudmap: A visual notation for representing and managing cloud resources. Springer International Publishing, p. 427–443, 2016.

WINKLER, S. **Terraform in action**. Shelter Island: Manning Publications, 2021.

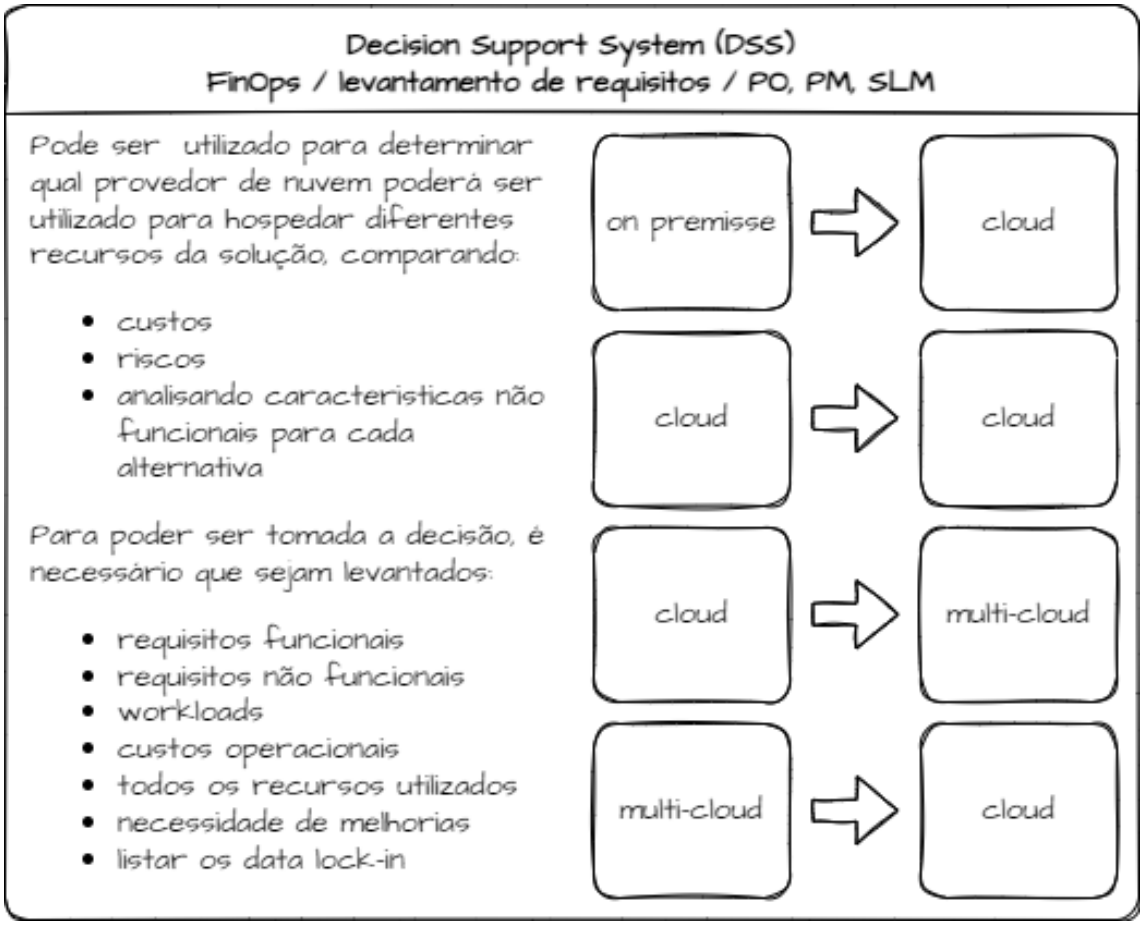
YOU, E. **Vue**. Disponível em: Disponível em: <https://vuejs.org/>. Acesso em: 26 de abr de 2022.

ZHANG, K.; ZHANG, K. B. Graph grammars for visual programming. In: ____. **Software Visualization**. Boston, MA: Springer, 2003. (The Springer International Series in Engineering and Computer Science, v. 734).

# APÊNDICE A – DEPENDÊNCIAS ENTRE OS RECURSOS

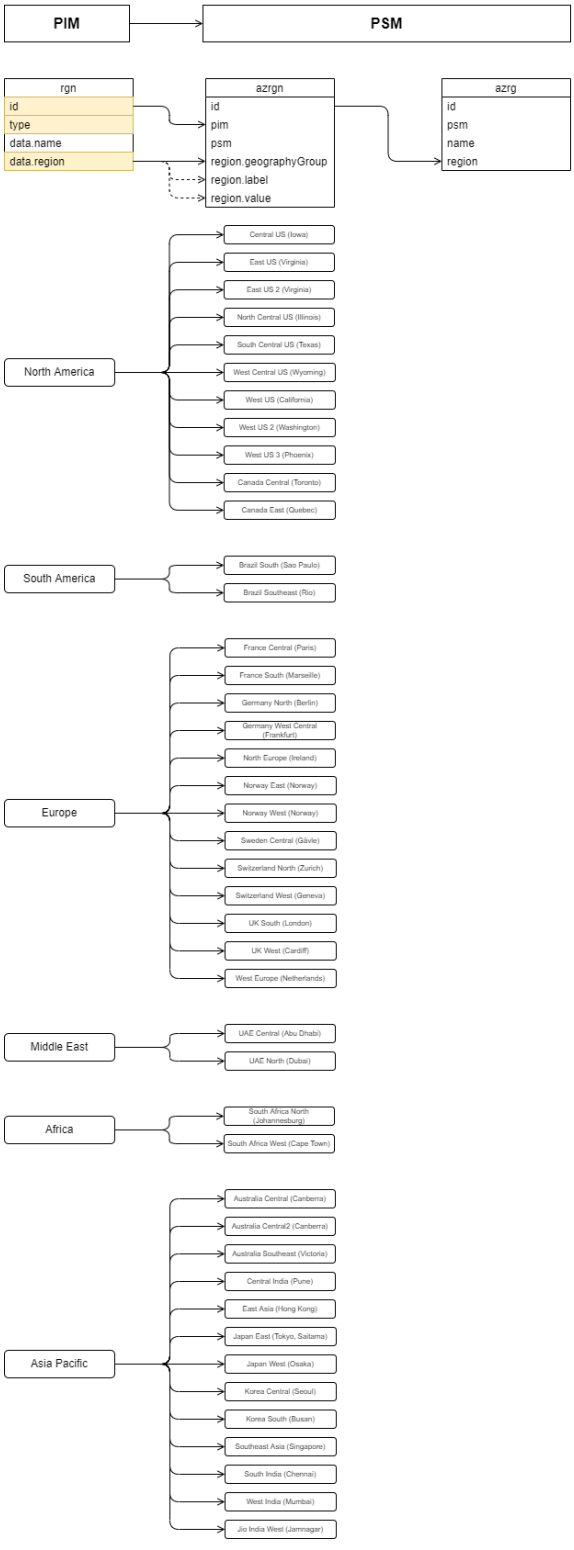


# APÊNDICE B – SISTEMA DE SUPORTE A DECISÃO

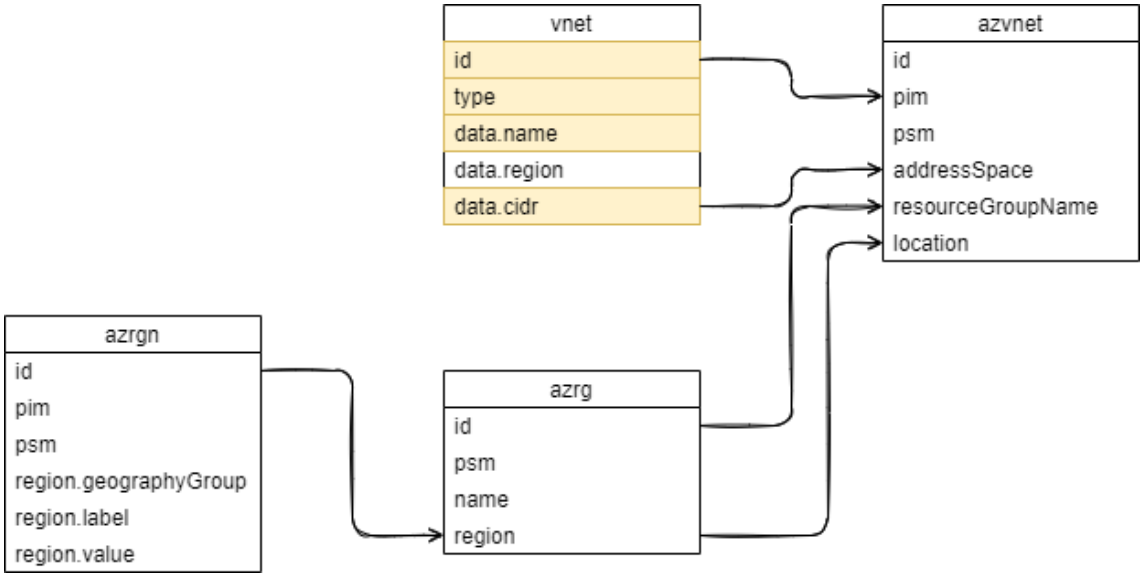


# APÊNDICE C – TRANSFORMADOR DE PIM PARA PSM - REGIÃO

| Geographical | | | | |
|---|---|---|---|---|
| **Resource PIM** | **ID PIM** | **Region PIM** | **PSM** | **Region PSM** |
| {{rgn.data.name}} | {{rgn.id}} | {{rgn.data.region}} | {{azrgn.psm}} | {{azrgn.region}} |



# APÊNDICE D – TRANSFORMADOR DE PIM PARA PSM - REDE



# APÊNDICE E – ÍCONES E CÓDIGO TERRAFORM EQUIVALENTE

| ícones PIM | ícones PSM | estrutura de código terraform esperada |
|---|---|---|
| region |  | |
| |  | resource "azurerm_resource_group" "example" {<br>name = "example"<br>location = "West Europe"<br>} |
| virtual network |  | resource "azurerm_virtual_network" "main" {<br>name = "${var.prefix}-network"<br>address_space = ["10.0.0.0/16"]<br>location = azurerm_resource_group.example.location<br>resource_group_name = azurerm_resource_group.example.name<br>} |
| subnet |  | resource "azurerm_subnet" "internal" {<br>name = "internal"<br>resource_group_name = azurerm_resource_group.example.name<br>virtual_network_name = azurerm_virtual_network.example.name<br>address_prefixes = ["10.0.2.0/24"]<br>} |
| network adapter private (NIC) |  | resource "azurerm_network_interface" "main" {<br>name = "${var.prefix}-nic"<br>location = azurerm_resource_group.example.location<br>resource_group_name = azurerm_resource_group.example.name<br><br>ip_configuration {<br>name = "testconfiguration1"<br>subnet_id = azurerm_subnet.internal.id<br>private_ip_address_allocation = "Dynamic"<br>}<br>} |
| public ip (PIP) |  | resource "azurerm_public_ip" "example" {<br>name = "acceptanceTestPublicIp1"<br>resource_group_name = azurerm_resource_group.example.name<br>location = azurerm_resource_group.example.location<br>allocation_method = "Static"<br><br>tags = {<br>environment = "Production"<br>}<br>} |
| virtual machine |  | resource "azurerm_virtual_machine" "main" {<br>name = "${var.prefix}-vm"<br>location = azurerm_resource_group.example.location<br>resource_group_name = azurerm_resource_group.example.name<br>network_interface_ids = [azurerm_network_interface.main.id]<br>vm_size = "Standard_DS1_v2"<br><br>storage_image_reference {<br>publisher = "Canonical"<br>offer = "UbuntuServer"<br>sku = "16.04-LTS"<br>version = "latest"<br>}<br>storage_os_disk {<br>name = "myosdisk1"<br>caching = "ReadWrite"<br>create_option = "FromImage"<br>managed_disk_type = "Standard_LRS"<br>}<br>os_profile {<br>computer_name = "hostname"<br>admin_username = "testadmin"<br>admin_password = "Password1234!"<br>}<br>os_profile_linux_config {<br>disable_password_authentication = false<br>}<br>tags = {<br>environment = "staging"<br>}<br>} |
| disk |  | resource "azurerm_disk_pool" "example" {<br>name = "example-disk-pool"<br>resource_group_name = azurerm_resource_group.example.name<br>location = azurerm_resource_group.example.location<br>sku_name = "Basic_B1"<br>subnet_id = azurerm_subnet.example.id<br>zones = ["1"]<br>} |