Desenvolvimento e Implementação de uma Linguagem para Modelagem BPMN via API REST

Matheus Filipe Nascimento de Freitas, Herysson Rodrigues Figueiredo
Curso de Sistemas de Informação
UFN - Universidade Franciscana
Santa Maria - RS
m.filipe@ufn.edu.br, herysson.figueiredo@ufn.edu.br

***Resumo*—Este trabalho apresenta o desenvolvimento e a implementação de uma linguagem textual interpretada, de domínio específico, voltada à modelagem de processos de negócio utilizando a notação BPMN (*Business Process Model and Notation*). A aplicação foi estruturada em uma arquitetura composta por três componentes principais: um interpretador de linguagem, *API REST* e uma interface web. Essa integração possibilita que o usuário escreva *scripts* textuais em uma linguagem simples e intuitiva e receba, como resultado, diagramas BPMN estruturados em formato *XML*, compatíveis com o padrão BPMN 2.0. Os resultados obtidos demonstram que a aplicação é funcional e gera diagramas a partir do código textual.**

***Palavras-chave*: BPMN; Modelagem de Processos; Linguagem Específica de Domínio; API REST; Aplicação Web**

I. Introdução

A modelagem de processos de negócio é uma prática fundamental no ambiente corporativo, pois permite compreender, documentar, otimizar e automatizar os fluxos operacionais que compõem a estrutura funcional das organizações. Nesse cenário, a notação BPMN (*Business Process Model and Notation*) consolidou-se como um padrão internacional para a representação visual de processos, proporcionando clareza, padronização e comunicação eficaz entre os diversos agentes envolvidos na gestão empresarial [1]. No entanto, as ferramentas gráficas amplamente utilizadas para a modelagem com BPMN frequentemente apresentam barreiras como custo elevado, baixa flexibilidade e uma curva de aprendizagem acentuada, o que dificulta sua adoção por pequenas empresas, desenvolvedores independentes e equipes com recursos limitados [2].

As soluções disponíveis no mercado atualmente, em sua grande maioria, são proprietárias, exigem licenciamento pago e apresentam limitações quanto à customização e integração com outros sistemas. Além disso, o foco excessivo na interface gráfica pode desviar a atenção do entendimento semântico dos processos, dificultando a automação.

Nesse contexto, foi desenvolvida uma linguagem interpretada específica para modelagem BPMN, aliada a uma *API REST*, como uma solução inovadora e estratégica para superar as limitações identificadas nas ferramentas tradicionais. Essa abordagem proporciona a democratização do acesso à modelagem de processos, oferecendo uma solução leve, flexível e de fácil integração com sistemas web.

A. Objetivos

1) Objetivo Geral: Este trabalho teve como objetivo desenvolver uma linguagem de programação interpretada. Ela é acessível por meio de uma API REST. Sua função é gerar diagramas BPMN no formato XML. Também foi criada uma interface que permite escrever os códigos. Por fim, essa interface possibilita visualizar os diagramas gerados.

2) Objetivos Específicos:

- Estudar e analisar os conceitos fundamentais de BPMN e linguagens específicas de domínio aplicadas à modelagem de processos de negócio.
- Projetar a estrutura da linguagem, definindo sua sintaxe.
- Desenvolver uma API utilizando *Spring Boot* para interpretar a linguagem criada e gerar diagramas BPMN em formato XML.
- Desenvolver uma interface web para facilitar a escrita, manipulação e validação dos *scripts*.
- Implementar documentação de integração da *API*, permitindo sua incorporação aos fluxos empresariais.
- Garantir a validação e exportação dos modelos gerados, assegurando conformidade com o padrão BPMN.

Com o objetivo de fundamentar teoricamente o desenvolvimento do sistema, o Referencial Teórico aborda os principais conceitos e tecnologias relacionados ao tema.

Na seção de Trabalhos Correlatos, são apresentados os estudos e artigos que serviram de base para a pesquisa e implementação. A seguir, a seção de Metodologia descreve as etapas e abordagens utilizadas na implementação do projeto. Posteriormente, a seção de resultados detalha o funcionamento da aplicação assim como a ilustração de exemplos de diagramas e scripts criados utilizando a aplicação.

Por fim, a seção de conclusão retoma os objetivos do trabalho, discute os resultados alcançados e indica possíveis caminhos para aprimoramentos futuros.

II. Referencial teórico

Nesta seção, apresentam-se os conceitos-chave e as inovações tecnológicas que fundamentam o desenvolvimento deste trabalho, enfatizando o papel da modelagem de processos e suas aplicações para melhorar a eficiência na criação de modelos de negócio. Também foram pesquisadas soluções para gerenciamento de sintaxe e tecnologias para controle e programação, tornando o sistema mais robusto.

A. Business Process Management (BPM)

A modelagem de processos é uma técnica utilizada para representar visualmente o fluxo de atividades de um processo de negócio. Ela permite analisar, otimizar e automatizar processos, garantindo assim, que haja uma maior eficiência e um melhor alinhamento com os objetivos organizacionais [3].

Business Process Management (BPM) é uma abordagem sistemática que visa melhorar os processos empresariais através do gerenciamento e automatização desses processos. Ele combina metodologias, tecnologias e boas práticas para garantir que os processos tenham maior eficiência, sejam processos mais flexíveis e que tenham alinhamento com os objetivos estratégicos das empresas [4].

BPM busca a otimização constante dos processos, integrando ferramentas tecnológicas para aumentar a eficiência, permitindo ajustes dinâmicos conforme mudanças nos negócios, utilizando indicadores para monitorar e avaliar processos e priorizando a gestão baseada em fluxos de atividades e não em estruturas funcionais tradicionais [3].

Atualmente, o BPM evoluiu com a incorporação de tecnologias como Inteligência Artificial, *Robotic Process Automation* (RPA) e análise de dados. As soluções modernas de BPM possibilitam maior agilidade empresarial e estão alinhadas com iniciativas de transformação digital e flexíveis a mudanças de mercado [4].

B. Business Process Model and Notation (BPMN)

A *Business Process Model and Notation* (BPMN) é uma linguagem padronizada voltada à modelagem de processos de negócio, permitindo representar de forma clara e compreensível o fluxo de atividades, facilitando a comunicação entre áreas e a automação de processos [1, 5].

A Figura 1 ilustra um exemplo de diagrama BPMN, retirado do livro de Silver [6], composto por quatro elementos principais: eventos, atividades, gateways e eventos finais. O evento inicial (círculo) marca o início do processo; as atividades (retângulos com cantos arredondados) representam tarefas manuais, automáticas ou de usuário; os gateways (losangos) controlam o fluxo, permitindo decisões exclusivas ou paralelas; e o evento final (círculo com borda grossa) indica o encerramento do processo.

O processo é estruturado em três piscinas (*pools*) horizontais *Sales*, *Finance* e *Warehouse* cada uma com responsabilidades específicas. O fluxo inicia em *Sales* com o recebimento do pedido (*Receive Order*), segue para *Finance* para verificação de crédito (*Check Credit*) e, caso aprovado, continua para *Warehouse* com a tarefa de atendimento (*Fulfill Order*). Após confirmar o estoque, o processo retorna a *Finance* para emissão da fatura (*Send Invoice*) e é finalizado com o evento *Order Complete*.

Esse exemplo demonstra o potencial da BPMN como ferramenta de modelagem de processos, promovendo clareza, padronização e integração entre setores.

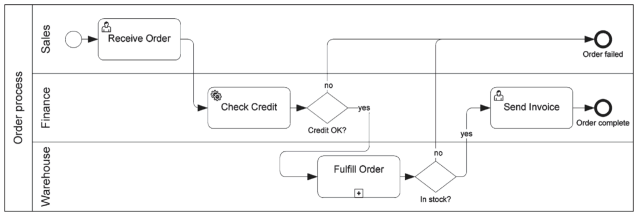

Figura 1. Adaptação do diagrama BPMN [6].

C. Linguagem de Programação

Segundo Sebesta, a linguagem de programação é um sistema formal composto por regras sintáticas e semânticas, utilizado para gerar instruções que podem ser interpretadas e executadas por máquinas computacionais [7]. Dessa forma, viabiliza-se a comunicação entre seres humanos e computadores, permitindo a construção de algoritmos e sistemas capazes de abstrair e automatizar atividades complexas e repetitivas [8].

A linguagem de programação funciona como uma interface entre a lógica de solução de um problema e sua implementação em uma máquina computacional. Por sua vez, Van Roy e Haridi [9] destacam que a linguagem de programação não é apenas uma ferramenta técnica, mas também uma forma de pensamento computacional estruturado [7].

Nesse contexto, o exemplo apresentado no listing 1 ilustra a estrutura de uma linguagem orientada a objetos como o *Java*. A classe denominada `ContaBancaria`, que ilustra os princípios fundamentais da Programação Orientada a Objetos (POO), tais como encapsulamento, construtores e métodos de acesso. A classe possui dois atributos privados: `titular`, que representa o nome do proprietário da conta, e `saldo`, que armazena o valor disponível.

O construtor permite inicializar uma nova conta com nome e saldo inicial positivo. Os métodos públicos `depositar` e `sacar` possibilitam modificar o saldo, respeitando regras básicas de validação, como valores positivos e saldo suficiente para saque. Os métodos `getSaldo` e `getTitular` permitem acessar as informações da conta de forma controlada. Essa estrutura promove a segurança dos dados e a modularidade do código, características essenciais na construção de sistemas robustos.

```java
public class ContaBancaria {
    private String titular;
    private double saldo;

    public ContaBancaria(String titular,
        double saldoInicial) {
        this.titular = titular;
        if (saldoInicial > 0) {
            this.saldo = saldoInicial;
        }
    }

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        }
    }

    public void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
        }
    }

    public double getSaldo() {
        return saldo;
    }

    public String getTitular() {
        return titular;
    }
}
```

Listing 1. Classe ContaBancaria em *Java* baseada em Deitel e Deitel [8]

Além dos aspectos de POO analisados no listing 1, é possível analisar a estrutura léxica da linguagem de programação utilizada.

D. Estrutura léxica

A estrutura léxica diz respeito ao conjunto de regras que definem como os caracteres são agrupados em unidades significativas chamadas *tokens*, como identificadores, palavras-chave, operadores e delimitadores. No exemplo da classe `ContaBancaria`, observam-se diversos elementos léxicos: `public`, `class`, `private`, `double`, entre outros, são palavras-chave da linguagem Java; `titular`, `saldo` e `valor` são identificadores definidos pelo programador; os símbolos `=`, `+`, `-` são operadores; os caracteres `{`, `}`, `(`, `)` atuam como delimitadores.

Esses componentes são fundamentais para a análise léxica, uma das primeiras etapas do processo de interpretação ou compilação da linguagem, responsável por transformar o código fonte em uma sequência de *tokens* que serão, em seguida, analisados e executados [10]. Dessa forma, a estrutura léxica assegura que o código seja interpretado de maneira consistente, promovendo clareza e evitando ambiguidade na interpretação do programa.

A forma como esses *tokens* são processados e executados depende do paradigma de execução adotado pela linguagem, o que leva à distinção entre linguagens compiladas e interpretadas.

Nas linguagens compiladas, como C e C++, o código-fonte é traduzido por um compilador para código de máquina antes da execução, gerando um arquivo executável. Esse processo costuma resultar em melhor desempenho, já que o programa não precisa ser traduzido em tempo de execução. Por outro lado, nas linguagens interpretadas, como *Python* e *JavaScript*, o código é lido e executado linha por linha por um interpretador, o que facilita testes e depuração, mas geralmente com custo de desempenho. Algumas linguagens modernas, como *Java*, utilizam uma abordagem híbrida: o código é compilado para *bytecode* (código intermediário gerado por compiladores), que é então interpretado e executado por uma máquina virtual [7].

Atualmente, linguagens como *Python*, *Java*, *JavaScript*, *CSharp* e *Go* estão entre as linguagens mais utilizadas no desenvolvimento de sistemas em diversos contextos, incluindo aplicações corporativas e plataformas web. A escolha por essas linguagens está relacionada à sua versatilidade, robustez e amplo suporte por comunidades e ferramentas. Indicadores como o *TIOBE Index* [11] e a *Stack Overflow Developer Survey* [12] disponibilizam *rankings* periódicos que refletem a popularidade e o uso dessas linguagens em escala global, evidenciando tendências tecnológicas e preferências da indústria.

E. REST

Os serviços REST (*Representational State Transfer*) são amplamente utilizados para construir APIs (*Application Programming Interfaces*) que permitem a comunicação entre diferentes aplicações de forma simples, escalável e seguindo o princípio de *statelessness*, no qual o servidor não mantém informações sobre o estado da comunicação entre requisições [13]. As operações principais são realizadas por meio do protocolo de comunicação HTTP (*HyperText Transfer Protocol*), amplamente adotado no padrão REST para transmissão de dados [14].

O conceito de REST foi introduzido por Roy Fielding em sua tese de doutorado intitulada *Architectural Styles and the Design of Network-based Software Architectures* na Universidade da Califórnia, Irvine, em 2000. Fielding foi um dos principais autores da especificação do protocolo HTTP e propôs REST como uma forma de padronizar a comunicação entre sistemas na web de forma simples, leve e eficiente [15].

Com a publicação da tese de Fielding em 2000, REST apresentou uma abordagem inovadora que priorizava a simplicidade, a utilização dos padrões já existentes da web, e a separação entre cliente e servidor [15].

F. Ferramentas

1) Angular: Plataforma de desenvolvimento para aplicações web desenvolvida pelo Google. O Angular permite

a criação de interfaces dinâmicas e responsivas... Através do Angular, torna-se mais simples consumir *APIs REST*, integrando de forma eficiente o cliente e o servidor em aplicações web modernas [16].

2) ANTLR v4: também conhecida como *Another Tool for Language Recognition*, é uma poderosa ferramenta utilizada para gerar analisadores léxicos e sintáticos a partir de gramáticas formais.

O *ANTLR v4* permite a definição de regras gramaticais em uma linguagem própria, que são então utilizadas para criar interpretadores, compiladores ou tradutores de linguagens. Com ele, é possível desenvolver novas linguagens interpretadas de forma estruturada e eficiente, garantindo maior controle sobre a sintaxe e semântica do código-fonte. Essa abordagem é especialmente útil em projetos acadêmicos ou profissionais que demandam a criação de DSLs [17].

3) Trello: ferramenta baseada na web para gerenciamento de projetos, que utiliza o conceito de quadros (boards), listas e cartões (*cards*) para organizar tarefas e fluxos de trabalho de forma visual e intuitiva. Amplamente utilizado em equipes ágeis, o *Trello* permite a colaboração em tempo real, a definição de prazos, a atribuição de responsáveis e a integração com outras ferramentas. Sua simplicidade e flexibilidade o tornam útil tanto para projetos de software quanto para planejamento pessoal ou acadêmico [18].

4) BPMN JS: biblioteca *open source* baseada em *JavaScript*, que permite a visualização, edição e manipulação de diagramas BPMN diretamente em aplicações web. A biblioteca oferece um conjunto robusto de módulos reutilizáveis que possibilitam aos desenvolvedores incorporar facilmente editores BPMN personalizados, renderizar diagramas existentes ou até mesmo estender funcionalidades conforme necessidades específicas. Com suporte à notação BPMN 2.0, o BPMN JS facilita a integração da modelagem de processos em soluções web modernas, promovendo flexibilidade, interatividade e aderência aos padrões de mercado [19].

5) Camunda Library: plataforma de automação de processos baseada na notação BPMN 2.0, amplamente utilizada para modelar, executar e monitorar fluxos de trabalho e decisões de negócio. A biblioteca do Camunda permite a integração direta com motores de execução de processos, possibilitando que diagramas BPMN criados programaticamente sejam interpretados e executados de forma automatizada além de facilitar a compilação de diagramas escritos programaticamente para o formato XML [20].

III. Trabalhos correlatos

Nesta seção, são apresentados os artigos relacionados ao uso e desenvolvimento de *APIs REST* e suas tecnologias associadas, que contribuíram para o desenvolvimento deste projeto. Os artigos apresentados nesta seção trazem informações sobre a dificuldade na utilização de ferramentas gráficas para a montagem de modelos de processos assim como, a viabilidade da criação de uma nova linguagem de programação.

A. Cesno: Possibility of Creating a New Programming Language

O trabalho de Ozelot Vanilla e colaboradores [21], discute a viabilidade da criação de uma nova linguagem de programação, considerando aspectos técnicos, sintáticos e semânticos que envolvem esse processo. Os autores analisam os fundamentos necessários para o desenvolvimento de uma linguagem do zero, abordando tópicos como design lexical, definição de gramática, construção de interpretadores e a importância da documentação durante o ciclo de desenvolvimento. O estudo se destaca por apresentar uma reflexão teórica e prática sobre as motivações e desafios associados à criação de linguagens personalizadas, visando atender nichos ou demandas específicas.

Os resultados da pesquisa demonstram que é possível criar uma linguagem funcional e adaptada a um domínio específico, desde que haja clareza nos requisitos e uma estrutura modular bem definida. O trabalho também evidencia que a personalização de linguagens pode melhorar a expressividade e facilitar a comunicação entre usuários técnicos e não técnicos em contextos restritos.

O trabalho forneceu uma base teórica importante para estruturar a linguagem interpretada deste projeto, especialmente na criação de critérios para definição de tokens e regras gramaticais. Essas discussões foram fundamentais para orientar a construção da sintaxe proposta. Além disso, o artigo destaca como decisões de design linguístico podem atender objetivos práticos, reforçando a eficácia de linguagens específicas na solução de problemas pontuais em desenvolvimento de software.

B. Desenvolvimento de uma API REST utilizando os princípios SOLID para proporcionar o acesso a dados públicos para aplicações no Brasil

O trabalho de Rafael Londero Cancian e Ana Paula Canal [22] aborda o desenvolvimento de uma *API RESTful* com foco na padronização do acesso a dados públicos em aplicações brasileiras. A proposta do artigo consiste na criação de uma interface de programação que, utilizando a arquitetura REST e boas práticas de desenvolvimento, disponibiliza endpoints claros e bem estruturados para consulta de informações como CEP, DDD e CNPJ.

Os resultados obtidos com a implementação da API indicam que o uso dos princípios SOLID (são cinco princípios da POO) promoveu maior organização e facilidade de manutenção no código. Além disso, a padronização das respostas e rotas REST contribuiu para uma melhor experiência de integração por parte dos desenvolvedores, demonstrando a viabilidade de soluções nacionais robustas para o acesso a dados públicos.

Durante o processo de elaboração deste Trabalho de Conclusão de Curso, o artigo foi essencial para a compreensão dos fundamentos que norteiam a arquitetura REST, especialmente no que se refere à separação de responsabilidades, ao uso adequado dos métodos HTTP e à padronização de rotas e respostas. Além disso, o estudo contribuiu significativamente para a escolha da metodologia ágil FDD (*Feature-Driven Development*), uma vez que reforça a importância da organização e da clareza no desenvolvimento de funcionalidades centradas em requisitos específicos.

C. A Complementary Analysis of BPMN 2.0-Based Tools Behavior Regarding Process Modeling Problems

O trabalho de João Vitor de Camargo [23] apresenta uma análise complementar do comportamento das ferramentas baseadas em *BPMN* 2.0 no que se refere a problemas de modelagem de processos. O estudo avalia como essas ferramentas tratam anti-padrões de modelagem de processos e as diferenças entre ferramentas pagas e gratuitas em termos de *feedback* fornecido sobre problemas nos modelos. Essa análise é essencial para compreender as limitações e os desafios encontrados ao se utilizar ferramentas de modelagem de processos em ambientes corporativos, especialmente quando essas ferramentas precisam lidar com aspectos complexos da notação *BPMN*.

Como resultado, o estudo revelou que a maioria das ferramentas analisadas não oferece suporte adequado para a detecção de falhas conceituais nos modelos, o que reforça a importância de abordagens mais automatizadas e customizáveis como a linguagem proposta neste trabalho que permitam validar e interpretar processos com maior controle e precisão.

A pesquisa evidenciou as principais dificuldades no uso de ferramentas gráficas para modelagem *BPMN*, como baixa flexibilidade, limitações de personalização e desafios de integração. Também destacou a falta de opções gratuitas eficientes. Esses pontos ajudam a orientar o desenvolvimento da *API*, reforçando a necessidade de uma linguagem mais adaptável e independente de ferramentas gráficas complexas.

Os trabalhos correlatos apresentados fornecem uma base sólida para este projeto, orientando a escolha de metodologias e práticas de desenvolvimento.

IV. METODOLOGIA

Esta seção descreve a metodologia adotada para o desenvolvimento da linguagem proposta. O trabalho foi conduzido com base na abordagem ágil Feature-Driven Development (FDD), reconhecida por sua orientação a funcionalidades e pela entrega contínua e incremental de software. O processo foi dividido em cinco etapas principais: desenvolvimento do modelo geral, construção da lista de funcionalidades, planejamento de funcionalidades, projeção por funcionalidades e construção por funcionalidade [24]. Essa estrutura possibilitou um ciclo iterativo de desenvolvimento, promovendo a evolução constante do sistema.

A. Concepção do projeto

Neste trabalho, desenvolveu-se uma linguagem de programação interpretada voltada especificamente para a geração de diagramas BPMN. Ao enviar um *script* seguindo as regras da linguagem proposta para uma API REST, o usuário recebe como resultado um diagrama BPMN estruturado em formato XML.

Conforme ilustrado na Figura 2, a API, representada pelo elemento Servidor, recebe *scripts* escritos na linguagem DSL proposta por meio de requisições HTTP indicadas pelos elementos de Comunicação. Esses *scripts* podem ser enviados tanto pela interface gráfica da aplicação quanto por sistemas externos integrados, representados pelo elemento API do usuário.

Ao receber uma requisição, a API executa uma validação estrutural do script, verificando se os elementos declarados seguem as regras definidas pela linguagem e se o fluxo do processo apresenta coerência lógica. Em seguida, o script é interpretado internamente e convertido em um documento BPMN no formato XML.

O XML gerado é retornado como resposta pela API e pode ser consumido de duas formas: pela API do usuário, ou pela interface gráfica, que utiliza a biblioteca BPMNJS para renderizar visualmente o diagrama BPMN com base na estrutura definida no XML.

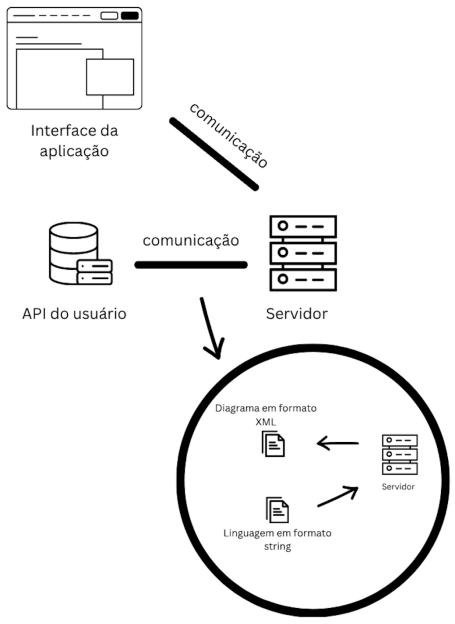

Figura 2. Representação do fluxo de comunicação da aplicação.

B. Levantamento e Definição de Requisitos

Durante o processo de levantamento e definição de requisitos, optou-se por utilizar a ferramenta Trello para organizar as atividades. A estruturação do trabalho foi distribuída em quatro etapas principais elicitação, análise, especificação e validação.

1) Elicitação: Na fase de elicitação, buscou-se identificar as potenciais funcionalidades da linguagem interpretada, que facilitassem a criação de diagramas BPMN em contextos acadêmicos e corporativos, considerando a substituição ou complementação de ferramentas visuais tradicionais.

2) Análise: Os requisitos coletados foram analisados quanto à viabilidade técnica, relevância e aderência ao escopo do projeto. Essa etapa permitiu priorizar as funcionalidades mais pertinentes, eliminando redundâncias e conflitos, e garantindo o foco nas características essenciais ao sistema.

3) Especificação: Durante a especificação, os requisitos foram organizados de forma estruturada e clara, sendo classificados em funcionais e não funcionais. Cada requisito recebeu critérios de aceitação bem definidos. Alguns foram detalhados para garantir sua correta implementação por exemplo, a funcionalidade de exportação de modelos no formato XML BPMN 2.0.

4) Validação: A etapa de validação teve como objetivo assegurar que os requisitos fossem consistentes e compreensíveis. A validação foi realizada através análises de escopo e funcionalidades. Os requisitos validados serviram de base para as etapas subsequentes de desenvolvimento e testes.

Tabela I
TABELA DE REQUISITOS FUNCIONAIS

Requisito	Descrição
RF01	O sistema deverá interpretar os scripts recebidos e gerar XML que através de uma biblioteca será gerado o diagrama.
RF02	A ferramenta deve suportar a criação de elementos de evento *BPMN*, incluindo eventos de início e fim, conforme especificações da notação *BPMN* 2.0.
RF03	O sistema deverá oferecer suporte à criação de gateways do tipo exclusivo e paralelo, respeitando a sintaxe definida.
RF04	O sistema deverá oferecer suporte à criação de tarefas do tipo manual, automatizada e tarefas de usuário, respeitando a sintaxe definida.
RF05	A sequência de fluxo entre os elementos do processo deve ser estabelecida com base na estrutura textual interpretada.
RF06	Em caso de falhas sintáticas ou semânticas, a aplicação deverá retornar mensagens de erro claras e objetivas via resposta HTTP.
RF07	O usuário poderá incluir comentários no código textual com fins documentais, sem que isso afete a interpretação lógica do script.
RF08	O sistema deve realizar a validação do modelo quanto à integridade lógica de retornar o XML.
RF09	O sistema deverá oferecer uma interface gráfica, por meio da qual seja possível criar e visualizar diagramas, por meio da linguagem criada.

Tabela II
TABELA DE REQUISITOS NÃO FUNCIONAIS

Requisito	Descrição
RNF01	A linguagem textual de entrada deve apresentar uma sintaxe simples e intuitiva, favorecendo o aprendizado e a adoção por usuários com diferentes níveis de conhecimento técnico.
RNF02	Toda a documentação referente à linguagem e à *API* deve ser pública e conter exemplos práticos de uso, facilitando a integração e compreensão por parte dos desenvolvedores.
RNF03	A **API** deve seguir boas práticas de segurança, incluindo validação das entradas do usuário.
RNF04	O projeto deverá ser licenciado como software livre, preferencialmente sob licenças reconhecidas como **MIT**, permitindo seu uso e modificação por terceiros.

C. Projeto

Na fase inicial de projeto, foi elaborado um diagrama de classes representando os principais componentes da aplicação: `StartEvent`, `EndEvent`, `Gateway`, `Task`, `Pool` e `Process`, que representam as entidades centrais do sistema, conforme demonstrado na Figura 3.

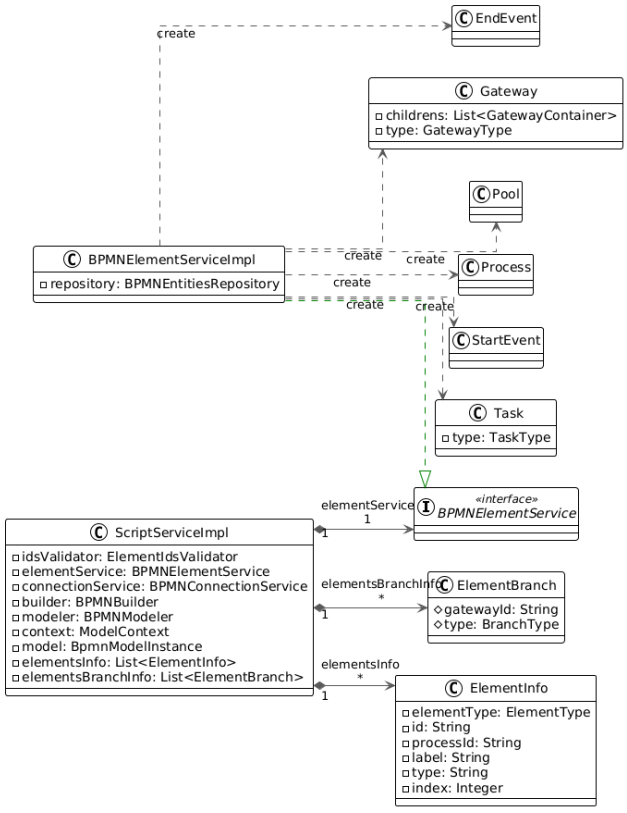

Figura 3. Diagrama de classe da Aplicação

A partir dessa modelagem estrutural, o serviço responsável pela interpretação do *script*, representado pela classe `ScriptServiceImpl`, realiza a identificação dos elementos que compõem o fluxo e os organiza em estrutu-

ras auxiliares, como `ElementInfo` e `ElementBranch`. Nessa fase, são executadas validações fundamentais como a verificação de identificadores duplicados e a confirmação de que todos os *ids* correspondem a entidades existentes no *script* garantindo assim a integridade e a consistência dos dados.

Na sequência, o `ScriptServiceImpl` delega às classes `BPMNElementServiceImpl` e `BPMNConnectionServiceImpl` a responsabilidade de converter essas estruturas auxiliares em entidades BPMN concretas, que são persistidas em memória para posterior associação. Em seguida, o serviço define as relações lógicas entre os elementos criados, estabelecendo sequências, dependências e condições de fluxo que determinam o comportamento do processo durante sua execução.

Com isso, é construída uma instância de `BpmnModelInstance`, classe disponibilizada pela biblioteca Camunda e encarregada de gerenciar todos os componentes internos de um modelo BPMN. As entidades previamente armazenadas em memória são então incorporadas dentro do modelo, permitindo a definição completa das estruturas que compõem o diagrama. Após concluída a modelagem lógica, o sistema gera os elementos gráficos e determina seus posicionamentos, finalizando a etapa de construção do fluxo a partir do *script* fornecido.

D. Construção

A fase de construção consistiu na implementação prática dos componentes da linguagem proposta e de sua integração com o ambiente gráfico (interface da documentação e editor de código online).

Inicialmente, foram definidos os *tokens* e regras gramaticais da linguagem DSL (*Domain-Specific Language*). Essa definição teve como objetivo estabelecer a estrutura sintática da linguagem, especificando palavras-chave, delimitadores e padrões válidos de escrita para os elementos correspondentes a atividades, eventos, gateways e fluxos de processo. As regras foram formalizadas utilizando a ferramenta ANTLR v4, amplamente utilizada para geração de analisadores léxicos e sintáticos. A partir do arquivo de gramática demonstrado no Listing 2, o ANTLR gerou automaticamente o interpretador léxico-sintático, responsável por identificar e validar os componentes da linguagem durante a execução.

O trecho apresentado no Listing 2 ilustra parte do arquivo léxico da linguagem utilizada pelo interpretador desenvolvido neste trabalho. Esse arquivo, denominado `ProcessLexer.g4`, define a estrutura básica da linguagem, especificando os *tokens* que representam cada elemento sintático reconhecido pelo compilador gerado pelo ANTLR v4.

Entre os elementos definidos, destacam-se os identificadores de componentes da notação BPMN, como `pool`, `process`, `task` e `gateway`, bem como delimitadores e operadores responsáveis pela composição das instruções, como parênteses, chaves e setas de fluxo. Além disso, demonstra as regras criadas para ignorar espaços em branco e comentários (`WS`, `LINE_COMMENT` e `BLOCK_COMMENT`).

```
lexer grammar ProcessLexer;

POOL         : 'pool';
PROCESS      : 'process';
START        : 'start';
TASK         : 'task';
END          : 'end';
GATEWAY      : 'gateway';
SCOPE        : 'scope';
YES          : 'yes';
NO           : 'no';

TaskType: 'MANUAL' | 'AUTOMATED' | 'USER';
GatewayType: 'EXCLUSIVE' | 'PARALLEL';

ID       : [a-zA-Z_][a-zA-Z_0-9]*;
STRING   : '"' (~["\\] | '\\' .)* '"' ;
LPAREN   : '(';
RPAREN   : ')';
LBRACE   : '{';
RBRACE   : '}';
COMMA    : ',';
SEMICOLON: ';';
ARROW    : '->';

WS       : [ \t\r\n]+ -> skip ;
LINE_COMMENT : '//' ~[\r\n]* -> skip ;
BLOCK_COMMENT : '/*' .*? '*/' -> skip ;
```

Listing 2. Arquivo de configuração de estrutura de linguagem

Em seguida, foi iniciado o desenvoldimento da API REST utilizando o *framework Spring Boot* da linguagem *Java*. Inicialmente, propôs-se o desenvolvimento de uma *API RESTful* contendo um interpretador para a linguagem de programação proposta, responsável pela geração de diagramas *BPMN* em formato *XML*.

Durante o processo de desenvolvimento, observou-se que a adoção integral do padrão *REST* se tornava inviável para este contexto, uma vez que a aplicação não faz uso de cache, não mantém entidades persistidas em banco de dados e os *scripts* são processados dinamicamente a cada requisição. Assim, cada diagrama é interpretado e gerado no momento do envio, sem necessidade de armazenamento prévio.

Essa mudança resultou em uma arquitetura mais simples, leve e direta, proporcionando maior agilidade no desenvolvimento e facilitando a documentação da *API*. Além disso, a abordagem adotada simplificou o processo de integração com sistemas externos, uma vez que o retorno da *API* é imediato e baseado apenas no conteúdo do *script* enviado.

O primeiro módulo a ser desenvolvido foi o de validação semântica, cuja função é garantir a coerência lógica entre os elementos do *script*. Nessa etapa, o sistema verifica, por exemplo, se os identificadores de tarefas, eventos e

gateways são únicos, se as conexões respeitam as regras de fluxo estabelecidas pela linguagem e se o processo contém pontos de início e fim devidamente definidos. Essa validação é essencial para evitar a geração de modelos BPMN inconsistentes ou incompletos. Além disso, durante a fase de validação dos *tokens*, são extraídas as entidades identificadas por cada *token* do *script*, através de classes utilizadas pelo ANTLR v4, conforme ilustrado no Listing 3.

```
public void exitTaskRule(ProcessParser.
    TaskRuleContext ctx) {
    String id = ctx.ID().getText();
    String label = stripQuotes(ctx.STRING().
        getText());
    String type = ctx.TaskType().getText();
    String currentProcessId =
        processContextStack.peek();
    Integer line = ctx.ID().getSymbol().
        getLine();
    elements.add(new ElementInfo(id,
        ElementType.TASK, label, type,
        currentProcessId, line));
}
```

Listing 3. Método responsável por extrair entidades do script

Após a validação, desenvolveu-se o módulo de conversão de entidades, responsável por transformar as estruturas extraídas do *script* em objetos manipuláveis pela aplicação. As informações capturadas pelo interpretador são mapeadas para classes de domínio, como `Task`, `Gateway`, `StartEvent`, `EndEvent` e `Process`, representando os elementos fundamentais de um diagrama BPMN. As entidades retiradas do script são mapeadas para entidades do sistema e são armazenados em memória por meio dos repositórios `BPMNEntityRepository` e `BPMNConnectionRepository`, garantindo uma organização eficiente dos dados para as etapas seguintes.

Com as entidades e conexões devidamente estruturadas, foi desenvolvido o modelador BPMN, utilizando as classes fornecidas pela biblioteca Camunda. Esse módulo é responsável por instanciar os elementos BPMN, adicionar atributos específicos (como nomes, identificadores e tipos de evento) e construir o objeto `BpmnModelInstance`, que representa o modelo completo em memória.

A partir dessa estrutura, o sistema gera o documento XML BPMN 2.0, totalmente compatível com motores de execução e ferramentas de modelagem baseadas no padrão BPMN. O Listing 4 mostra os métodos utilizados para realizar a adição de uma entidade `Task` ao modelo.

```
    Task taskEntity = (Task) entity;
    var serviceTask = model.newInstance(org.
        camunda.bpm.model.bpmn.instance.
        ServiceTask.class);
    serviceTask.setId(task.getId());
    serviceTask.setName(task.getLabel());
    org.camunda.bpm.model.bpmn.instance.
        Process process = model.
        getModelElementById(processId);
    process.addChildElement(serviceTask);
}
```

Listing 4. Método responsável por adicionar entidades ao modelo

Na sequência, foi implementado o gerador de elementos gráficos, componente responsável por posicionar os elementos no diagrama e definir suas dimensões e relações visuais. Esse módulo calcula automaticamente as coordenadas dos nós e das arestas, levando em consideração o tipo e o tamanho de cada elemento, de modo a garantir uma disposição legível e organizada no diagrama.

Para a visualização da documentação da aplicação foi desenvolvida uma interface utilizando o *framework Angular*. A interface é composta por um editor de código online e um visualizador de diagramas BPMN, através do uso da biblioteca *BPMNJS Viewer*. Assim, a partir do diagrama em formato XML ou BPMN gerado pela API, o *viewer* é responsável por renderizar graficamente o diagrama, permitindo ajustes de posicionamento dos elementos e conexões criadas.

V. Resultados

O resultado final foi um protótipo funcional, capaz de interpretar *scripts* escritos na linguagem DSL, validar sua estrutura e lógica, e gerar automaticamente um diagrama BPMN em formato XML, pronto para visualização ou integração com outras ferramentas.

A documentação e interação com a *API REST* foram disponibilizadas por meio de uma interface gráfica desenvolvida em *Angular*, que inclui um editor de código online. Essa interface permite ao usuário testar a linguagem, gerar chaves de integração entre sistemas e visualizar o resultado dos *scripts* diretamente na página. A Figura 4 apresenta o componente principal da aplicação, responsável pela edição e visualização do código.

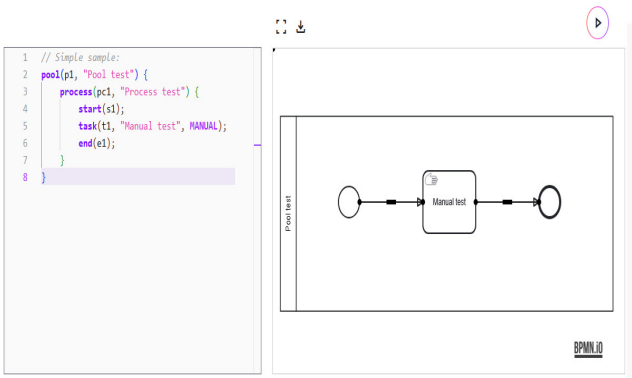

Figura 4. Interface gráfica da aplicação.

A Figura 4 apresenta a interface principal da aplicação, na qual o editor de código é exibido à esquerda e a visualização do diagrama BPMN gerado aparece à direita. O Listing 5 mostra o script utilizado no editor ilustrado na figura, servindo como exemplo básico da linguagem DSL desenvolvida:

```
pool(p1, "Pool test") {
    process(pc1, "Process test") {
        start(s1);
        task(t1, "Manual test", MANUAL);
        end(e1);
    }
}
```

Listing 5. Exemplo de script na linguagem DSL

A partir do *script* mostrado, a *API* processa a requisição, interpreta os elementos declarados e gera um diagrama *BPMN* representado no formato *XML*. O resultado retornado pela *API* é exemplificado no Listing 6.

```
<?xml version="1.0" encoding="UTF-8"
    standalone="no"?>
<definitions>
  <process id="pc1" isExecutable="true" name=
      "">
    <startEvent id="s1"/>
    <manualTask id="t1" name="Manual test"/>
    <endEvent id="e1"/>
    <sequenceFlow id="flow_s1_t1" sourceRef="
        s1" targetRef="t1"/>
    <sequenceFlow id="flow_t1_e1" sourceRef="
        t1" targetRef="e1"/>
  </process>
  <collaboration id="
      collaboration_1762714240784">
    <participant id="p1" name="Pool test"
        processRef="pc1"/>
  </collaboration>
  <!-- graphic elements tags -->
</definitions>
```

Listing 6. Exemplo de diagrama BPMN gerado em formato XML

O resultado gerado pela aplicação é totalmente compatível com o padrão *BPMN 2.0* e pode ser importado em ferramentas externas de modelagem, além de ser renderizado visualmente na própria interface da aplicação, demonstrado na figura 5.

Além disso o usuário pode ajustar o posicionamento dos elementos e fluxos diretamente na interface, tornando o processo de modelagem mais dinâmico e intuitivo.

Por fim, a aplicação conta com uma documentação completa online [1] explicando a estrutura de cada entidade da linguagem, seus tipos e sintaxe, além de exemplos práticos de integração com outras *APIs* ou aplicações externas. Essa documentação tem como objetivo facilitar o uso da linguagem e promover sua aplicação em diferentes contextos acadêmicos e corporativos.

[1] https://bpmn-runner.dev

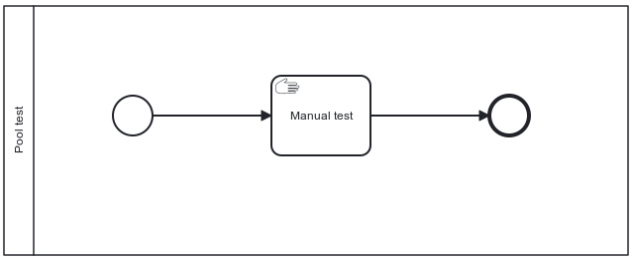

Figura 5. Diagrama resultado do listing 6.

O diagrama apresentado na figura 5 corresponde à representação visual gerada a partir do XML exibido no listing 6, o qual, por sua vez, foi produzido a partir da interpretação do *script* definido no listing 5.

VI. Conclusão

Este trabalho apresentou o desenvolvimento de uma linguagem textual interpretada voltada à modelagem de processos de negócio segundo a notação BPMN, juntamente com uma *API REST* e uma interface web integrada. A aplicação foi concebida para suprir limitações de ferramentas gráficas tradicionais, oferecendo uma alternativa prática, extensível e de fácil integração com outros sistemas.

Durante o processo de desenvolvimento, foram definidos e implementados os fundamentos teóricos e tecnológicos necessários para a construção da solução. Foram aplicados conceitos de BPM, BPMN 2.0, linguagens específicas de domínio (DSLs) e arquitetura REST, aliados às tecnologias *Spring Boot*, *Angular*, *ANTLR v4* e *Camunda Model API*. Essa combinação permitiu criar um sistema capaz de interpretar *scripts* textuais e gerar automaticamente diagramas BPMN em formato XML compatível com o padrão internacional.

A linguagem desenvolvida possui sintaxe simples e intuitiva, permitindo a criação de tarefas, eventos, gateways e fluxos de sequência de maneira textual. O interpretador foi implementado com o auxílio do ANTLR v4, que realiza a análise léxica e sintática dos scripts. A API processa as requisições e retorna o XML gerado, que pode ser visualizado e manipulado na interface web. A integração com a biblioteca *BPMNJS Viewer* possibilita a renderização dinâmica dos diagramas e ajustes de layout em tempo real.

Os resultados obtidos demonstraram que a aplicação atende aos requisitos funcionais definidos, permitindo a modelagem de processos de forma rápida, consistente e padronizada, sem a necessidade de ferramentas gráficas complexas. Além disso, o sistema mostrou-se de fácil uso e integração, podendo ser incorporado em diferentes contextos acadêmicos e corporativos.

Devido ao uso da biblioteca Camunda para interpretar e validar os modelos, todos os diagramas gerados pela aplicação são garantidamente válidos segundo o padrão internacional BPMN. Caso algum elemento ou estrutura não

esteja em conformidade, o sistema identifica o problema e retorna um erro de compilação, evitando a geração de diagramas incorretos.

Apesar dos resultados alcançados, o trabalho apresentou algumas limitações. A principal delas foi a escolha de um conjunto reduzido de entidades e elementos BPMN para implementação, focando apenas no necessário para demonstrar a viabilidade da linguagem e da API. Essa redução permitiu entregar uma solução funcional, porém com cobertura limitada da notação completa.

Como trabalhos futuros, recomenda-se a evolução da aplicação para permitir a execução automatizada dos processos modelados, a adição de suporte a elementos avançados da notação BPMN (como eventos intermediários e subprocessos) e a ampliação da interface web com recursos colaborativos e de versionamento de modelos.

Conclui-se que o desenvolvimento da linguagem e de sua API resultou em uma ferramenta funcional e inovadora, capaz de contribuir significativamente para a democratização e simplificação da modelagem de processos de negócio, reforçando o potencial das DSLs como instrumentos de abstração e automação em ambientes computacionais modernos.

REFERÊNCIAS

[1] A. Campos. *BPMN – Modelagem e Simulação de Processos de Negócio*. Editora Atlas, 2013. ISBN: 978-8574525847.

[2] K. Talebi, M. Imanipour e J. Rezazadeh. "Business Process Management (BPM) Implementation and Adoption in SMEs: Inhibiting Factors for Iranian ERetail Industry". Em: *ResearchGate* (2012). Available: https://www.researchgate.net/publication/228217184. Accessed: 2025-06-04.

[3] M. Dumas et al. *Fundamentals of Business Process Management*. 2nd. Springer, 2018. ISBN: 978-3642195554.

[4] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. 4th. Springer, 2024. ISBN: 978-3-662-69517-3. DOI: 10.1007/978-3-662-69518-0.

[5] Object Management Group. *BPMN 2.0 Specification*. Available: https://www.bpmn.org/. Accessed: 2025-06-04. 2024.

[6] B. Silver. *BPMN Method and Style*. 2nd. CodyCassidy Press, 2011. ISBN: 978-0963852912.

[7] R. W. Sebesta. *Concepts of Programming Languages*. 9th. AddisonWesley, 2010. ISBN: 978-0131395312.

[8] P. Deitel e H. Deitel. *Java: Como Programar*. 10th. Pearson, 2016. ISBN: 978-0134448237.

[9] P. Van Roy e S. Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004. ISBN: 978-0262100744.

[10] A. V. Aho et al. *Compilers: Principles, Techniques, and Tools*. 2nd. Boston, MA: AddisonWesley, 2006. ISBN: 978-0321486813.

[11] TIOBE Software. *TIOBE Index for April 2024*. Available: https://www.tiobe.com/tiobe-index/. Accessed: 2025-04-18. 2024.

[12] Stack Overflow. *Stack Overflow Developer Survey 2024*. Available: https://survey.stackoverflow.co/2024/. Accessed: 2025-04-18. 2024.

[13] MuleSoft. *What is a RESTful API?* Available: https://www.mulesoft.com/api/rest/what-is-rest-api. Accessed: 2025-06-04. 2024.

[14] L. Gupta. *HTTP Methods*. Available: https://restfulapi.net/http-methods/. Accessed: 2025-06-04. 2023.

[15] R. T. Fielding. "Architectural Styles and the Design of Networkbased Software Architectures". Available: https://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm. Tese de dout. University of California, Irvine, 2000.

[16] Angular Team. *Angular Documentation*. Available: https://angular.dev/overview. Accessed: 2025-04-21.

[17] T. Parr. *ANTLR v4 Documentation*. Available: https://www.antlr.org. Accessed: 2025-04-21. 2025.

[18] Atlassian. *Trello Guide*. Available: https://trello.com/guide. Accessed: 2025-06-04. 2024.

[19] bpmn.io. *bpmn.io – BPMN Toolkit*. Available: https://bpmn.io. Accessed: 2025-06-04. 2024.

[20] Camunda Services GmbH. *Camunda Documentation*. Available: https://docs.camunda.org. Accessed: 2025-06-04. 2024.

[21] O. Vanilla et al. *Cesno: Possibility of Creating a New Programming Language*. Available: https://www.researchgate.net/publication/369592169. Accessed: 2025-06-04. 2023.

[22] R. L. Cancian e A. P. Canal. "Desenvolvimento de uma API REST utilizando os princípios SOLID para proporcionar o acesso a dados públicos para aplicações no Brasil". TCC, Univ. Franciscana, 2023. Available: https://tfgonline.lapinf.ufn.edu.br/media/midias/Artigo$_V644bJn.pd_j$ $2025 - 06 - 04$.

[23] J. V. de Camargo. "A Complementary Analysis of BPMN 2.0Based Tools Behavior Regarding Process Modeling Problems". Available: https://lume.ufrgs.br/handle/10183/235216. Accessed: 2025-06-04. Diss. de mestr. UFRGS, 2021.

[24] K. Beck e et al. *Manifesto for Agile Software Development*. Available: https://agilemanifesto.org/. Accessed: 2025-05-04. 2001.