

RODRIGO PERRUCCI MACHARELLI

**GERADOR DE CÓDIGO PARA JVM A PARTIR
DE FLUXOS DE INTEGRAÇÃO SEGUINDO
PADRÕES EIP**

São Paulo
2020

RODRIGO PERRUCCI MACHARELLI

**GERADOR DE CÓDIGO PARA JVM A PARTIR
DE FLUXOS DE INTEGRAÇÃO SEGUINDO
PADRÕES EIP**

Trabalho apresentado à Escola Politécnica
da Universidade de São Paulo para ob-
tenção do Título de Engenheiro .

São Paulo
2020

RODRIGO PERRUCCI MACHARELLI

**GERADOR DE CÓDIGO PARA JVM A PARTIR
DE FLUXOS DE INTEGRAÇÃO SEGUINDO
PADRÕES EIP**

Trabalho apresentado à Escola Politécnica
da Universidade de São Paulo para ob-
tenção do Título de Engenheiro .

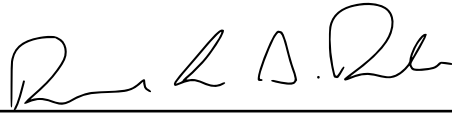
Área de Concentração:

Engenharia de Computação

Orientador:

Prof. Dr. Ricardo Luis de Azevedo
da Rocha

São Paulo
2020

A handwritten signature in black ink, appearing to read 'Ricardo Luis de Azevedo da Rocha', written in a cursive style.

Ricardo Luis de Azevedo da Rocha

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Catálogo-na-publicação

Macharelli, Rodrigo

Gerador de código para JVM a partir de fluxos de integração seguindo padrões EIP / R. Macharelli -- São Paulo, 2020.

88 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.MONTADORES E COMPILADORES 2.GERAÇÃO DE CÓDIGO
3.LINGUAGEM DE PROGRAMAÇÃO I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

AGRADECIMENTOS

Ao professor Ricardo Luis de Azevedo da Rocha pela orientação, acompanhamento e contribuição para a elaboração e realização deste projeto.

Ao Francisco Elias Barguil por ter possibilitado a realização do projeto de TCC como um projeto de estágio na Opus Software.

As equipes da Opus Software que auxiliaram na realização e organização de projeto deste trabalho.

RESUMO

A dinâmica da evolução dos processadores, passando para arquiteturas paralelas com diversos núcleos e a disponibilidade de infraestrutura de computação distribuída influenciaram a forma com que novas aplicações são concebidas e implementadas, muitas vezes fruto da utilização de sistemas diferentes e independentes para a criação da aplicação final. Esse tipo de arquitetura requer abordagens relacionadas à comunicação e integração entre os diversos sistemas que os constituem. Diversas formas e padrões foram propostos para auxiliar no processo de desenvolvimento de sistemas de integração, em particular os conceitos de Enterprise Integration Patterns (EIP), que podem ser implementados com diferentes tecnologias, sendo uma delas o framework Apache Camel, que incorpora diretamente esses conceitos. Esse projeto aborda o desenvolvimento de uma aplicação Web que auxilie o processo de criação dos fluxos de integração de sistemas utilizando uma linguagem de programação visual desenvolvida, por meio de uma interface gráfica simples e acessível, capaz de gerar códigos e projetos Java que consomem as funcionalidades do Apache Camel. Esta aplicação também pode ser estendida para funcionamento com outras linguagens de programação visuais e para geração de diferentes tipos de código.

Palavras-Chave Compiladores, Geração de Código, Linguagens de Programação Visuais, Enterprise Integration Patterns, Fluxos de Integração, Apache Camel, Aplicação Web

LISTA DE FIGURAS

1	Representação de Rotas	23
2	Arquitetura da Aplicação Cliente-Servidor	38
3	Detalhes dos subsistemas do back-end	39
4	Protótipo da Interface com o Usuário - uma visão geral da interface da aplicação, as figuras 6 e 7 mostram detalhes específicos de forma mais clara.	39
5	Diagrama de sequência de interações do usuário	40
6	Exemplo do painel de configuração para o elemento Content Based Router. A Figura 7 mostra outro tipo de painel de configuração.	42
7	Diferentes apresentações do painel de configuração para diferentes protocolos para mensagens.	42
8	Exemplo de código gerado a partir do fluxo de integração de exemplo. . . .	44
9	Tela de Cadastro e Login no Sistema	46
10	Tela base da aplicação, contendo o canvas principal com o qual o usuário pode interagir	46
11	Diagrama entidade relacionamento utilizado para criação do banco de dados do projeto	50
12	Diagrama dos componentes do sistema de geração de código para sistemas de integração Apache Camel	51
13	Diagrama da arquitetura do sistema de geração de código para a linguagem Kaleidoscope	57
14	Diagrama para a implementação da função de obtenção de números da sequência de Fibonacci. Figura em maior escala presente no Apêndice A. .	60
15	Exemplo do código Kaleidoscope gerado e resultado da execução das chamadas da função “fib” para os valores 4, 5 e 10.	61
16	Diagrama da estrutura da situação a ser simulada utilizando a aplicação para auxiliar a geração de código de integração para os sistemas 1 e 2. . . .	65

17	Diagrama utilizando os elementos EIP disponíveis na aplicação para a estruturação de um fluxo de integração.	66
18	Diagrama utilizando os elementos EIP disponíveis na aplicação para a estruturação de um fluxo de integração.	67
19	Exemplo da estrutura dos arquivos gerados automaticamente pela aplicação.	68
20	Exemplo de arquivo de rotas e arquivo de configuração gerados automaticamente.	69
21	Detalhe da tela de Cadastro e Login no Sistema	78
22	Detalhe da tela base da aplicação, contendo o canvas principal com o qual o usuário pode interagir	79
23	Detalhe do diagrama para a implementação da função de obtenção de números da sequencia de Fibonacci.	80
24	Simple diagrama para exemplificação da geração de código para BASIC. .	84
25	Exemplo do código BASIC gerado.	84

SUMÁRIO

1	Introdução	17
1.1	Contextualização	17
1.2	Objetivo	19
1.3	Motivação	19
1.4	Justificativa	20
2	Aspectos Conceituais	21
2.1	Paralelismo	21
2.2	Modelo do Ator	22
2.3	Padrões de Integração para Sistemas Distribuídos	22
2.4	Enterprise Integration Patterns - EIP	22
2.5	Geração de Código	24
2.6	Programação Visual	25
3	Tecnologias Utilizadas	27
3.1	Front-End	27
3.1.1	Flutter	27
3.2	Back-End	28
3.2.1	HTTP	28
3.2.2	REST API	28
3.2.3	RPLY	29
3.2.4	LLVM	29
3.2.5	Apache Camel e Java	29
3.2.6	PostgreSQL	30

3.2.7	Travis CI	30
3.2.8	Ferramentas de Nuvem	30
4	Metodologia do Trabalho	31
4.1	Conceitos e Ferramentas	31
4.2	Protótipo Funcional	31
4.3	Complementação e Refinamento	32
4.4	Expansão e Finalização	32
5	Especificação do Sistema	33
5.1	Requisitos do Sistema	33
5.2	Casos de Uso	34
5.3	Arquitetura da Aplicação	38
5.4	Front-End	39
5.4.1	Diagrama de Fluxos de Integração	40
5.4.2	Painel de Seleção dos Elementos EIP	41
5.4.3	Painel de Configuração dos Elementos EIP	41
5.5	Back-End	43
5.5.1	Servidor Web	43
5.5.2	Sistema de Gerenciamento de Usuários e Projetos	43
5.5.3	Sistema de Geração de Código	43
6	Implementação do Projeto	45
6.1	Front-End	45
6.1.1	Desenvolvimento da Interface	45
6.1.2	Representação dos Elementos EIP	47
6.1.3	Elementos Arrastáveis	47
6.1.4	Canvas Principal	48

6.1.5	Interatividade dos Componentes do Diagrama	48
6.2	Back-End	49
6.2.1	Servidor Web	49
6.2.2	Sistema de Gerenciamento de Usuários e Projetos	49
6.2.2.1	Estrutura do Banco de Dados	49
6.2.2.2	Sistema de Cadastro e Login	50
6.2.3	Sistema de Geração de Código	51
6.2.3.1	Parser	52
6.2.3.2	Gerador de Código	53
6.2.3.3	Gerador de Projeto	54
7	Expansão do Projeto	55
7.1	Proposta	55
7.2	Alterações no Front-End	56
7.3	Alterações no Back-End	56
7.3.1	Parser	57
7.3.2	Gerador de Código Kaleidoscope	58
7.3.3	Compilador Kaleidoscope para IR LLVM	58
7.3.4	Compilação da IR e Montagem	59
7.4	Exemplo de Funcionamento	59
8	Testes	63
8.1	Testes Unitários	63
8.2	Testes de Integração	63
8.3	Testes de Geração de Código	64
8.4	Simulação de Uso	64
8.4.1	Explicação da Simulação	64
8.4.2	Especificação das Estruturas dos Arquivos e Forma de Comunicação	65

8.4.3	Proposta de Solução	66
8.4.4	Geração das Rotas e de Projeto Camel	67
9	Considerações Finais	71
9.1	Conclusões do Projeto de Formatura	71
9.2	Contribuições	72
9.3	Perspectivas de Continuidade	73
9.4	Acesso ao Projeto e Código Fonte	74
	Referências	75
	Apêndice A – Imagens da Interface da Aplicação	77
	Apêndice B – Gerador de Código BASIC	81
B.1	Alterações no Front-End	81
B.2	Alterações no Back-End	81
B.2.1	Parser	82
B.2.2	Gerador de Código Basic	83
B.2.3	Compilador BASIC Para Linguagem de Máquina	83
B.3	Exemplo de Funcionamento	83

1 INTRODUÇÃO

Neste capítulo são apresentadas as principais ideias e propostas deste projeto: qual o contexto em que se insere, qual objetivo pretende-se alcançar, quais as motivações e justificativa para sua realização.

1.1 Contextualização

Os processadores atuais seguem arquiteturas multi-core [1], transformando a forma como softwares são estruturados e planejados, com foco em programação paralela simultânea, que por sua vez gera diferentes desafios e problemas que devem ser solucionados.

A programação paralela mostra-se difícil de implementar de forma correta, exigindo experiência e habilidade por parte do desenvolvedor [2, 3, 4]. Porém sua utilização correta é essencial para obter mais velocidade de processamento, resultando na dificuldade de escalar os sistemas verticalmente explorando sua capacidade de multi-processamento.

Uma forma de explorar os conceitos de paralelismo de forma a não depender da escalabilidade vertical do sistema é a estratégia de escalar o sistema horizontalmente, adicionando mais nós de processamento ao sistema. Os sistemas Web são um bom exemplo de utilização deste conceito, alavancados pela crescente facilidade de utilização de sistemas em nuvem, passando a utilizar arquiteturas distribuídas [5].

A distribuição do sistema em diferentes máquinas e serviços gera a necessidade de estabelecimento de um sistema de comunicação eficiente entre os seus diferentes nós. Além disso, diferentes sistemas também devem se comunicar, quando deseja-se integrá-los para que trabalhem em conjunto. Observa-se porém a não homogeneidade nas formas e protocolos das diferentes ferramentas e tecnologias envolvidas em sistemas mais complexos.

Os livros *Patterns of Enterprise Application Architecture* de M. Fowler [6] e *Enterprise Integration Patterns* [7] apresentam propostas de solução para diversos problemas

envolvendo a integração de sistemas, podendo ser utilizados como padrões a serem seguidos.

Tais padrões podem ser implementados com auxílio de bibliotecas e frameworks diversos, com diferentes características e aplicações. Dentre eles encontra-se o Apache Camel [8], que utiliza-se diretamente dos elementos de Enterprise Integration Patterns (EIP).

A utilização de frameworks como o Apache Camel para implementação de soluções de integração permite a utilização dos padrões EIP de forma simples e direta, disponibilizando elementos prontos para serem utilizados e configurados conforme a necessidade. Entretanto, para o desenvolvedor de um projeto de integração, é necessário que se entenda as diferentes funcionalidades disponibilizadas neste framework, além das diferentes formas e métodos disponíveis para implementação de uma solução em particular.

Dentro do modelo de programação do Apache Camel a codificação de um projeto de integração pode ser separado em duas partes: preparação da estrutura do projeto e definição das rotas de mensagens.

A primeira parte envolve a criação de um projeto Camel no qual serão inseridas as rotas. O código correspondente normalmente é gerado automaticamente via interfaces de linha de comando de programas de gestão de projetos, como é o caso do Apache Maven, porém nota-se a existência de uma barreira de aprendizagem inicial, relacionada, em particular, com a necessidade de importação de diferentes pacotes e bibliotecas Java relacionadas ao Apache Camel.

A segunda parte consiste da codificação específica das rotas de tratamento e roteamento de mensagens para formar os caminhos de integração entre sistemas, utilizando os elementos EIP disponíveis. Este processo normalmente é feito diretamente pelo desenvolvedor e é escrito diretamente em Java ou outros tipos de linguagem suportados pelo Camel.

Há aplicações atualmente que atuam para facilitar o desenvolvimento de projetos Camel possibilitando a implementação das rotas por meio de diagramas. Um exemplo é a ferramenta JBoss Fuse, mantida pela Red Hat. Nestes diagramas os elementos EIP são disponibilizados para serem interconectados entre si e configurados de forma individual para facilitar a criação das rotas, gerando o código correspondente automaticamente.

Estas aplicações normalmente existem dentro de outros contextos ou na forma de extensões para editores e outras aplicações, como é o caso do JBoss Fuse que está disponível como uma extensão da IDE Eclipse.

1.2 Objetivo

O objetivo principal deste projeto de TCC é projetar e desenvolver uma aplicação com base em um editor web de fluxos de integração utilizando uma Linguagem de Programação Visual, a ser desenvolvida, capaz de gerar código Java utilizando as funcionalidades do Apache Camel.

De forma detalhada, deve-se desenvolver um editor WEB de fluxos de integração, baseado na estruturação de diagrama, utilizando-se dos componentes e elementos EIP, e um gerador de código em Java (que pode ser modificado de forma incremental), partindo da representação gerada na interface.

Como objetivos secundários pretende-se mostrar a utilização da aplicação para geração de código de diferentes tipos, partindo de diferentes representações gráficas, mostrando a possibilidade de expansão e generalização dos conceitos empregados em seu desenvolvimento.

Dentre as possibilidades de expansão, pretende-se desenvolver uma representação gráfica de uma linguagem de programação de propósito geral simples, que pode ser utilizada para gerar código executável a partir de sua diagramação, mostrando as possibilidades de utilização da estrutura geral da aplicação.

Todo o software foi desenvolvido em código aberto.

1.3 Motivação

A utilização dos conceitos e elementos do EIP é uma boa prática para a solução de problemas de integração e as ferramentas e bibliotecas relacionadas auxiliam diretamente na sua execução, porém a complexidade e o número de elementos envolvidos em projetos de integração podem dificultar sua implementação.

Dentro do ambiente de programação Java, assim como em diversos outros, nota-se uma tendência de produção de software e ferramentas que auxiliam o processo de desenvolvimento. Isso pode ser realizado de diversas formas, seja por meio de ferramentas de gerenciamento de pacotes e automação de construção de projetos, como o Apache Maven [9], Ant[10] ou até mesmo para geração automática de código comumente utilizado (chamado “código boilerplate”), como é o caso do Projeto Lombok.

Uma outra forma de facilitar a criação de sistemas é a utilização de editores visuais,

que possibilitam a geração de artefatos de software a partir da elaboração de diagramas. No contexto deste projeto, pode-se observar a possibilidade de implementação das rotas Camel, que podem ser facilmente mapeadas para uma linguagem visual simples, utilizando diretamente os elementos EIP.

A ferramenta JBoss Fuse Tooling, desenvolvida pela Red Hat, é um bom exemplo deste conceito. Funcionando como uma interface para utilização do Apache Camel, implementada utilizando o conceito de diagramação ou programação visual. Porém, sua utilização está associada com a IDE Eclipse, sendo de fato uma extensão que deve ser instalada.

Este projeto propõe-se a implementar uma aplicação que possa existir independentemente e acessível com interface Web, de forma a facilitar a utilização e elaboração dos fluxos de integração por meio da disponibilização de um editor visual e geração de automática de código, abstraindo os detalhes de organização de pacotes e da estrutura do projeto.

Outro fator de interesse é a forma de geração de código associada a este projeto, possibilitando a generalização do conceito de geração a partir de diagramas de fluxo para outras linguagens e aplicações que não sejam relacionadas a sistemas de integração.

1.4 Justificativa

A aplicação desenvolvida tende a facilitar a implementação de sistemas complexos de integração e garantir o emprego de boas práticas e padrões consolidados em suas elaborações por meio da utilização dos elementos e ideias trazidos pelo EIP.

A disponibilidade da interface Web independente, a automação de diversas etapas de geração de código, organização do projeto, gerenciamento de pacotes e bibliotecas colaboram para facilitar o acesso à ferramenta e possibilitam interações mais simples com o desenvolvedor.

Além disso, nota-se que esta aplicação pode ser estendida para geração de diferentes tipos de código, baseados em diversas linguagens de entrada, disponibilizando um sistema de edição para Linguagens Visuais de modo geral, podendo ser reutilizada para diferentes finalidades.

2 ASPECTOS CONCEITUAIS

Apresenta-se, a seguir, os principais conceitos e técnicas utilizados como base para a elaboração e implementação deste projeto, contendo breves explicações sobre os assuntos e quais as suas finalidades para a realização do trabalho.

2.1 Paralelismo

O desenvolvimento de software sofreu uma mudança significativa devido à mudança do núcleo único para o núcleo múltiplo no design da arquitetura do processador. Para extrair mais velocidade, os desenvolvedores precisavam projetar seu aplicativo com paralelismo em mente [11].

Infelizmente, obter a mesma velocidade de processamento provou ser extremamente difícil, como mostra a Lei de Amdahl [12]. Os aplicativos baseados na Web voltaram sua atenção para a escalabilidade horizontal, com o aumento da hospedagem e da computação em nuvem. Assim, a natureza desses aplicativos se tornou multithread e distribuída [5].

Paralelizar um aplicativo de API significa acessar simultaneamente o banco de dados. Em um ambiente com um único banco de dados, isso significa que as corridas de dados ocorrerão quando as solicitações precisarem de acesso exclusivo ao mesmo recurso [13].

A declaração acima demonstra um impacto significativo no desempenho, pois uma transação deve bloquear recursos. A otimização de operações quando o recurso está bloqueado é possível, mas não altera o gargalo de desempenho.

Como Hewitt [14] afirmou, “O advento da concorrência maciça por meio da computação em nuvem cliente e arquiteturas de computadores com muitos núcleos despertou interesse no modelo de ator”.

2.2 Modelo do Ator

O modelo computacional amplamente utilizado para definir a arquitetura do computador e as linguagens de programação é a Máquina de Turing. O design do processador, linguagens que variam de Algol a C#, define um modelo interno baseado na transformação de estado [15, 16].

O modelo de ator escolheu outra direção desde o início [17, 18], a sua base é o estilo de passagem de mensagens assíncrona, sem efeitos colaterais, assim ele fornece uma maneira diferente de lidar com a simultaneidade. Além disso, no modelo de ator, a ordem sequencial é um caso particular, derivado da computação simultânea, que algumas linguagens funcionais como Scala permitem [19].

2.3 Padrões de Integração para Sistemas Distribuídos

A arquitetura das aplicações distribuídas apresenta um conjunto vasto de problemas cuja solução demanda proposições distintas, em geral baseada em troca de mensagens.

No livro sobre Patterns of Enterprise Application Architecture de M. Fowler [6], apresenta-se um conjunto amplo de padrões de arquitetura para a solução de problemas. Neles observa-se que o emprego da troca de mensagens assíncronas torna-se uma medida não apenas desejável, mas necessária.

Observa-se claramente que o uso de mensagens assíncronas colabora fortemente para a solução dessas questões, embora tenha a contrapartida de acrescentar novos desafios como [7]: modelo de programação complexo, questões ligadas à sincronia (em cenários síncronos ou sequenciais), questões ligadas ao desempenho, existência de suporte à plataforma.

Esses problemas afetam o desenvolvimento de projetos, e algumas das práticas foram compiladas em padrões. O padrões adotados em projetos que envolvem integração foram compilados no denominado

2.4 Enterprise Integration Patterns - EIP

Os conceitos de Design Patterns expostos no livro Design Patterns: Elements of Reusable Object-Oriented Software (E. GAMMA, R. HELM, R. JOHNSON, and J. VLISSIDES) [20] aplicados a de integração de sistemas resultou nos padrões apresentados no livro Enterprise Integration Patterns [7], que descreve diversos problemas e suas resoluções.

Trata-se de guias e ideias de estrutura e design de soluções para problemas de integração independentes de tecnologia, produto da compilação e estudo de soluções existentes que foram implementadas por diferentes desenvolvedores mas apresentam propostas de solução semelhantes.

Os diferentes elementos e padrões que o constituem podem ser utilizados em diversos cenários de forma bastante similar, diferindo essencialmente nas tecnologias específicas com as quais interagem, facilitando o processo de desenvolvimento da arquitetura dos sistemas.

Estes elementos podem ser entendidos como blocos independentes que efetuam ações bem definidas e podem ser interconectados com outros elementos.

Dentre os padrões apresentados no livro *Enterprise Integration Patterns* [7] incluem-se diferentes tipos de estilos de integração, padrões de mensagem, canais, roteamento, transformação de dados, entre outros.

Um conceito de especial importância para este projeto é a criação de rotas, dentro do contexto Apache Camel. Estas permitem criar uma sequência de ações e padrões, representados pelos elementos EIP, a serem aplicadas sobre uma determinada mensagem. As rotas serão geradas diretamente a partir do diagrama de integração de fluxo criado na interface pelo usuário.

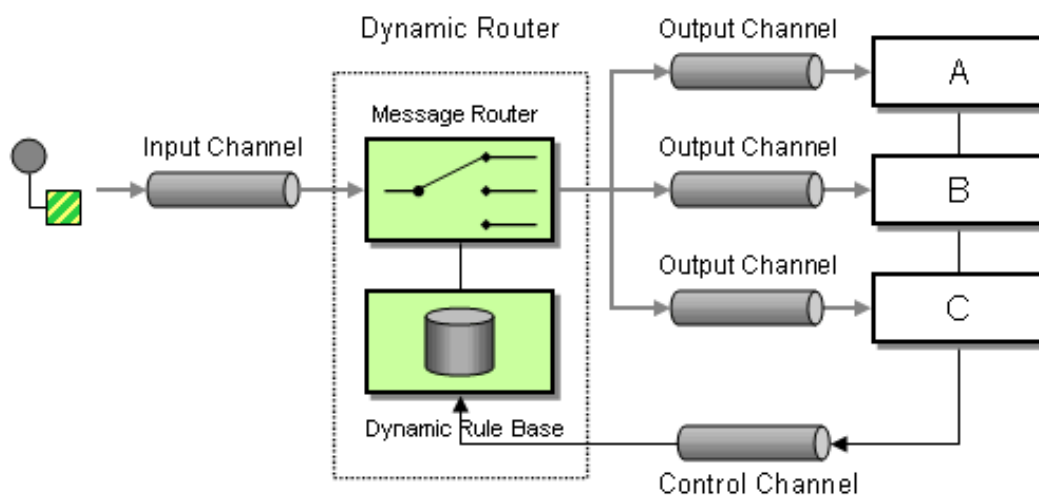


Figura 1: Representação de Rotas

Fonte: https://camel.apache.org/components/latest/eips/_images/eip/DynamicRouter.gif

Por se tratar de conceitos independentes de tecnologia, pode-se encontrar diferentes implementações e ferramentas que utilizam-se das ideias propostas no livro. Em especial há frameworks que traduzem os elementos EIP quase de forma direta, incorporando-os

no seu funcionamento. Apache Camel e Spring Integration são exemplos bons, nos quais pode-se encontrar os padrões EIP disponíveis para serem utilizados em suas APIs.

2.5 Geração de Código

Os conceitos utilizados para transformar a representação dos fluxos de integração de diagramas para código Java são semelhantes aos empregados em compiladores, contidos nos livros *Compilers: Principles, Techniques, and Tools* (A. V. Aho, M. S. Lam, R. Sethi, e J. D. Ullman) [21] e *Introdução à Compilação* (João José Neto) [22].

O objetivo do processo de compilação é o de transformar o código fonte fornecido em outro código equivalente. Em um caso comum, tem-se como entrada arquivo em formato de texto escrito em uma linguagem de programação de alto nível (código fonte) e como saída o equivalente código de baixo nível específico para uma determinada arquitetura de computador.

O processo de geração de código se dá, de maneira geral, seguindo uma sequência de passos e tratamentos a partir do código fonte disponível, utilizando os processos de análise léxica, análise sintática, análise semântica, geração de código intermediário, otimizações e geração de código final.

Os compiladores são capazes de verificar a correta utilização de linguagens de programação pelos desenvolvedores, baseados nas regras gramaticais que definem tais linguagens. Desta forma, a correta elaboração da gramática da linguagem é essencial para que se possa desenvolver o compilador correspondente.

Os processos de análise léxica e sintática são responsáveis pela extração de informações do código fonte de forma possibilitar a verificação das construções gramaticais empregadas quanto a sua estrutura sintática, isto é, se os elementos da linguagem estão organizados e dispostos de forma correta.

A análise semântica tem como objetivo a verificação do código analisado no que diz respeito as regras de escopo, visibilidade, consistência de tipos de variáveis e operandos, existência de variáveis, entre outros. De forma geral, são responsáveis por validar as informações que não podem ser expressas diretamente pelas gramáticas livres de contexto, normalmente utilizadas para descrição de linguagens de programação.

Embora sejam necessários ajustes e adaptações para o formato de entrada deste projeto (diagramas de fluxo), por diferir do formato convencional utilizado em compiladores

de linguagens convencionais (cadeias de caracteres), diversos conceitos continuam sendo de grande utilidade.

2.6 Programação Visual

Linguagens de programação visual (LPV) possibilitam ao usuário desenvolver programas de forma gráfica, por meio de manipulação de elementos visuais em diagramas, ao invés de lidar diretamente com código em representação textual

Cada um destes elementos visuais representam funções ou blocos de código e podem ser conectados entre si, seguindo regras sintáticas e semânticas definidas por uma gramática. As conexões entre os elementos representam o fluxo de dados entre eles. [23].

As linguagens de programação visuais apresentam vantagens e desvantagens claras. Dentre as desvantagens, percebe-se que são pouco interessantes para programas extensos, uma vez que sua representação gráfica pode ser tornar difícil de compreender. Outro fator é a necessidade de utilização de ferramentas específicas para programação visual, em contraste com a possibilidade de utilização de qualquer editor textual para linguagens de programação convencionais.

Há, porém, vantagens na utilização das LPV. A facilidade de entendimento da estrutura do código por meio de sua visualização de forma gráfica pode ser considerado um fator positivo. Este fato contribui para a grande utilização deste tipo em ambientes educacionais, em especial no aprendizado de programação. Um exemplo de aplicação interessante é a linguagem Scratch [24] desenvolvida no MIT desenvolvida para auxiliar no processo de aprendizagem de programação para jovens e crianças.

Além disso, pode-se adicionar camadas extras de abstração com a utilização de elementos visuais (blocos) com funcionalidades complexas, que podem ser utilizados e incorporados facilmente ao programa, bastando inseri-lo no diagrama.

Por fim, o fato de as LPV estarem acopladas aos seus respectivos editores visuais pode trazer benefícios no que diz respeito as possibilidades de auxílio na configuração e utilização dos componentes da linguagem, podendo apresentar sugestões e opções pré estabelecidas que permitem ao programador entender as possibilidades e funcionalidades de determinada componente da linguagem sem necessariamente ter de recorrer a sua documentação.

3 TECNOLOGIAS UTILIZADAS

A realização deste projeto envolve a elaboração de uma aplicação Web, baseada no modelo de implementação cliente-servidor, no qual o usuário interage com a interface Web (front-ent), que por sua vez se comunica com o servidor (back-end) para obter dados e efetuar processamentos.

As tecnologias serão apresentadas em seus contextos de utilização tanto no front-end quanto no back-end.

3.1 Front-End

Esta seção apresenta a principal ferramenta utilizada para a implementação da interface da aplicação, assim como o motivo para sua escolha.

3.1.1 Flutter

Para o desenvolvimento da interface Web da aplicação utiliza-se o framework Flutter. Trata-se de uma ferramenta desenvolvida pelo Google para possibilitar a criação de interfaces de aplicações para plataformas Android, iOS, Web e Desktop utilizando a mesma base de código. [25]

Utiliza a linguagem de programação Dart, também desenvolvida pela equipe do Google, e é capaz de gerar aplicativos nativos para as plataformas, sendo assim mais rápidos e eficientes.

Os projetos realizados com o Flutter podem ser compilados (transpilados) para JavaScript e HTML, de forma que podem ser hospedados facilmente em muitas plataformas.

O Flutter foi escolhido para implementação do front-end devido a sua grande versatilidade e facilidade de implementação para diferentes plataformas. Além disso apresenta uma comunidade ativa com diversas fontes de aprendizado e pesquisa. Outras tecnologias

poderiam ser utilizadas para tal fim.

Atualmente existem diversos frameworks para desenvolvimento de interfaces em diferentes linguagens: Angular, React e Vue são exemplos frequentemente utilizados, cada qual com suas vantagens e limitações.

3.2 Back-End

Os principais conceitos e tecnologias utilizados para a implantação do back-end da aplicação são mostrados nesta seção, mostrando como são utilizados de forma a possibilitar a comunicação com o front-end e o processamento das informações para geração de código e projetos diversos.

3.2.1 HTTP

As comunicações entre front e back-end são realizadas via requisições HTTP - Hypertext Transfer Protocol.

Trata-se de protocolo de comunicação amplamente utilizado na WEB. Consiste em uma estrutura de mensagens que contém diferentes tipos de métodos (GET, POST, PUT, DELETE, entre outros), campo de cabeçalho e campo de mensagem.

Os diferentes tipos de mensagens podem ser interpretados pelo servidor, realizando diferentes ações. É comum a utilização de convenções e padrões sobre quais tipos de métodos e tipos de mensagens devem ser utilizados em cada caso específico.

3.2.2 REST API

A implementação de servidores WEB pode ser feita de diferentes formas, utilizando diferentes linguagens e frameworks. Node.js e Python são amplamente utilizadas para este fim.

Representational state transfer (REST) é um modelo de organização de software que propõe regras e convenções para comunicação cliente-servidor. Trata-se de um conceito elaborado e apresenta uma série de exigências para que uma API seja considerada RESTful.

Este projeto utiliza algumas ideias e conceitos baseados nas propostas do modelo REST, sem o compromisso de satisfazer todas as características necessárias, porém.

Utiliza-se a linguagem Python em conjunto com o framework Flask neste projeto pela facilidade de implementação e integração do servidor WEB com os outros elementos presentes no back-end, também desenvolvidos em Python.

3.2.3 RPLY

A biblioteca RPLY disponível para Python é capaz de facilitar a geração de analisadores Léxicos e Sintáticos de forma bastante simples. [26]

Disponibiliza uma API bastante simples, que se utiliza das funcionalidades de anotação de código do Python, para auxiliar o desenvolvimento dos analisadores sintáticos. Baseia-se na utilização direta das regras gramaticais da linguagem a ser compilada, utilizando notação baseada na forma Backus–Naur (BNF), amplamente utilizada para representações gramaticais.

3.2.4 LLVM

O LLVM é formado por um conjunto de ferramentas e programas que permitem a criação de compiladores [27], contemplando as partes de análise sintática, semântica, geração e otimização de código, além de possibilitar a execução em tempo real dos códigos gerados.

Em especial é possível gerar, a partir da definição da gramática equivalente, uma representação intermediária de código (IR LLVM) que pode ser utilizado para geração de código otimizado para diversas plataformas e linguagens.

Utilizado em conjunto com o RPLY para a expansão do projeto para a geração de código para linguagens de propósito geral, auxiliando no processo de estruturação do compilador da linguagem.

3.2.5 Apache Camel e Java

A linguagem Java e o framework Apache Camel não são utilizados diretamente na implementação do projeto, mas são, na verdade, o produto esperado da aplicação, na forma de um projeto completo gerado a partir da elaboração do diagrama de integração por parte do usuário.

A utilização do Apache Camel é interessante para este projeto pois disponibiliza APIs simples e diretas para implementação dos conceitos EIP, facilitando o processo geração

de código e execução dos projetos finais.

3.2.6 PostgreSQL

É um sistema de gerenciamento de bancos de dados (SGBD) relacionais de código aberto e gratuito.

Trata-se de um SGBD amplamente utilizado e documentado, tendo sido escolhido para este projeto pela sua facilidade de utilização, em especial sua disponibilidade em serviços de nuvem.

3.2.7 Travis CI

Travis CI é uma ferramenta de integração contínua (Continuous Integration - CI) que tem como objetivo automatizar e facilitar a construção, teste e deploy de projetos armazenados em sistemas de controle de versão como o GitHub.

Sistemas como este são amplamente utilizados em cenários reais de desenvolvimento de projetos, trazendo diversos benefícios e agilizando processos importantes.

3.2.8 Ferramentas de Nuvem

Para o processo de hospedagem e deploy deste projeto foram utilizadas ferramentas e infraestrutura em nuvem.

Em particular foram utilizados os serviços do Heroku para hospedagem do back-end, implementado em Python, e o front-end foi hospedado no GitHub Pages.

4 METODOLOGIA DO TRABALHO

O desenvolvimento da aplicação deve ser realizado de forma incremental e continuada, sendo necessário o entendimento do problema e a elaboração de formas de trabalho.

As etapas básicas a serem seguidas são descritas neste capítulo.

4.1 Conceitos e Ferramentas

A primeira parte no processo de desenvolvimento do projeto se baseia no entendimento e clarificação dos conceitos e ferramentas utilizadas.

Os conceitos de Enterprise Integration Patterns e as aplicações utilizando o Apache Camel são essenciais para a o desenvolvimento do projeto. O estudo destes se baseia na leitura da literatura associada, constituída pelos livros Enterprise Integration Patterns [7], Patterns of Enterprise Application Architecture [6] e Camel in Action [8].

Outra parte importante do projeto é a geração de código, baseado nos conceitos associados a compiladores. O livro Compilers: Principles, Techniques, and Tools [21] pode ser utilizado como fonte para implementação e conceitualização de diversos componentes necessários ao projeto.

4.2 Protótipo Funcional

Com os conceitos essenciais entendidos, parte-se para uma implementação inicial do sistema completo, servindo como um protótipo da aplicação final.

Este protótipo tem como ideia possibilitar a verificação dos conceitos entendidos e validação da estrutura proposta ao projeto. Deve-se obter uma aplicação simples com front-end e back-end de forma a realizar suas funcionalidades básicas.

Para isso, codifica-se a interface e o servidor, incluindo a funcionalidade de dia-

gramação com elementos EIP e geração de código. Não são utilizados todas as funcionalidades e elementos EIP nesse momento, mas sim representantes de tipos diferentes de elementos para garantir a funcionalidade básica do gerador de código.

Implementam-se as funcionalidades de forma modular e de fácil expansão, baseado em conceitos de programação orientada a objetos.

4.3 Complementação e Refinamento

Com a estrutura inicial proporcionada pelo protótipo, pode-se expandir as funcionalidades da aplicação tanto no quesito de elementos disponíveis para a geração dos diagramas, quanto sistemas secundários de autenticação de usuário, interface de salvamento e abertura de projetos e melhorias na interface.

A ideia é que esta fase do desenvolvimento seja facilitada pela elaboração do projeto de forma continuada e modularizada.

Ao final pretende-se obter a aplicação em sua forma mais semelhante com a pretendida ao fim do projeto.

4.4 Expansão e Finalização

Neste momento a estrutura do projeto deve estar consolidada, podendo ser expandida. A proposta é que este seja um projeto em código aberto e que possa ter suas funcionalidades aprimoradas por outros desenvolvedores interessados.

A estruturação do projeto e disponibilização no modelo Open Source estão previstas para esta fase.

Por fim, pretende-se expandir as capacidades do editor visual de forma a ser utilizado não apenas para desenvolvimento de fluxos de integração Apache Camel.

Inclui-se a possibilidade de criação de projetos envolvendo uma linguagem de programação de propósito geral, com seus respectivos blocos visuais e sistema de geração de código, aproveitando as estruturas e funcionalidades dinâmicas de tratamento dos componentes visuais implementados no projeto.

5 ESPECIFICAÇÃO DO SISTEMA

É apresentada neste capítulo a especificação do sistema, contemplando os requisitos funcionais e não funcionais, principais casos de uso e detalhes de arquitetura do front-end e do back-end, introduzindo os componentes do sistema que serão discutidos no capítulo 6.

5.1 Requisitos do Sistema

A aplicação Web desenvolvida neste projeto tem como objetivo final a geração de código para ser utilizado dentro do escopo do framework Apache Camel, sendo esta sua funcionalidade principal.

Há, contudo, outras funcionalidades que o sistema deve ser capaz de realizar, para possibilitar a utilização do usuário final, constituindo os requisitos funcionais do sistema.

Os requisitos funcionais do sistema são:

- RF01 - Cadastrar usuário (registro e login)
- RF02 - Criar e editar diferentes projetos de fluxos de integração utilizando padrões EIP e outros tipos de projetos para diferentes linguagens
- RF03 - Criar diagramas no estilo “flowchart” utilizando-se de elementos de programação visual
- RF04 - Editar e modificar a disposição dos elementos visuais dentro do diagrama
- RF05 - Possibilitar a configuração dos diferentes elementos EIP para criação das rotas do Apache Camel

Como requisitos não funcionais do sistema, listam-se:

- RNF01 - Disponibilizar documentação e suporte para modelo open source

- RNF02 - Apresentar interface simples e intuitiva
- RNF03 - Apresentar fluidez e alta velocidade de resposta

5.2 Casos de Uso

1. Caso de uso: Efetuar login no sistema

- **Descrição:** Procedimento de login no sistema
- **Evento iniciador:** Usuário acessa a página inicial da aplicação
- **Atores:** Usuário
- **Pré-condições:** Usuário cadastrado e não autenticado
- **Sequência de Eventos:**
 - 1. Usuário informa seu e-mail e senha
 - 2. Sistema realiza a autenticação do usuário
 - 3. Usuário é redirecionado para a interface de projetos
- **Pós Condições:** Usuário autenticado no sistema
- **Extensões**
 - 1. E-mail ou senha incorretos. Sistema exibe mensagem de erro ao usuário.
(Passo 2)

2. Caso de uso: Efetuar cadastro no sistema

- **Descrição:** Procedimento de cadastro de novo usuário no sistema
- **Evento iniciador:** Usuário acessa a página inicial da aplicação
- **Atores:** Usuário
- **Pré-condições:** Usuário não cadastrado no sistema
- **Sequência de Eventos:**
 - 1. Usuário informa e-mail e senha para cadastro
 - 2. Sistema verifica que o e-mail não está previamente cadastrado e realiza o cadastro do usuário
 - 3. Sistema autentica o usuário
 - 4. Usuário é redirecionado para a interface de projetos
- **Pós Condições:** Usuário cadastrado e autenticado no sistema

- **Extensões**

- 1. E-mail já cadastrado. Sistema exibem mensagem de erro ao usuário.
(Passo 2)

3. Caso de uso: Criação de Novo Projeto

- **Descrição:** Descrição do processo de criação de novo projeto de diagrama de programação visual
- **Evento iniciador:** Usuário clica em “Novo Projeto”
- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema
- **Sequência de Eventos:**
 - 1. Usuário preenche os campos de tipo e nome do projeto
 - 2. Sistema verifica disponibilidade de nome do projeto
 - 3. Sistema cria novo projeto em branco
- **Pós Condições:** Novo projeto criado e associado ao usuário
- **Extensões**
 - 1. Nome do projeto já sendo utilizado. Sistema exhibe mensagem de erro.
(Passo 2)

4. Caso de uso: Carregamento de Projeto Existente

- **Descrição:** Descrição do processo de carregamento (ou recuperação) de um projeto existente
- **Evento iniciador:** Usuário clica em “Abrir Projeto”
- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema e projetos associados ao usuário no sistema anteriormente
- **Sequência de Eventos:**
 - 1. Usuário escolhe projeto a partir de uma lista de projetos associados
 - 2. Sistema carrega o projeto e o exhibe no diagrama
- **Pós Condições:** Projeto carregado e pronto para edição

5. Caso de uso: Salvamento de Projeto

- **Descrição:** Descrição do processo de salvamento do projeto sendo editado
- **Evento iniciador:** Usuário clica em “Salvar Projeto”
- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema, projeto criado ou carregado sendo editado
- **Sequência de Eventos:**
 - 1. Sistema informa que o projeto foi salvo
 - 2. Usuário volta para a interface de edição do projeto
- **Pós Condições:** Estado corrente do projeto persistido pelo sistema

6. Caso de uso: Inserção de elemento visual no diagrama

- **Descrição:** Descrição do processo de inserção de um elemento visual que compõe a linguagem de programação visual sendo tratada.
- **Evento iniciador:** Usuário arrasta um elemento do painel de seleção para o canvas principal
- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema, projeto criado ou carregado sendo editado
- **Sequência de Eventos:**
 - 1. Usuário seleciona um dos elementos visuais listados no painel esquerdo da interface
 - 2. Usuário arrasta o elemento para o canvas principal
 - 3. Sistema instancia (insere) elemento no canvas principal
 - 4. Sistema disponibiliza interface de configuração do elemento
- **Pós Condições:** Estado corrente do canvas alterado. Novo elemento inserido no diagrama

7. Caso de uso: Configuração de Elemento Visual no Diagrama

- **Descrição:** Descrição do processo de configuração de um elemento visual componente do diagrama
- **Evento iniciador:** Usuário seleciona um elemento presente no canvas principal

- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema, projeto criado ou carregado sendo editado, elemento visual previamente inserido no diagrama
- **Sequência de Eventos:**
 - 1. Usuário seleciona um dos elementos visuais instanciados
 - 2. Sistema disponibiliza interface de configuração do elemento
 - 3. Usuário insere as configurações específicas do elemento sendo configurado, conforme interface disponibilizada
 - 4. Usuário salva as configurações do componente
- **Pós Condições:** Estado corrente do canvas alterado. Elemento configurado no sistema
- **Extensões**
 - 1. Usuário descarta as configurações atuais. Estado do canvas não é atualizado (Passo 4)

8. Caso de uso: Conexão Entre Elementos Visuais

- **Descrição:** Descrição do processo interconexão entre elementos visuais instanciados no diagrama
- **Evento iniciador:** Usuário seleciona um elemento visual no diagrama
- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema, projeto criado ou carregado sendo editado, elementos visual previamente inseridos no diagrama
- **Sequência de Eventos:**
 - 1. Usuário seleciona o primeiro elemento visual, representando a fonte da conexão
 - 2. Usuário seleciona o segundo elemento visual, representando o destino da conexão
 - 3. Sistema realiza a interconexão dos elementos com indicação visual
- **Pós Condições:** Estado corrente do canvas alterado. Elementos conectados.

9. Caso de uso: Geração de Código e Projeto

- **Descrição:** Descrição do processo de geração de código e obtenção da estrutura de projeto a partir do diagrama criado

- **Evento iniciador:** Usuário clica em “Gerar Código”
- **Atores:** Usuário
- **Pré-condições:** Usuário autenticado no sistema, projeto criado ou carregado sendo editado
- **Sequência de Eventos:**
 - 1. Usuário clica em “Gerar Código”
 - 2. Sistema apresenta página contendo o código gerado pelo back-end
 - 3. Usuário clica em “Download do Projeto”
 - 4. Sistema disponibiliza o arquivo compactado do projeto Apache Camel Gerado
- **Extensões**
 - 1. Sistema de geração de código sinaliza erro de compilação do código devido a erros no diagrama. (Passo 2)

5.3 Arquitetura da Aplicação

Trata-se de uma aplicação Web que utiliza os conceitos de REST APIs para realizar a comunicação do front-end com o back-end.

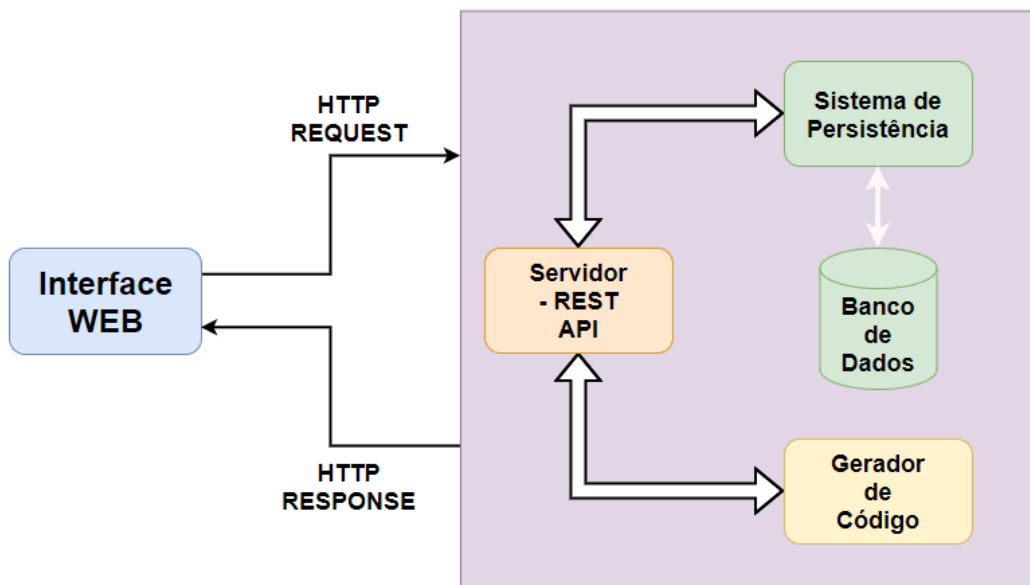


Figura 2: Arquitetura da Aplicação Cliente-Servidor

A estrutura geral da aplicação WEB cliente-servidor pode ser visualizada na imagem anterior, tratando-se de uma arquitetura bastante simples e comumente utilizada.

A figura a seguir apresenta mais informações sobre os subsistemas do back-end.

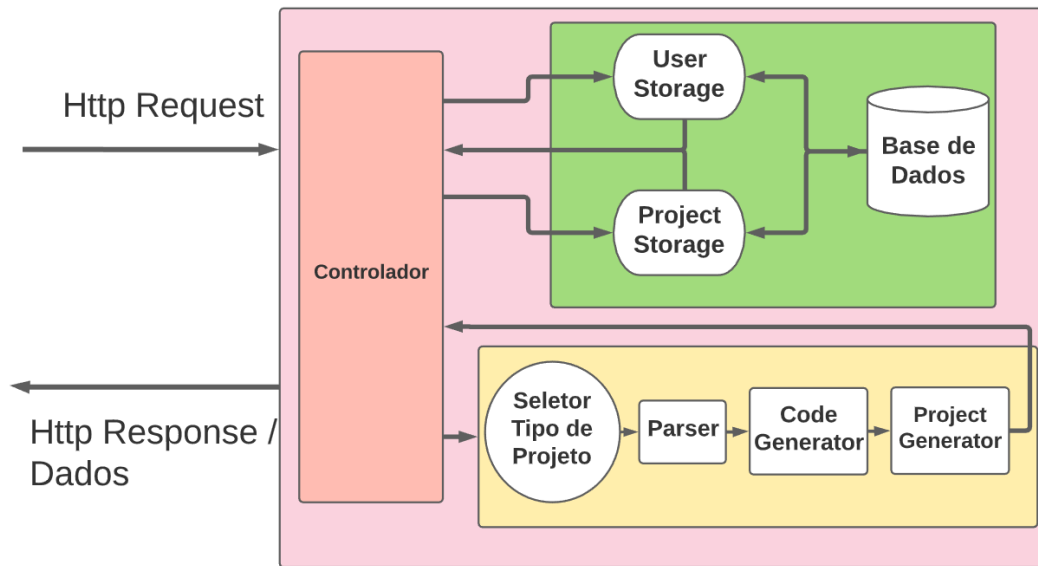


Figura 3: Detalhes dos subsistemas do back-end

Tais sistemas são apresentados com maiores detalhes em seções posteriores.

5.4 Front-End

Representado na parte esquerda da Figura 2 como interface Web.

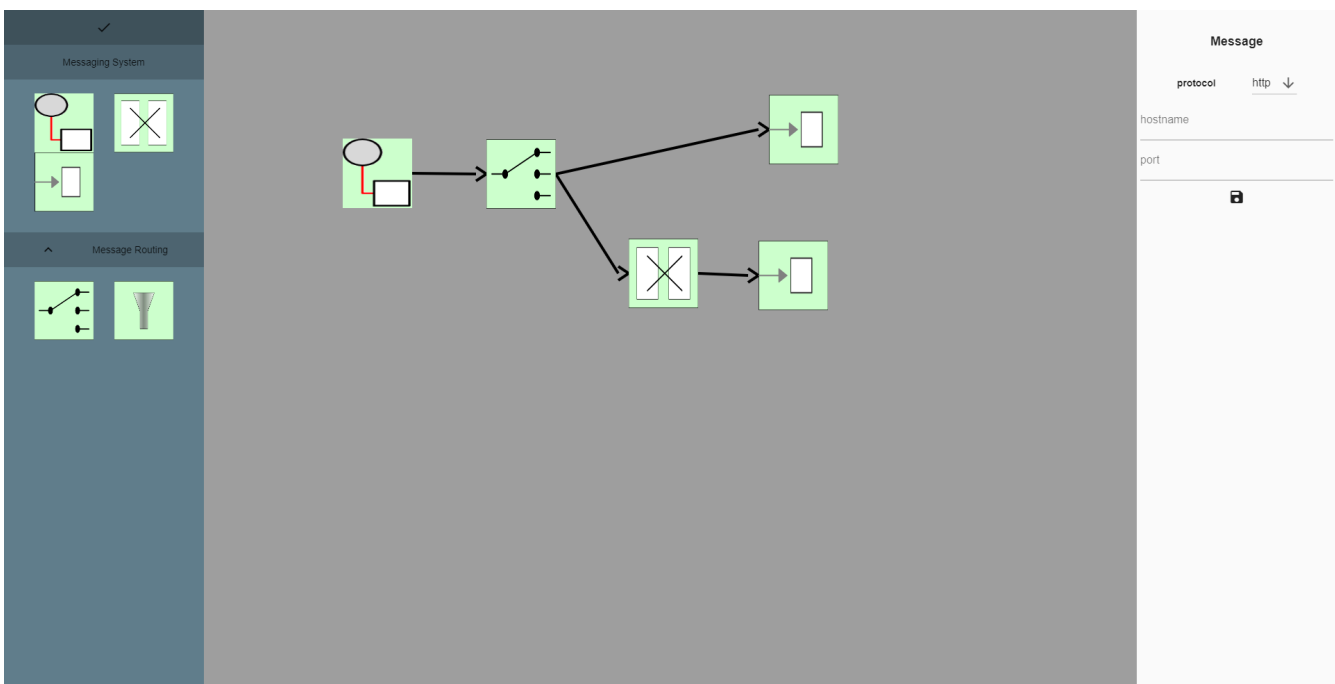


Figura 4: Protótipo da Interface com o Usuário - uma visão geral da interface da aplicação, as figuras 6 e 7 mostram detalhes específicos de forma mais clara.

Responsável pela interação direta com o usuário. Apresenta sistema de gerenciamento de cadastro e projetos, além da interface de criação e edição dos diagramas de integração de sistemas. Comunica-se com o back-end através de requisições HTTP, interagindo via REST API.

A Figura 4 representa um protótipo da interface da aplicação implementada na primeira parte deste projeto.

O diagrama a seguir representa uma possível sequência de interações do usuário com a interface do sistema, representando um fluxo comum de utilização da aplicação.

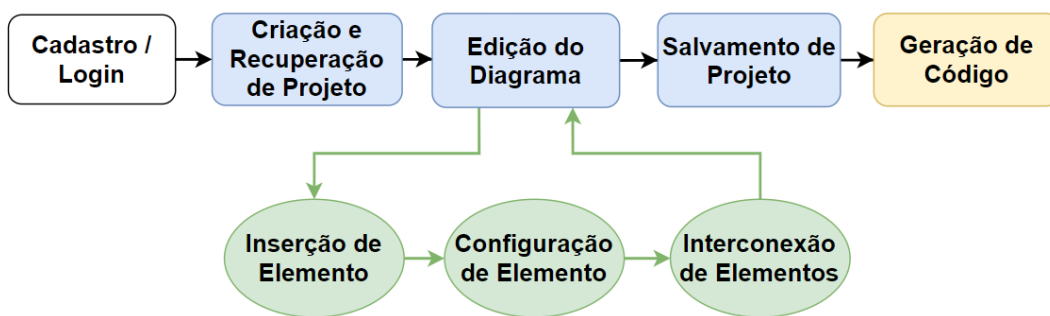


Figura 5: Diagrama de sequência de interações do usuário

Trata-se de uma interface simples, visando a facilidade de utilização e agilidade para criação e edição de projetos.

Os componentes que constituem a interface da aplicação são explicitados em detalhes a seguir.

5.4.1 Diagrama de Fluxos de Integração

Elemento principal do front-end, responsável pela interação com o usuário para gerar os dados que serão processados pelo Gerador de Código para se obter o código final em Java. Está localizado na região central da aplicação.

Consiste em uma região editável, denominada canvas, em que pode-se inserir, remover, mover e conectar elementos EIP, utilizando a interface de “Drag and Drop”.

A Figura 4 mostra uma situação em que elementos EIP foram arrastados do Painel de Seleção dos Elementos EIP (representados pelos retângulos verdes) e inseridos no canvas para formar um fluxo de integração simples.

5.4.2 Painel de Seleção dos Elementos EIP

Na lateral esquerda da imagem acima encontra-se o painel com elementos EIP que podem ser arrastados para a região do canvas.

O painel é dividido por tipos de elementos EIP e pode ser modificado para conter diferentes classes, dependendo da linguagem sendo representada.

5.4.3 Painel de Configuração dos Elementos EIP

O painel de edição dos elementos EIP encontra-se na lateral direita. Esse painel tem como um de seus objetivos facilitar a implementação dos fluxos de integração. A forma com que pretende-se garantir essa qualidade é disponibilizando interfaces simples e intuitivas para os diferentes tipos de elementos visuais.

Mostra-se na Figura 4 um exemplo de interface de configuração para mensagens, para as quais o framework Apache Camel oferece diferentes opções de protocolos pelos quais a mensagem deve ser recebida ou enviada, sendo assim as diferentes configurações para cada tipo de protocolo devem ser contempladas, como pode ser visto na Figura 7.

Um outro tipo de elemento EIP que pode-se observar é o chamado “Content Based Router” que atua como roteador de mensagens por meio de diferentes condições. Neste caso as configurações do elemento são, na verdade, as condições impostas para cada uma das diferentes opções de roteamento.

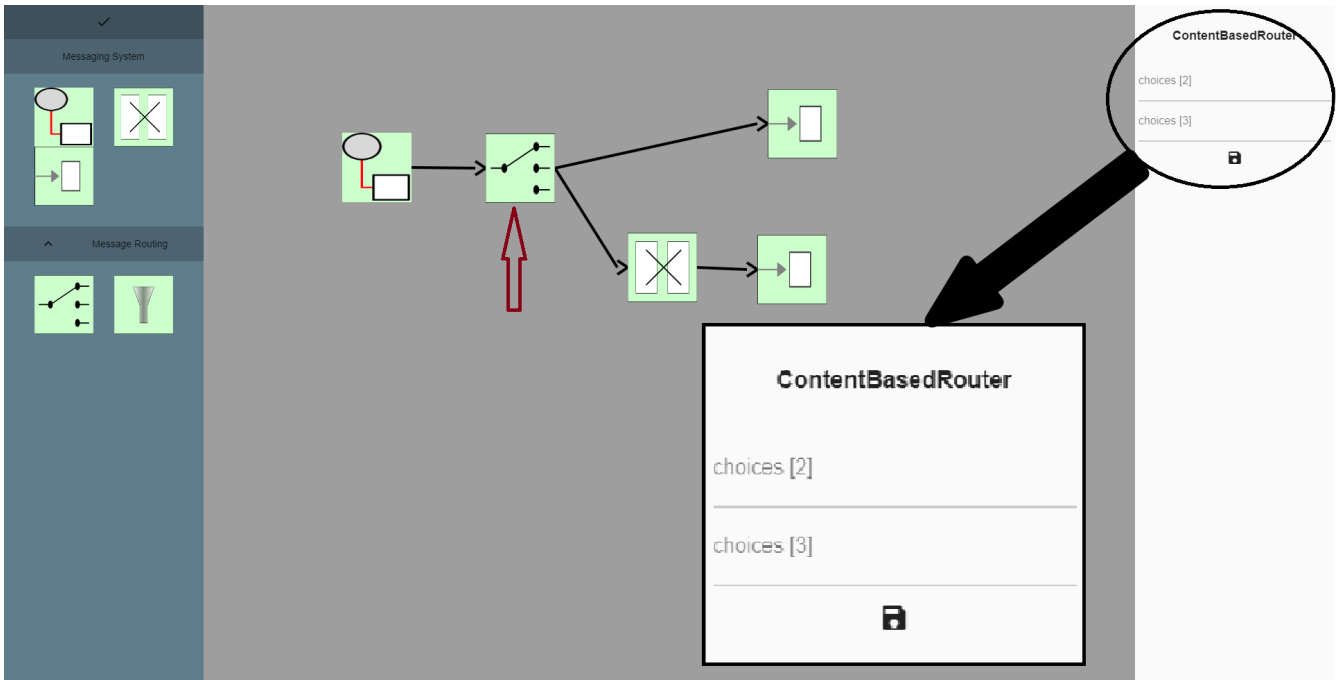


Figura 6: Exemplo do painel de configuração para o elemento Content Based Router. A Figura 7 mostra outro tipo de painel de configuração.

A Figura 6 mostra o painel de configuração do elemento Content Based Router (indicado com uma seta vermelha) inserido no exemplo. A interface reconhece a presença de duas possíveis escolhas para roteamento e gera dois inputs para inserção das respectivas condições.

A seguir são apresentadas outras interfaces de configuração de elementos EIP.

Message	Message	Message
<p>protocol <u>ftp</u> ↓</p> <p>host</p> <hr/> <p>port</p> <hr/> <p>filename</p>	<p>protocol <u>http</u> ↓</p> <p>hostname</p> <hr/> <p>port</p>	<p>protocol <u>direct</u> ↓</p> <p>name</p>

Figura 7: Diferentes apresentações do painel de configuração para diferentes protocolos para mensagens.

5.5 Back-End

Representado no lado direito da imagem. É formado por 3 blocos distintos, que serão detalhados a seguir.

5.5.1 Servidor Web

Atua como o controlador das requisições geradas pelo cliente na interface, ativando os outros componentes e enviando as respectivas respostas.

5.5.2 Sistema de Gerenciamento de Usuários e Projetos

Sistema responsável pela criação e manutenção de usuários e dados relacionados a eles. Além disso, também é responsável pelo sistema de armazenamento dos projetos salvos.

Atua em conjunto com o banco de dados para acessar e persistir os dados necessários.

Os sistemas secundários do projeto (que não envolvem a geração de código diretamente) interagem com esse sistema.

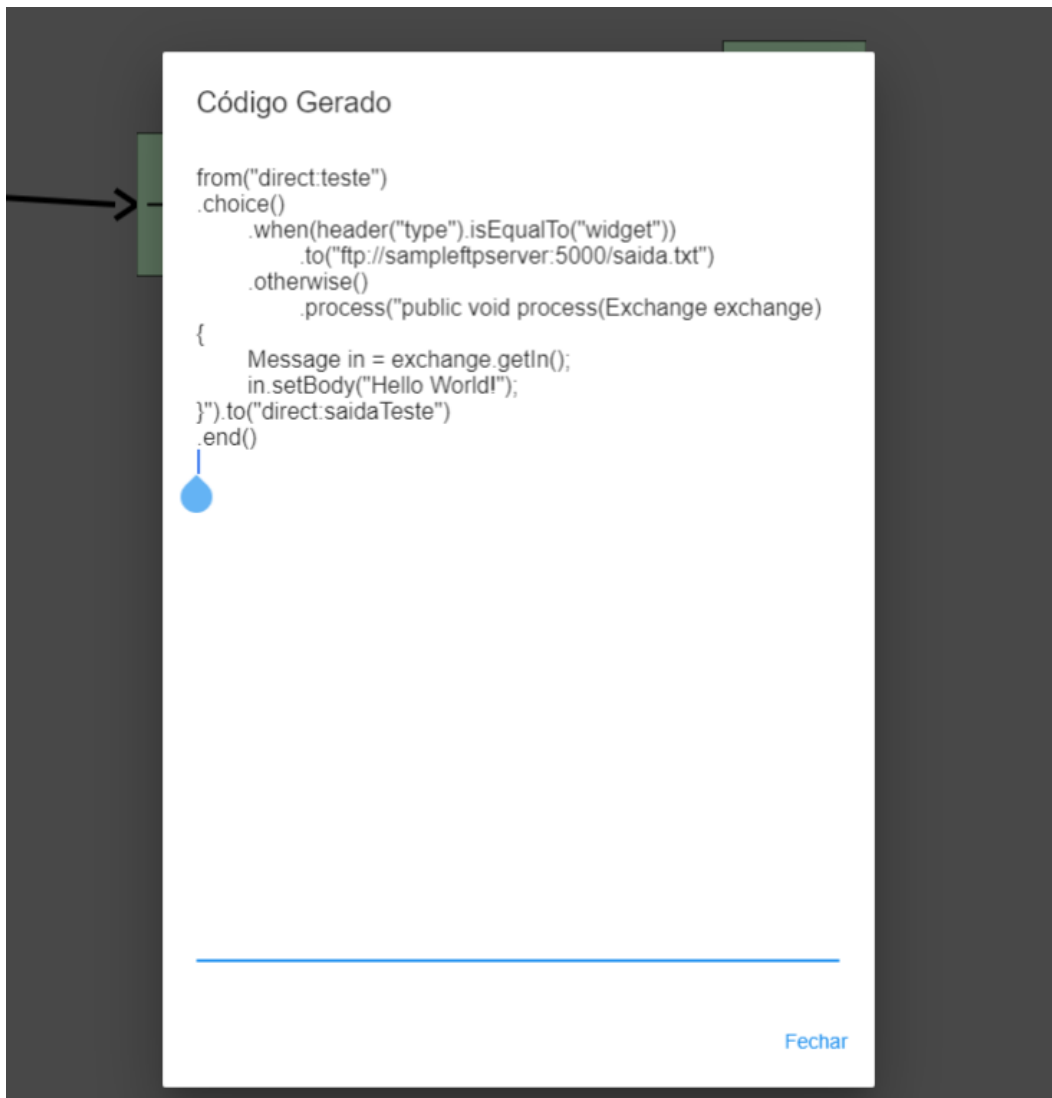
5.5.3 Sistema de Geração de Código

Sistema que implementa a parte de geração e verificação de códigos, utilizando conceitos de compiladores.

Interage com a interface durante o processo de desenvolvimento dos fluxos de integração propriamente ditos.

É capaz de interpretar os dados do diagrama gerados pela interface e efetuar os processamentos necessários de forma independente.

Mais detalhes sobre o sistema de geração de código podem ser obtidos a seguir.



```
Código Gerado

from("direct:teste")
.choice()
  .when(header("type").isEqualTo("widget"))
    .to("ftp://sampleftpserver:5000/saida.txt")
  .otherwise()
    .process("public void process(Exchange exchange)
{
  Message in = exchange.getIn();
  in.setBody("Hello World!");
}").to("direct:saidaTeste")
.end()
```

Fechar

Figura 8: Exemplo de código gerado a partir do fluxo de integração de exemplo.

6 IMPLEMENTAÇÃO DO PROJETO

Neste capítulo são apresentados, de forma detalhada, os processos de implementação das diferentes partes deste projeto, abordando questões técnicas e decisões de projeto tomadas.

6.1 Front-End

Desenvolvido utilizando o framework Flutter, consiste em uma aplicação WEB de página única (Single Page Application - SPA) que pode ser executada, de forma nativa, em diferentes plataformas.

O código é baseado em elementos denominados widgets, representando os elementos visuais da interface e suas separações por funcionalidade.

Trata-se de uma interface editável e dinâmica, de forma a permitir que os blocos representantes dos elementos EIP possam ser dispostos no diagrama e, posteriormente, editados de forma independente.

Para sua implementação dividiu-se a interface de geração de diagramas em três partes, como mostrado na Figura 4, sendo elas: Painel de seleção de elementos EIP (esquerda), canvas principal (centro) e painel de configuração dos elementos EIP (direita).

6.1.1 Desenvolvimento da Interface

Nas seções anteriores mostram-se imagens do protótipo de interface utilizado durante o processo de elaboração e início de desenvolvimento do projeto.

Nota-se, porém, que neste momento não haviam sido levados em consideração aspectos de design e interação com o usuário.

Em conjunto com o time de desenvolvimento da Opus foram propostas ideias e esboços sobre possíveis implementações visuais para o front-end.

A seguir são exibidas algumas das telas da aplicação seguindo tais propostas, servindo apenas para exemplificar as alterações realizadas. Versões ampliadas das Figuras 9 e 10 podem ser encontradas no Apêndice A - Imagens da Interface da Aplicação.

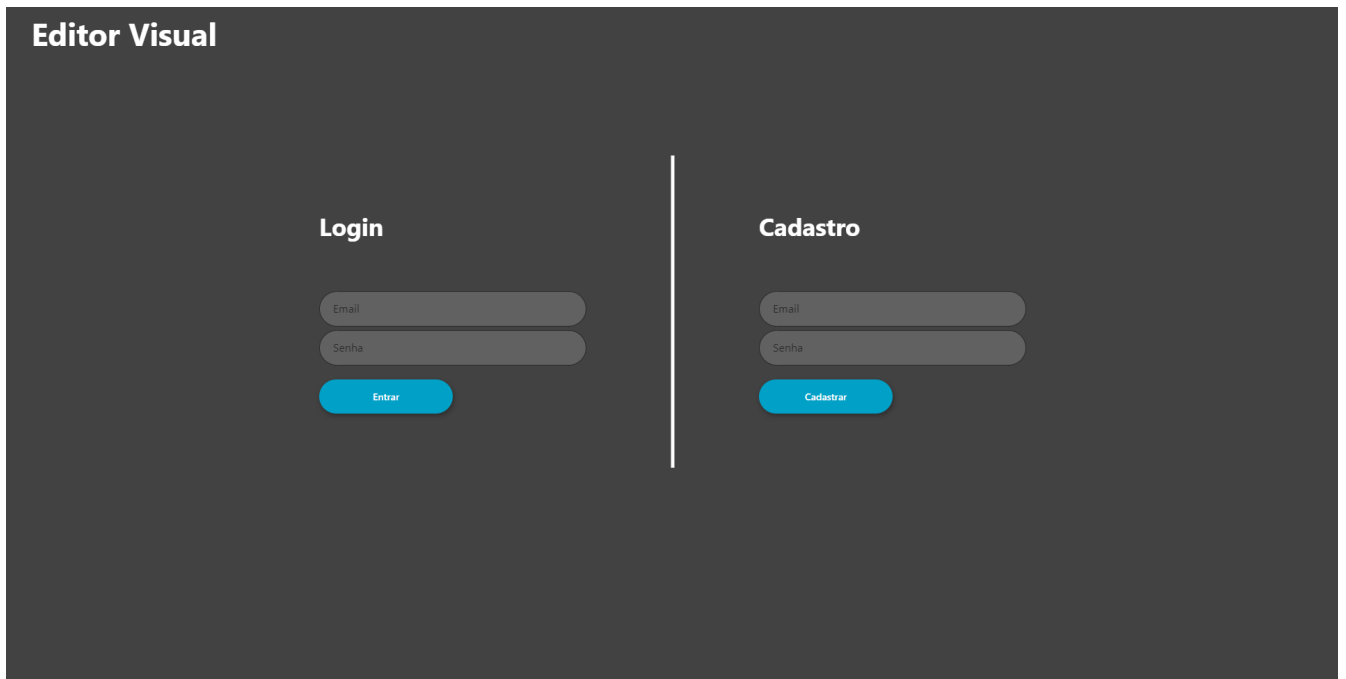


Figura 9: Tela de Cadastro e Login no Sistema

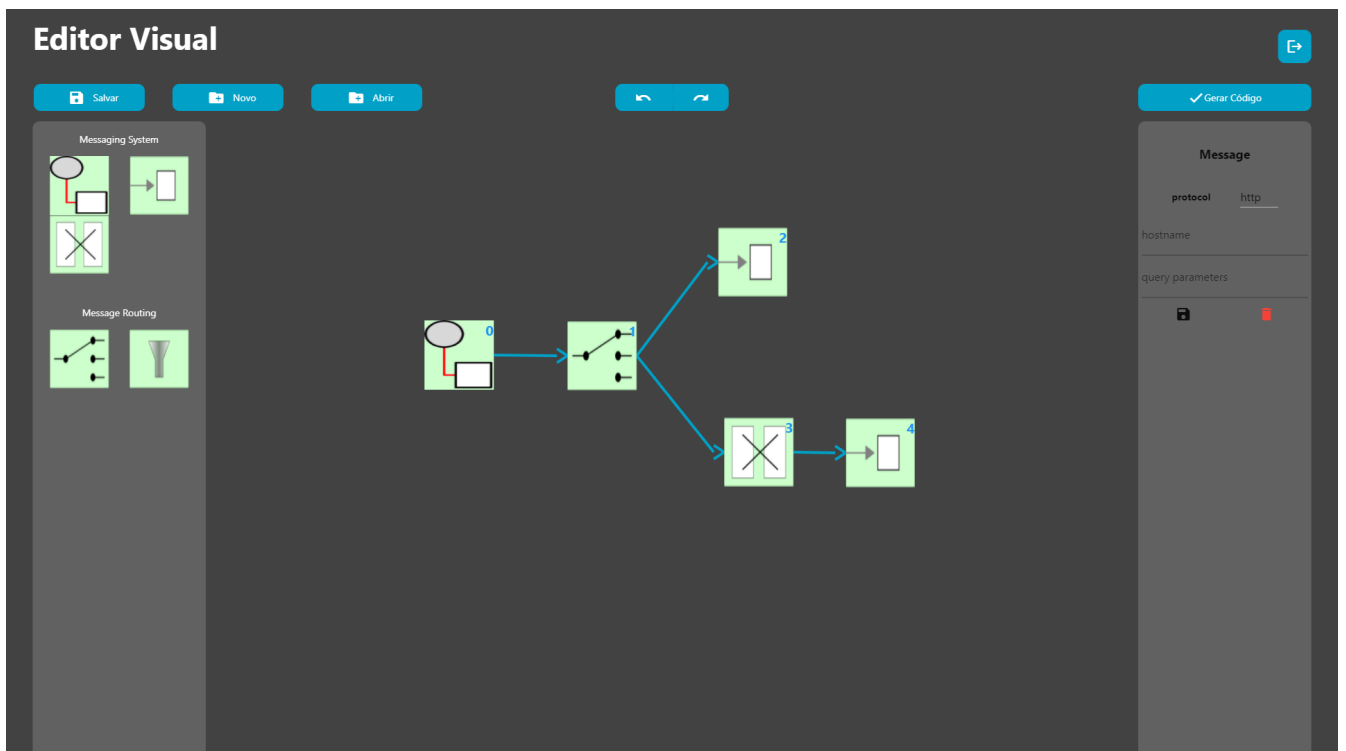


Figura 10: Tela base da aplicação, contendo o canvas principal com o qual o usuário pode interagir

6.1.2 Representação dos Elementos EIP

Este projeto foi elaborado tendo em mente a possibilidade de expansão e modificação dos elementos EIP contemplados pelo editor, além de possibilitar a utilização de outros tipos de elementos para outras finalidades, como por exemplo para programação visual com linguagens de propósito geral.

Sendo assim, cada elemento EIP deve ser representado por uma classe contendo informações necessárias para sua utilização no sistema, sendo elas:

1. Tipo e grupo, definindo qual o elemento EIP representado e a qual classe de elementos pertence, respectivamente.
2. Configurações do componente, corresponde as possíveis configurações deste elemento, podendo estar estruturado da forma mais conveniente para o elemento em questão.
3. Widget do componente, responsável por sua representação visual.
4. Método para atualização de configurações, utilizado pelo sistema quando o usuário altera as informações de um determinado componente instanciado no diagrama. Interage diretamente com as configurações do componente do item 2.
5. Método para geração do widget de configuração do elemento a ser renderizado no painel de configuração de componentes. Deve ser também baseado nas configurações do componente do item 2.
6. Ícone a ser mostrado como representação do elemento EIP.

Desta forma, a classe representante do componente é pode ser consumida de forma dinâmica pelo diagrama, de forma que diferentes elementos possam ter diferentes representações e configurações, bastando implementar os métodos necessários.

6.1.3 Elementos Arrastáveis

Trata-se de um widget elaborado com as ferramentas do flutter de tal forma que possa ser arrastado e posicionado no canvas principal. Utiliza um sistema de detecção de posição do mouse para atualizar, em tempo real, a posição do elemento na tela.

Todos os componentes EIP são mapeados em Elementos Arrastáveis quando instanciados, utilizando as propriedades contidas na classe representante do componente.

Desta maneira, um Elemento Arrestável representa um componente EIP instanciado no diagrama, contendo as suas informações de estado (neste caso as configurações específicas do componente).

6.1.4 Canvas Principal

É o componente central do Diagrama de Edição de Fluxos de Integração, tendo como função possibilitar a disposição de diferentes widgets dentro de sua área editável.

Implementou-se este widget utilizando um sistema de “stack”(pilha) disponibilizado pelo Flutter, permitindo que diferentes widgets possam ser colocados uns sobre os outros.

A estrutura da pilha que constitui o canvas principal contém em sua base um widget do tipo “CustomPaint” que permite a geração das setas para representação de conexões entre os componentes no diagrama (descrito em mais detalhes na subseção seguinte). Os Elementos Arrastáveis descritos anteriormente constituem os possíveis outros elementos da pilha.

6.1.5 Interatividade dos Componentes do Diagrama

A representação dos componentes como Elementos Arrastáveis adiciona as capacidades básicas de interatividade por parte do usuário, tornando possível que sejam arrastados, interconectados e selecionados para edição.

Tais capacidades são obtidas por meio da orquestração dos Elementos Arrastáveis pela aplicação, de forma que se comunica com os painéis laterais para a instanciação de componentes no diagrama e configuração de elementos existentes.

As representação das conexões entre os componentes do diagrama são realizadas com setas. Estas são geradas automaticamente e desenhadas no canvas principal, por meio da utilização do widget “CustomPaint” presente em sua base, quando as conexões são feitas pelo usuário.

Além disso, o projeto apresenta as funcionalidades de desfazer e refazer ações, podendo retomar qualquer estado do diagrama a partir do estado atual.

Estas foram implementadas com base em um sistema com duas pilhas. Em uma delas são armazenados os estados anteriores do diagrama, utilizado para a função de desfazer ações. Na outra são armazenados os estados futuros, aplicando-se ao caso em que o usuário tenha desfeito um número arbitrário de ações sem ter realizado nenhuma

alteração no diagrama, possibilitando refazer tais ações.

6.2 Back-End

O back-end da aplicação foi desenvolvido em Python e é dividido em três partes principais: Servidor WEB, sistema de gerenciamento de usuários e projetos e sistema de geração de código.

6.2.1 Servidor Web

Implementa os endpoints HTTP da API REST utilizada para a comunicação com o front-end, Utilizando-se da framework Flask (disponível como uma biblioteca Python), que permite sua configuração de forma simples e eficiente.

Toda a comunicação entre front e back-end é realizada por meio de requisições e respostas HTTP, regidas pela API disponibilizada pela aplicação.

As requisições são tratadas pelo servidor, recebendo os dados e executando as funcionalidades necessárias para cada tipo de requisição, comunicando-se com o sistema de gerenciamento de usuários e de geração de código.

6.2.2 Sistema de Gerenciamento de Usuários e Projetos

Permite o cadastro e login de usuários no sistema e a associação de projetos aos usuários.

Utiliza-se de um banco de dados relacional (implementado utilizando postgresql) para a modelagem e realização da persistência dos dados.

6.2.2.1 Estrutura do Banco de Dados

Trata-se de uma estrutura bastante simples, contendo apenas duas tabelas, mostradas no diagrama entidade relacionamento abaixo.

Trata-se de um relacionamento simples do tipo um para muitos (ou zero) entre usuário (representado por client) e projeto.

Cada usuário pode ser identificado por um e-mail, que deve ser único. Os campos “pass” e “salt” armazenam a senha encriptada do usuário e o valor aleatório utilizado para

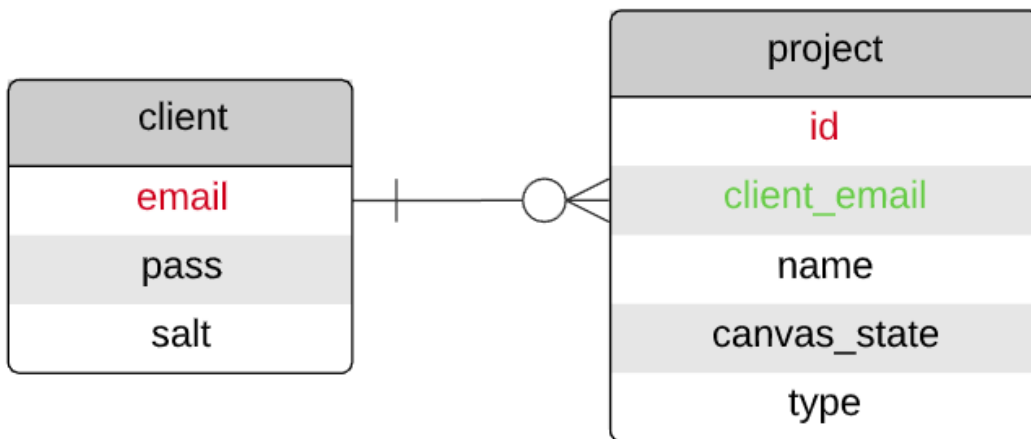


Figura 11: Diagrama entidade relacionamento utilizado para criação do banco de dados do projeto

encriptá-la.

Projetos são identificados por valores sequenciais gerados automaticamente. São associados aos usuários por meio de uma chave estrangeira referenciando o e-mail dos usuários. Apresentam ainda o nome do projeto, o estado do diagrama (armazenado em texto como estrutura JSON) e o tipo de projeto em questão.

6.2.2.2 Sistema de Cadastro e Login

Foi implementado um sistema de cadastro simples em banco de dados, utilizando, como citado anteriormente, o e-mail dos usuários para sua identificação.

Por questões de segurança, a senha dos usuários não é armazenada diretamente no banco de dados.

Foi utilizado um sistema de “Hash and Salt”, que consiste na geração de um valor hash a partir da combinação da senha original do usuário com um valor gerado aleatoriamente na hora do cadastro (salt).

O hash gerado é uma string única, que representa tal combinação de senha e salt, e irreversível, isto é, não é possível se recuperar a senha original a partir desta string.

Para a verificação no momento de login do usuário, torna-se a obter o hash da senha informada com o salt armazenado para o usuário e então compara-se o hash gerado com o armazenado.

Este tipo de implementação evita que, em caso de falhas de segurança, os usuários tenham suas senhas expostas diretamente.

Também é responsável pela criação e manutenção de sessões com a interface, implementado em conjunto com a utilização de armazenamento local do navegador (“Cookies”) no front-end. Este tipo de abordagem é interessante para melhorar a utilização do sistema por parte do usuário, mantendo sua sessão ativa (autenticada) durante a utilização da aplicação.

6.2.3 Sistema de Geração de Código

Este sistema é separado em três partes: parser, gerador de código e gerador de projetos.

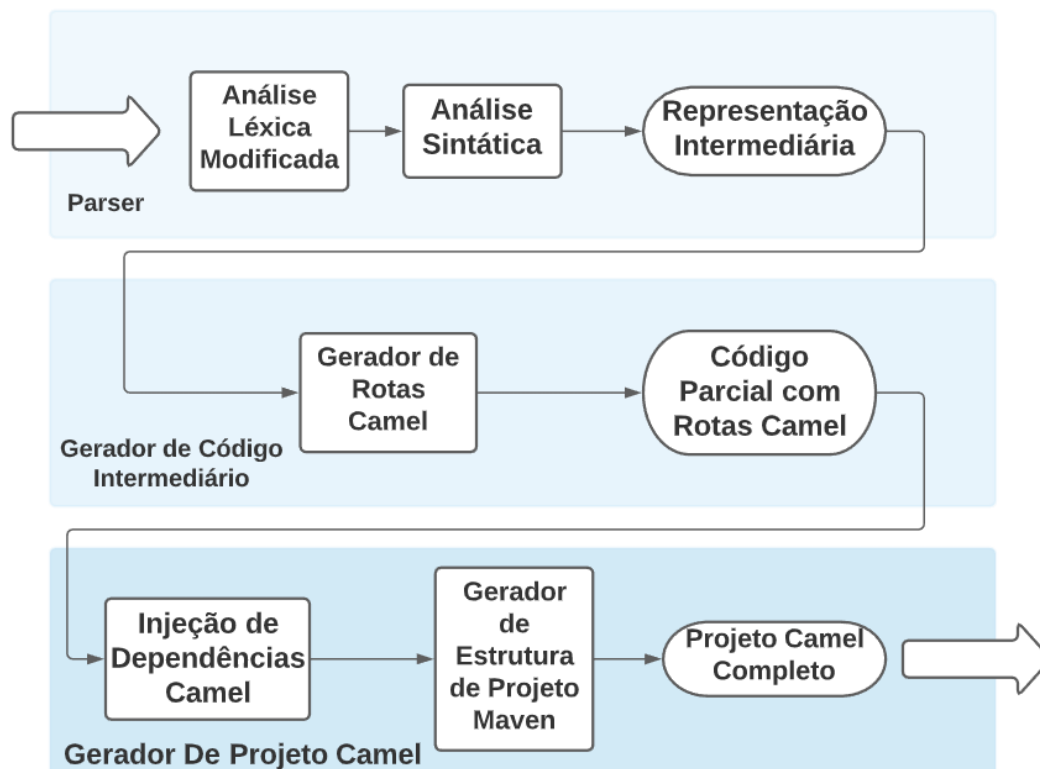


Figura 12: Diagrama dos componentes do sistema de geração de código para sistemas de integração Apache Camel

Todo o sistema de geração de código para o fluxos de integração Apache Camel foi desenvolvido integralmente, sem auxílio de bibliotecas e ferramentas para geração dos analisadores léxicos, sintáticos e para geração de código.

Posteriormente, neste relatório, apresenta-se a implementação do sistema de geração de código para uma linguagem de propósito geral. Nesta ocasião são utilizadas ferramentas adicionais para facilitar a sua implementação.

6.2.3.1 Parser

Responsável transformação da representação visual do projeto em uma representação intermediária que pode ser utilizada pelo gerador de código. Verifica se os elementos visuais do diagrama estão corretamente interligados, através da análise sintática dos elementos do diagrama (na forma de Tokens).

Para tal pode-se utilizar de uma gramática que represente a linguagem e, a partir desta, obtém-se uma forma simples de implementação de um parser

Uma forma de representação de gramáticas é a Backus-Naur Form (BNF), na qual pode-se representar os terminais, não terminais e as regras de produção de uma linguagem específica. Existem ainda diversas variantes e expansões da notação BNF.

A notação utilizada para descrever a gramática desenvolvida para os padrões EIP deste projeto é baseada na versão estendida da BNF (EBNF), contendo operadores úteis, como o operador de repetição “{}”, que representa a ocorrência de nenhuma ou qualquer quantidade do elemento ao qual o operador se aplica, além da representação de elementos opcionais pelo operador “|”.

No contexto de compiladores, o parser recebe tokens (diferentes elementos, palavras-chave, variáveis, etc...) em sequência, normalmente de um analisador léxico, e verifica se a forma com a qual os elementos estão sendo apresentados segue as regras gramaticais da linguagem analisada.

De fato o processo utilizado para criação do parser, a partir de uma gramática, pode ser descrito pelo algoritmo mostrado a seguir:

1. Cada regra de produção é representada como um método. As referências a esta regra são as chamadas deste método, que por sua vez segue a estrutura definida pela regra correspondente.
2. Elementos opcionais são transformados em blocos condicionais.
3. Um agrupamento com repetição se transforma em um laço.
4. Cada não terminal é consumido. Isso é feito verificando se o tipo do não terminal sendo consumido é o esperado segundo as normas da gramática e, em caso positivo, obtém o próximo token a ser analisado.

Foi construída uma gramática que representa as estruturas básicas encontradas nas rotas possíveis de serem montadas empregando padrões EIP e Apache Camel.

Nota-se que não é uma gramática completa em relação a todos os possíveis elementos EIP disponíveis, mas sim apenas para aqueles que foram incorporados ao projeto até o momento de sua implementação.

A gramática utilizada apresenta as seguintes regras de derivação:

1. route := “start”transformation
2. transformation := { “simple” } (“end”|fork)
3. fork := “onetomany”transformation

Os elementos representados entre aspas são os terminais, o restante são os não terminais da linguagem. Nota-se ainda que a linguagem representada pela gramática não utiliza os elementos EIP em si, mas sim os tipos de funcionalidade.

A lista a seguir apresenta as descrições dos terminais utilizados:

- “start” representa elementos de mensagem dentro dos padrões EIP, sendo a origem das rotas.
- “end” representa os “Message Endpoints” do EIP, sendo o destinatário da mensagem processada pela rota.
- “simple” representa os elementos EIP que fazem transformações simples na mensagem e no caminho desta, sendo eles filtros, processadores de mensagens, tradutores, entre outros.
- “onetomany” representa os elementos de roteamento do EIP, capaz de direcionar as mensagens para caminhos diferentes baseado em condições arbitrárias.

Esta gramática pode ser completada e estendida conforme a necessidade de inserção de novos elementos EIP, servindo como base para a realização da linguagem visual que representa os elementos EIP.

6.2.3.2 Gerador de Código

Pode-se observar na Figura 8 o resultado do gerador de código implementado para o protótipo utilizando o fluxo de integração mostrado na Figura 4. O código gerado neste momento do projeto representa as rotas do Apache Camel e já se encontram na forma esperada para utilização com tal framework.

Sua implementação se baseia no fato de que a representação dos dados do diagrama pode ser interpretada como uma forma de árvore sintática abstrata, na qual os elementos se conectam à elementos filhos. Sendo assim pode-se percorrê-la de forma recursiva para geração do código em Java.

O algoritmo utilizado para sua implementação efetua uma busca em profundidade recursiva partindo dos nós que representam o início das possíveis rotas.

Cada diferente elemento EIP apresenta sua própria forma de geração de código e regras a serem utilizadas para tal, isso inclui as funcionalidades, opções e configurações de cada elemento. Este fato é tratado de forma dinâmica pelo sistema para cada elemento EIP, utilizando a o método “parse” implementado na classe representante do componente para o back-end.

Ao terminar a busca em profundidade dos elementos do diagrama, o código da rota equivalente é gerado.

6.2.3.3 Gerador de Projeto

Após a verificação sintática do diagrama e da geração dos códigos das rotas equivalentes, pode-se finalmente gerar o projeto Camel completo.

Para isso gera-se uma estrutura de pastas e arquivos básico para projetos Camel com base nas estruturas utilizadas pelo Apache Maven. Todas as dependências necessárias e os códigos das rotas gerados são adicionadas nos arquivos necessários de forma automática.

Tais pastas e arquivos são compactados e disponibilizados para download quando requisitados pelo usuário.

7 EXPANSÃO DO PROJETO

Neste capítulo documenta-se a proposta e processo de implementação da expansão do projeto para diferentes tipos de linguagens e integração dos sistemas com outras tecnologias, em especial o framework LLVM que auxilia nos processos de desenvolvimento de compiladores, otimização e geração de código.

Adicionalmente, no Apêndice B - Gerador de Código BASIC, apresenta-se uma terceira implementação, envolvendo a linguagem Dartmouth BASIC.

7.1 Proposta

Afim de mostrar a facilidade de expansão das funcionalidades do projeto desenvolvido, pretende-se possibilitar a criação e edição de projetos utilizando uma linguagem de programação de propósito geral, sendo capaz de gerar, ao final do processo de geração de código, a representação textual na linguagem Kaleidoscope, seguindo suas regras de formação.

O projeto deve ser também capaz de realizar a transformação da representação na linguagem Kaleidoscope para a representação intermediária LLVM (IR LLVM). Com isto, pode-se gerar programas executáveis para diferentes arquiteturas e plataformas de forma bastante simples, com as ferramentas LLVM adequadas, garantindo maior flexibilidade ao projeto.

A linguagem Kaleidoscope que se propõe implementar é bastante simples que une funcionalidades de linguagens funcionais e imperativas. Foi escolhida pois é a linguagem utilizada como base para a realização do tutorial de apresentação das funcionalidades LLVM [28].

Para realização deste projeto, incorpora-se uma implementação de uso aberto do código apresentado no tutorial citado [28] portado para a linguagem Python [29] - utilizada neste projeto.

7.2 Alterações no Front-End

De forma geral, poucas alterações foram feitas na interface para a implementação desta expansão, dada a forma com a qual foram projetadas as estruturas do código e scripts de automação para a integração automática de novos componentes ao projeto.

Fez-se necessário apenas a inclusão de sistemas de seleção de tipos de projetos distintos no momento da criação de projetos, além de pequenos ajustes para viabilizar a correta visualização dos novos elementos no painel de seleção.

Os novos componentes visuais foram adicionados seguindo a estrutura proposta para tal, resultado em baixa dificuldade e agilidade em suas implementações.

A interface proposta para os novos tipos de projeto contém componentes visuais dos seguintes tipos:

- Expression:

Contém componentes de expressões - aritméticas ou utilizadas como retorno para funções - e de atribuição de valor a variáveis.

- Conditional:

Componentes de laços (For) e para realização de desvio condicional (If, Then, Else).

- Def:

Agrupar os elementos de definição (Def) e chamada de funções (Call).

7.3 Alterações no Back-End

A Figura 13 mostra os componentes envolvidos na realização da geração de código para proposta de extensão deste projeto.

Os detalhes de sua implementação são discutidos nas subseções a seguir, destacando as alterações e adaptações realizadas.

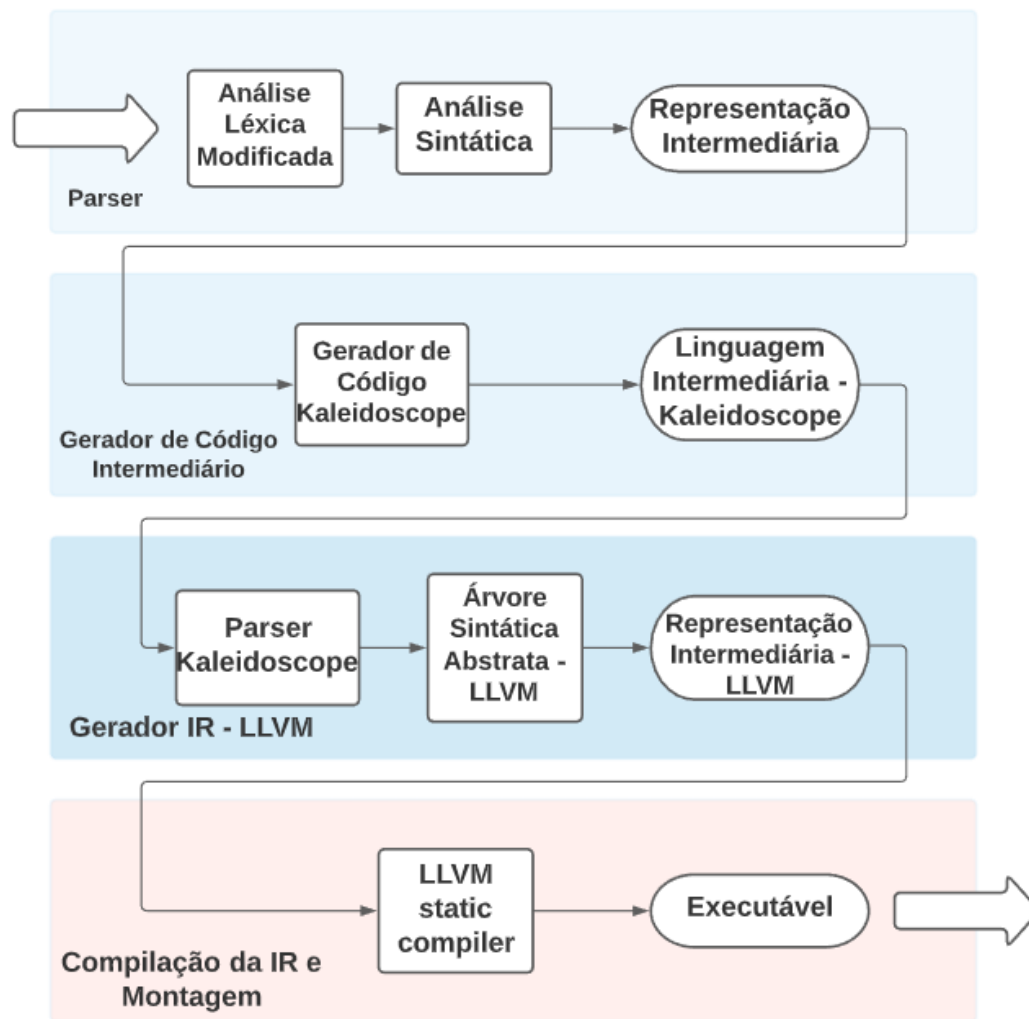


Figura 13: Diagrama da arquitetura do sistema de geração de código para a linguagem Kaleidoscope

7.3.1 Parser

Responsável pela interpretação das informações do diagrama gerado na interface gráfica e transformação para uma primeira forma de representação intermediária - AST - como era realizado no sistema de geração de código Java - Camel.

Utiliza-se como base as regras de formação da linguagem Kaleidoscope (representada em EBNF):

```

toplevel := definition | expression | ";"
definition := "def", prototype, expression
expression := primary, binoprhs
binoprhs := {[binop], primary}
primary := identifierexpr

```

```

:= numberexpr
:= parenexpr
:= ifexpr
:= forexpr
:= varexpr
identifierepr := identifier
                := identifier , "(" , {expression} , ")"
numberexpr := number
parenexpr := "(" , expression , ")"
ifexpr := "if", expression, "then", expression, "else", expression
forexpr := "for", identifier, "=", expr, ",", expr, [",", expr] "in", expr
varexpr := "var", identifier, ["=", expr] ,
          {"", identifier, ["=", expr]}, "in", expr

```

Ao final do processamento do Parser, obtém-se a representação do diagrama elaborado na forma de uma AST, sendo possível dar continuidade ao processo de geração de código.

7.3.2 Gerador de Código Kaleidoscope

Com a AST gerada, este componente realiza a transformação da AST para representação na linguagem Kaleidoscope definida no item anterior.

Este processo é realizado percorrendo-se a estrutura da AST de forma recursiva, gerando trechos de código que são concatenados.

O resultado do fim do processamento deste componente é um arquivo texto contendo a representação da lógica representada visualmente na linguagem Kaleidoscope.

7.3.3 Compilador Kaleidoscope para IR LLVM

Este componente utiliza ferramentas da biblioteca `llvmlite` para Python de forma a implementar um módulo de geração de uma segunda representação intermediária do código fonte fornecido, transformando-a em representação intermediária LLVM.

Este módulo é constituído por um parser da linguagem Kaleidoscope - desta vez em sua representação textual convencional - incluindo também o processo de análise léxica para esta linguagem.

Utiliza elementos da AST próprios do LLVM que permitem, ao final do processo, a

geração da representação intermediária LLVM.

O resultado desta etapa do processo é fornecido ao usuário final, ao baixar o projeto. Nota-se que este não é um arquivo binário executável, mas sim um documento de texto com a IR LLVM, que pode ser compilado e montado para a geração do executável correspondente. Este processo é descrito a seguir.

7.3.4 Compilação da IR e Montagem

O último módulo da estrutura de geração de código Kaleidoscope é, na realidade, um processo que deve ser realizado pelo usuário em posse da representação intermediária gerada pelo sistema.

Pode-se utilizar o compilador estático LLVM para realizar esta tarefa, gerando código para a arquitetura e plataforma definida nos cabeçalhos do arquivo gerado.

7.4 Exemplo de Funcionamento

Para demonstrar as funcionalidades da expansão do projeto, propõe-se a implementação de uma função que obtém os números da sequência de Fibonacci de forma iterativa.

Mostra-se, na Figura 14 o diagrama que representa o código da função citada, além de três realizações de chamada para a função, com diferentes argumentos.

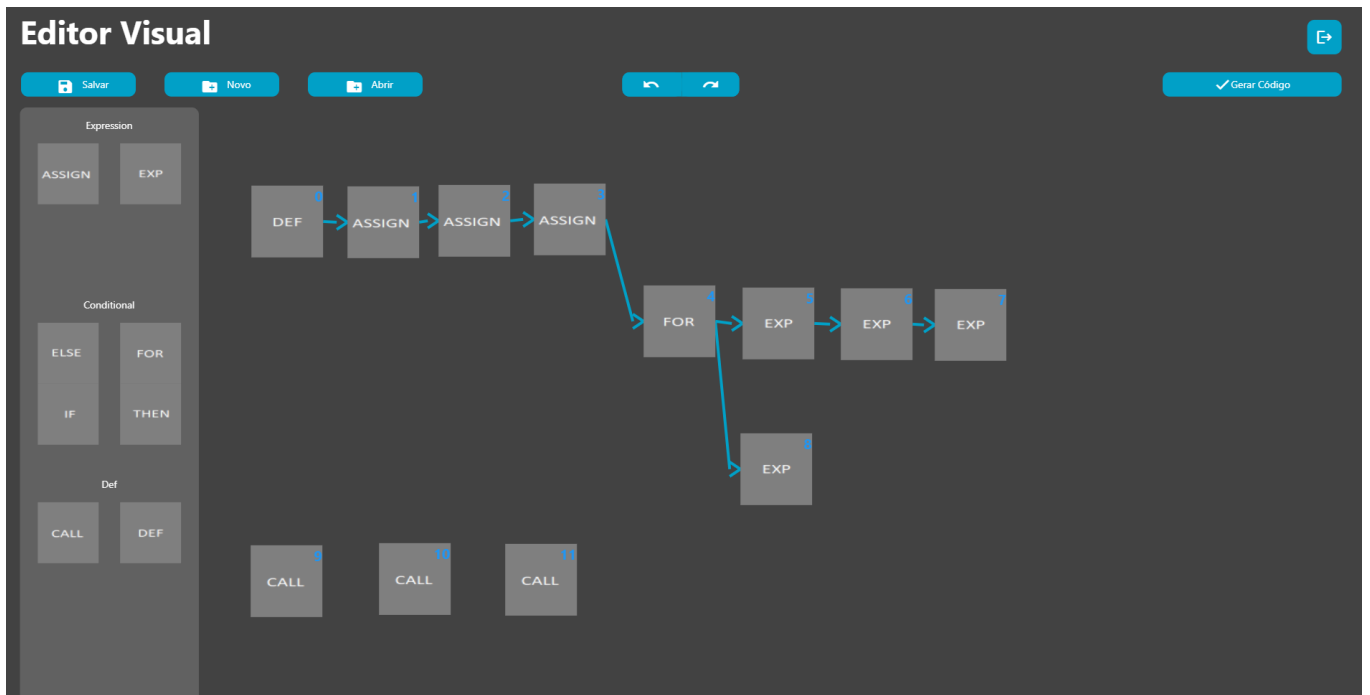


Figura 14: Diagrama para a implementação da função de obtenção de números da sequência de Fibonacci. Figura em maior escala presente no Apêndice A.

Ao realizar o processo de geração de código, obtém-se o seguinte resultado, mostrado na Figura 15.

Código Gerado

code

```
def fib(x)
  var a = 1 in
  var b = 1 in
  var c = 0 in
  (for i = 3, i < x in
  (c = a + b):
  (a = b):
  (b = c)):
  (b)
fib(4)

fib(5)

fib(10)
```

result

```
3.0
5.0
55.0
```


[Download Project](#) 

Figura 15: Exemplo do código Kaleidoscope gerado e resultado da execução das chamadas da função “fib” para os valores 4, 5 e 10.

Repare que, além do código Kaleidoscope gerado, mostra-se também (na região des-

tacada), os resultados da execução da simulação do código gerado, utilizando-se das ferramentas LLVM. Neste caso representando os valores dos elementos da sequência de Fibonacci no índices 4, 5 e 10.

8 TESTES

Neste capítulo são descritas as abordagens para realização dos testes da aplicação desenvolvida, realizados no front e back-end.

Para realização destes testes foram utilizadas bibliotecas e funcionalidades específicas para testes em projetos Python e Flutter, como PyTest, Flutter Test e Flutter Driver.

8.1 Testes Unitários

Os testes unitários são realizados contra classes, funções ou métodos específicos, sendo responsáveis pela verificação do correto funcionamento destas unidades.

Testes unitários podem ser realizados de maneira simples e direta, em especial com auxílio de bibliotecas e frameworks especializados neste tipo de atividade.

Os teste unitários do back-end foram realizados com a ferramenta PyTest, testando todas os métodos responsáveis pela realização de ações de controle de cadastro, criação e edição de projetos e geração de código.

Para o front-end utiliza-se as ferramentas de teste nativas do Flutter, sendo possível o teste tando de funções quanto dos Widgets específicos do projeto.

Estes testes são parte importante do desenvolvimento de software, porém não são suficientes, pois não possibilitam a verificação do correto funcionamento do sistema como um todo.

8.2 Testes de Integração

Os testes de integração são realizados de forma a verificar a correta execução de funcionalidades da aplicação que necessitam interagir com outras partes do código ou com dependências externas.

Foram realizados testes de integração do back-end em relação a funcionalidades que interagem com banco de dados e com o sistema de geração de código da aplicação.

Realizaram-se testes quanto a comunicação entre a interface e o servidor, para garantir o correto funcionamento do sistema como um todo.

8.3 Testes de Geração de Código

Uma parte importante deste projeto é a parte de geração de automática de código java e de projetos Apache Camel.

Desta maneira, foram realizados testes para verificação do correto funcionamento do código e projeto gerados pela aplicação para diferentes projetos de integração e diagramas.

8.4 Simulação de Uso

Como forma de teste e prova de conceito, foi proposta uma simulação de uso da aplicação em um cenário que simula um caso em que as convenções e ferramentas oferecidas pelos padrões EIP e pela ferramenta desenvolvida facilitam o desenvolvimento de uma solução.

8.4.1 Explicação da Simulação

Para este exemplo a simulação adotada apresenta dois sistemas distintos que, inicialmente, não se comunicam e devem passar a se comunicar. A seguir são apresentados mais detalhes sobre o cenário e os sistemas envolvidos. A figura 16 abaixo apresenta a visão de estrutura a ser simulada.

O sistema 1 representa um sistema legado, ou similar, que gera notificações e mensagens conforme ocorrem eventos no sistema e os armazena em forma de arquivos de texto. Pressupõe-se que este sistema, como se encontra, não tem capacidade de se comunicar com o sistema 2 pois não possui uma interface de comunicação http implementada.

O sistema 2 é um servidor WEB capaz de receber requisições HTTP contendo informações sobre mensagens geradas pelo sistema 1, classifica-las e obter informações e estatísticas úteis a partir delas. Este sistema inicialmente não consegue se comunicar com o sistema 1.

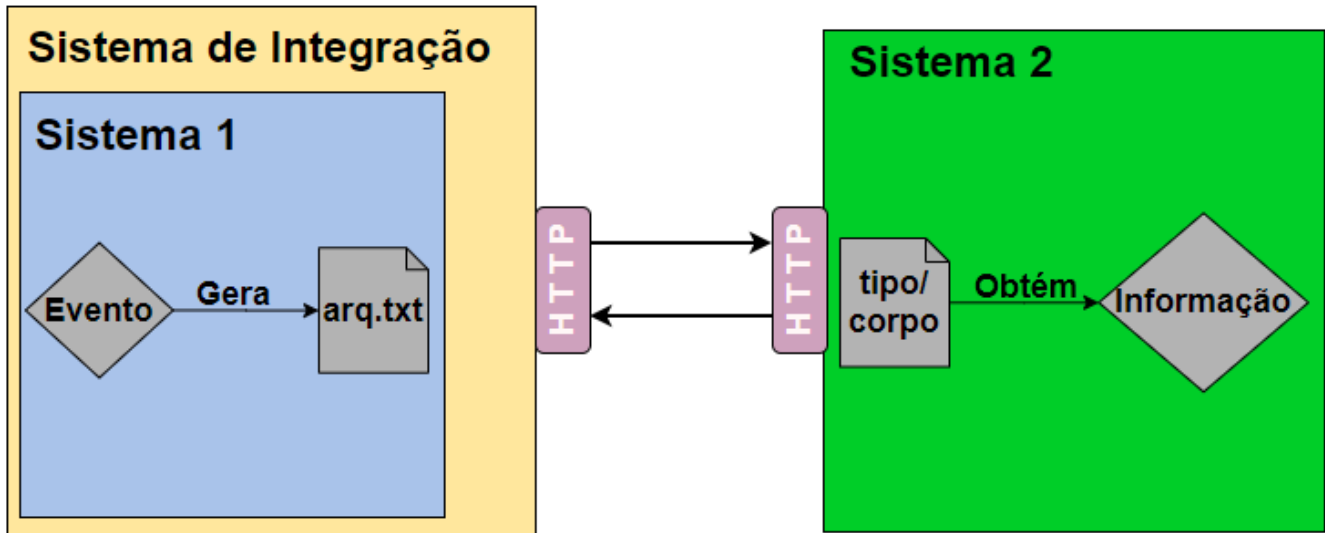


Figura 16: Diagrama da estrutura da situação a ser simulada utilizando a aplicação para auxiliar a geração de código de integração para os sistemas 1 e 2.

Ambos os sistemas foram implementados de maneira bastante simples com Python. O sistema 1 apenas gera arquivos de determinados tipos de forma aleatória em determinados intervalos de tempo. O sistema 2 apresenta um simples servidor http com o qual pode-se verificar se as mensagens estão sendo corretamente enviadas pelo sistema de integração.

8.4.2 Especificação das Estruturas dos Arquivos e Forma de Comunicação

Para que se possa projetar um sistema de integração, deve-se entender e estabelecer as formas com as quais os sistemas irão se comunicar e quais as estruturas de arquivos e mensagens que podem ser tratadas e devem ser encaminhadas.

Para fins de exemplo definem-se três classes de tipos de arquivos: Notificações de erros (Errors), notificação de mensagens (Message) e informações diversas.

As regras de comunicação com o sistema 2 são as seguintes:

1. Os dados erro devem ser enviados para o sistema 2 utilizando o parâmetro de tipo como sendo "Error" e o conteúdo da mensagem de erro
2. Os dados de mensagem devem ser apenas notificados para o sistema 2 com seu tipo de "Message"
3. Outros tipos de mensagem devem ser enviados com tipo "Unknown" e ao corpo da mensagem deve ser acrescido o trecho: "Mensagem não identificada"

8.4.3 Proposta de Solução

A proposta de solução para este sistema de integração utilizando padrões EIP pode ser representada pelo diagrama apresentado na figura 17.

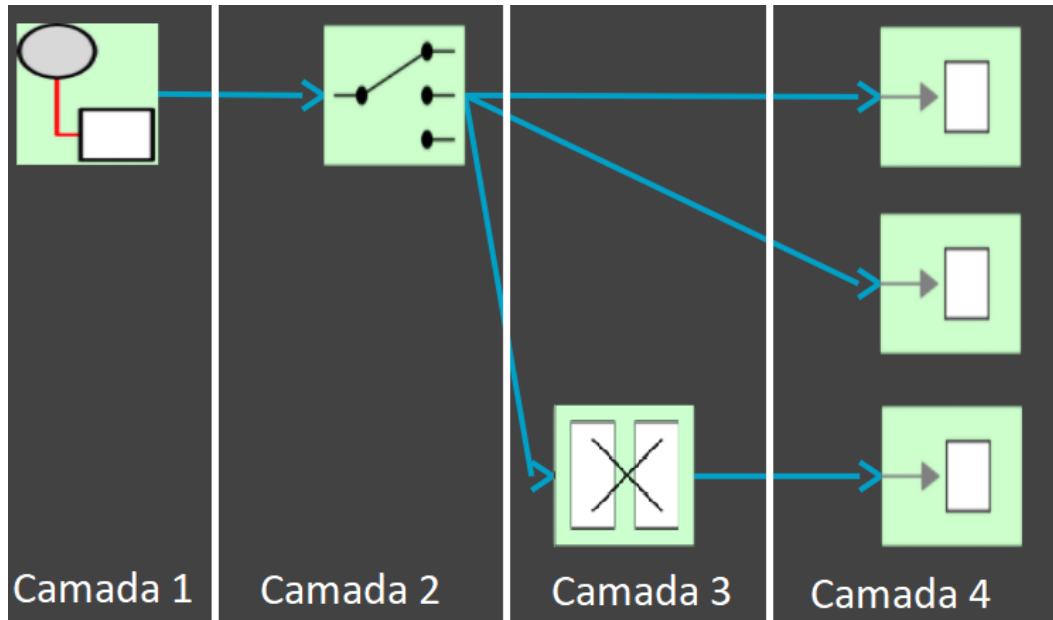


Figura 17: Diagrama utilizando os elementos EIP disponíveis na aplicação para a estruturação de um fluxo de integração.

O sistema de integração tem como finalidade obter os arquivos gerados pelo sistema 1, filtrá-los, processá-los e, por fim, oferecer uma interface HTTP pela qual possa ocorrer a comunicação com o sistema 2. Como sugere a figura 16 o sistema de integração é acoplado ao sistema 1 de forma a estender suas funcionalidades, em especial com a inserção do mecanismo de comunicação HTTP.

A primeira camada do diagrama contém o representante das entradas de dados. Neste exemplo os arquivos são gerados em um diretório denominado “entrada”, que devem ser inseridos no sistema de integração ao serem criados. O mecanismo implementado pelo apache Camel observa um diretório especificado e permite realizar ações quando arquivos são criados, modificados ou removidos.

A segunda camada é composta por um elemento de chaveamento de conteúdos (“Content Based Router”), responsável por filtrar os arquivos recebidos conforme as regras descritas na seção anterior. Neste elemento são configurados os tratamentos específicos para mensagens, erros e para os outros casos gerais.

A terceira camada contém um elemento de transformação de mensagens EIP denominado “Message Translator”. Este componente têm acesso ao corpo e cabeçalho do

arquivo e pode os alterar. Foi utilizado no caso de arquivos com conteúdo diferente de “Message” ou “Error”.

Por fim, a quarta camada contém os adaptadores de saída do sistema que efetuam a comunicação com o sistema 2 por meio de mensagens HTTP.

8.4.4 Geração das Rotas e de Projeto Camel

Após a configuração dos elementos inseridos no diagrama, pode-se gerar as rotas correspondentes a estrutura de integração proposta. O código Apache Camel de rotas gerado em Java pelo sistema de geração de código pode ser visto na figura 18.

```

from("file:entrada")
.choice()
  .when(bodyAs(String.class).contains("Message"))
    .toD("http://localhost:5005/?type=Message")
  .when(bodyAs(String.class).contains("Error"))
    .toD("http://localhost:5005/?type=Error&content=${body}")
  .otherwise()
    .process(new Processor() {
      public void process(Exchange exchange) {
        Message in = exchange.getIn();
        in.setBody("Mensagem nao identificada" + in.getBody(String.class));
      }
    })
    .toD("http://localhost:5005/?type=Unknown&content=${body}")
.end();

```

Figura 18: Diagrama utilizando os elementos EIP disponíveis na aplicação para a estruturação de um fluxo de integração.

Com as rotas configuradas pode-se gerar o projeto apache Camel que pode ser executado de maneira automática através de comandos Apache Maven. Para que isso seja possível o sistema deve gerar uma estrutura de diretório específica, além de realizar configurações e incluir dependências necessárias.

A figura 19 mostra a estrutura do projeto e a figura 20 mostra detalhes sobre alguns de seus arquivos. De maneira geral têm-se os seguintes elementos dentro da estrutura de projeto Apache Camel gerada:

1. **pom.xml**: Contém a configurações e dependências do projeto Montado a partir das informações de dependência obtidas na parte de geração de código

2. **LaunchApp.java**: Configurações para execução do projeto via Maven
3. **SampleRoutes.java**: Arquivo com a implementação do códigos de rota gerados

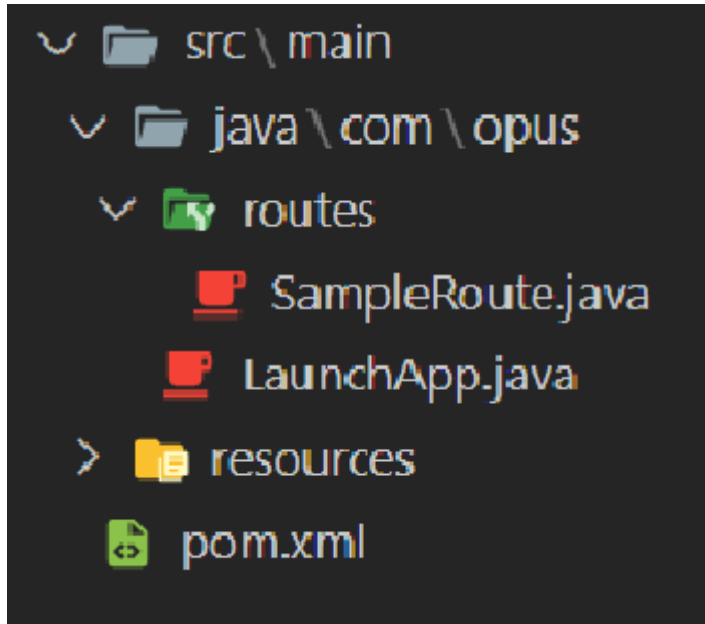


Figura 19: Exemplo da estrutura dos arquivos gerados automaticamente pela aplicação.


```

src > main > java > com > opus > routes > SampleRoute.java
1  package com.opus.routes;
2
3  import org.apache.camel.builder.RouteBuilder;
4  import org.apache.camel.Processor;
5  import org.apache.camel.Exchange;
6  import org.apache.camel.Message;
7
8  public class SampleRoute extends RouteBuilder {
9
10     @Override
11     public void configure() throws Exception {
12         from("file:entrada")
13     .choice()
14         .when(bodyAs(String.class).contains("Message"))
15             .toD("http://localhost:5005/?type=Message")
16         .when(bodyAs(String.class).contains("Error"))
17             .toD("http://localhost:5005/?type=Error&content=${body}")
18         .otherwise()
19             .process(new Processor() {
20     public void process(Exchange exchange) {
21         Message in = exchange.getIn();
22         in.setBody("Mensagem nao identificada" + in.getBody(String.class)
23     }
24     }).toD("http://localhost:5005/?type=Unknown&content=${body}")
25     .end();
26
27     }
28
29 }
30
pom.xml
pom.xml
38 </dependency>
39 <dependency>
40 <groupId>org.apache.camel</groupId>
41 <artifactId>camel-main</artifactId>
42 </dependency>
43
44 <!-- logging -->
45 <dependency>
46 <groupId>org.apache.logging.log4j</groupId>
47 <artifactId>log4j-slf4j-impl</artifactId>
48 <scope>runtime</scope>
49 <version>${log4j2-version}</version>
50 </dependency>
51
52 <!-- testing -->
53 <dependency>
54 <groupId>org.apache.camel</groupId>
55 <artifactId>camel-test</artifactId>
56 <scope>test</scope>
57 </dependency>
58
59 <dependency>
60 <groupId>org.apache.camel</groupId>
61 <artifactId>camel-http</artifactId>
62 </dependency>
63
64
65 </dependencies>
66
67 <build>
68 <defaultGoal>install</defaultGoal>
69
70 <plugins>
71 <plugin>
72 <groupId>org.apache.maven.plugins</groupId>

```

Figura 20: Exemplo de arquivo de rotas e arquivo de configuração gerados automaticamente.

Esta estrutura pode ser baixada através da aplicação e ser descompactada diretamente no diretório contendo o sistema 1. Para ser iniciado, basta a execução de dois comandos do Maven na pasta contendo o arquivo “pom.xml”, sendo eles: “mvn clean package” e “mvn exec:java -Dexec.mainClass=com.opus.LaunchApp”.

Feito isso o sistema de integração já está apto a identificar arquivos gerados pelo sistema 1 e os envia para o sistema 2 como esperado, o qual recebe os dados de forma categorizada utilizando as rotas previamente construídas.

9 CONSIDERAÇÕES FINAIS

Neste capítulo são apresentadas as conclusões finais deste trabalho, assim como as contribuições que possibilitaram sua realização, além de perspectivas de continuidade do projeto e possível implementação em ambientes reais de desenvolvimento.

9.1 Conclusões do Projeto de Formatura

O resultado deste projeto foi bastante positivo tanto em seu resultado quanto no processo de elaboração e desenvolvimento, em especial pelo formato no qual foi realizado, em conjunto com a empresa Opus Software.

Obteve-se ao final do processo de desenvolvimento, um artefato de software que possivelmente será integrado em um contexto real de desenvolvimento dentro da empresa colaboradora. Tendo sido o tema deste trabalho fruto de uma necessidade real de ferramenta de auxílio na elaboração de projetos de integração.

O ambiente proporcionado pela sua realização em conjunto com a Opus, no formato de estágio remunerado mostrou-se bastante eficiente e motivador. O contato direto com desenvolvedores, gerentes de projeto, designers e outros funcionários da empresa possibilitaram uma dinâmica de projeto e trocas de informações que não seriam possíveis em outros casos.

O processo de desenvolvimento da aplicação em si, tendo coberto os conceitos fundamentais de servidores, banco de dados, interface visual, tornou-se uma boa oportunidade para novos aprendizados de técnicas, conceitos e padrões de desenvolvimento. Em especial a utilização do Flutter como framework para desenvolvimento da interface, sendo uma ferramenta poderosa e em constante crescimento.

A utilização dos conceitos EIP e utilização de ferramentas e bibliotecas para Java também foram um fator positivo. A utilização do Apache Camel proporcionou o contato direto com os conceitos de padrões de integração, bastante úteis em diversos cenários

reais. O contato com os ambientes de desenvolvimento Java possibilitaram a descoberta de novas ferramentas e conceitos.

Foram exercitados também conceitos de testes unitários, de integração, sistemas de integração e deploy automatizados, utilização de serviços em nuvem, essenciais no processo de desenvolvimento de software.

9.2 Contribuições

O processo de elaboração deste projeto foi realizado em conjunto com a empresa Opus Software, com participação na proposta de tema, em conjunto com o professor orientador Ricardo Luis de Azevedo da Rocha, resultando em um projeto com possibilidade de aplicação real dentro do contexto de desenvolvimento de soluções de integrações de sistemas.

A Opus também disponibilizou, durante o processo de desenvolvimento da aplicação, suporte técnico e operacional, sendo adotado um modelo de projeto com reuniões semanais e acompanhamento direto das atividades de desenvolvimento.

Em especial, ressalta-se que a interface da aplicação foi proposta pela colaboradora da Opus Software Samanta Lira Ferreira, baseando-se no protótipo inicialmente realizado de forma independente.

Foram utilizados como base deste projeto diversos softwares de uso livre, frameworks de desenvolvimento de aplicações, bibliotecas diversas e sistemas de diagramação.

Listam-se a seguir os principais recursos utilizados para realização do projeto:

- **Overleaf:** Editor LaTeX online, utilizado em conjunto com o projeto “politex” disponibilizado por Luiz Chamon.
- **Lucidchart e Draw.io:** Utilizados para elaboração de diferentes diagramas utilizados neste projeto
- **Softwares Apache:** Foram utilizados para realização deste projeto o Apache Camel e Apache Maven

9.3 Perspectivas de Continuidade

Como citado anteriormente, pretende-se que este projeto seja utilizado dentro da empresa como ferramenta auxiliar para projetos de integração. Desta forma sua perspectiva de continuidade é um importante fator dentro deste projeto.

Citam-se a seguir algumas propostas e etapas de continuidade deste projeto:

- **Revisão, refatoração e complementação da Documentação do projeto:** Este trabalho foi realizado tendo-se em mente a necessidade de manutenção e possibilidade de expansão de suas funcionalidades, além de se propor a vir a se tornar um projeto open source. Desta maneira torna-se essencial garantir a qualidade do código, documentação e testes. Durante o seu desenvolvimento, tomou-se cuidado para que tais garantias fossem cumpridas, porém sempre há espaço para melhorias, em especial com o auxílio de profissionais e desenvolvedores mais experientes. Sendo assim, o processo de revisão, refatoração e complementação da documentação deve ser entendido como uma etapa essencial para a continuidade deste projeto.
- **Expansão das funcionalidades específicas de EIP:** A proposta principal deste projeto é a realização de um editor de fluxos de integração que utilizam o Apache Camel como ferramenta para execução das integrações. Tanto as definições EIP quanto as implementações presentes no Apache Camel são bastante numerosas, no que se diz respeito aos diferentes tipos e variantes de padrões de integração. Este projeto foi realizado de forma que uma versão funcional pudesse ser construída e expandida a partir deste estado, não tendo sido disponibilizados todos os padrões e elementos EIP para serem utilizados na diagramação dos projetos de integração. Esta portanto é uma etapa necessária de continuação para possibilitar utilizações mais abrangentes e completas.
- **Expansão das funcionalidades para outros tipos de projeto:** Durante o desenvolvimento deste projeto, notou-se a possibilidade de extensão de suas funcionalidades para outros tipos de projeto que não sejam de integração ou utilizem Apache Camel como base. De fato pode ser visto como um editor de linguagens visuais diversas, que pode ser utilizado para gerar níveis de abstração diferentes para diversas outras aplicações, por exemplo: Utilização para auxiliar na aprendizagem de programação por meio de interface visual amigável, ferramenta para agilizar prototipação e criação de projetos específicos como geração de redes neurais de forma visual utilizando TensorFlow ou PyTorch, entre outras.

- **Melhorias na interface:** Esta aplicação exige a apresentação de uma interface que seja simples, amigável e que seja capaz de agilizar e facilitar o processo de desenvolvimento e programação. Desta maneira, a continuidade no desenvolvimento de novos mecanismos que auxiliem nestes aspectos se torna necessária.

9.4 Acesso ao Projeto e Código Fonte

A seguir são listados alguns links úteis para acessar o projeto e seu código fonte:

- **Código Fonte:**

Pode ser encontrado no repositório aberto <https://github.com/rodipm/TccOpus>. Pretende-se, futuramente, mover este repositório para um específico do projeto. Quando feito, o novo endereço será indicado de forma clara no repositório atual.

- **Acesso ao Projeto:**

O projeto está hospedado no seguinte endereço: <https://rodipm.github.io/TccOpus/> contendo a aplicação e uma página com informações sobre o projeto.

REFERÊNCIAS

- [1] HENNESSY J. L.; PATTERSON, D. A. *Computer Architecture; A Quantitative Approach*. 6. ed. [S.l.]: San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017. ISBN 9780128119051.
- [2] MCKENNEY, P. E. *Is parallel programming hard, and, if so, what can you do about it?* [S.l.: s.n.], 2011.
- [3] YADAVA, R. S. Review of the parallel programming and its challenges on the multicore processors. *Asian Journal of Computer Science and Technology*, v. 4, n. 1, p. 8–13, 2015.
- [4] GIMÉNEZ, D. A practical parallel programming course based on problems of the spanish parallel programming contest. *Procedia Computer Science*, v. 80, n. 1, p. 1978–1988, 2016. ISSN 1877-0509. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1877050916310067>>.
- [5] D'ANGELO G.; MARZOLLA, M. New trends in parallel and distributed simulation: from many-cores to cloud computing. Disponível em: <<http://arxiv.org/abs/1407.6470>>.
- [6] FOWLER, M. *Patterns of Enterprise Application Architecture*. [S.l.]: USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420.
- [7] HOHPE G.; WOOLF, B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Prentice Hall, 2004. (The Addison-Wesley Signature Series), 2003. ISBN 9780321200686. Disponível em: <<http://books.google.com.au/books?id=dH9zp14-1KYC>>.
- [8] IBSEN C.; ANSTEY, J. *Camel in Action*. 2nd. ed. [S.l.]: USA: Manning Publications Co., 2018. ISBN 1617292931.
- [9] MILLER, F. P.; VANDOME, A. F.; MCBREWSTER, J. *Apache Maven*. [S.l.]: Alpha Press, 2010.
- [10] LOUGHRAN, S.; HATCHER, E. *Ant in Action*. [S.l.]: Manning Publications, 2007. ISBN 193239480X.
- [11] HALLER P.; SOMMERS, F. *Actors in Scala*. [S.l.]: USA: Artima Incorporation, 2012. ISBN 0981531652.
- [12] GUSTAFSON, J. L. Reevaluating amdahl's law. *Commun. ACM, ACM, New York, NY, USA*, v. 31, n. 5, p. 532–533, 1988. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/42411.42415>>.
- [13] ZHENG, L. e. a. Reevaluating amdahl's law. In: . Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques, New York, NY, USA: ACM: [s.n.], 2018. p. 26:1–26:13. ISBN 978-1-4503-5986-3.

- [14] HEWITT, C. A historical perspective on developing foundations iinfo(tm) information systems: iconult(tm) and ientertain(tm) apps using idescribers(tm) information integration for iorgs(tm) information systems. CoRR, abs/0901.4934, 2009. Disponível em: <<http://arxiv.org/abs/0901.4934>>.
- [15] HOARE C.; JIFENG, H. *Unifying Theories of Programming*. Prentice Hall, 1998. (Prentice Hall series in computer science). ISBN 9780134587615. Disponível em: <<https://books.google.com.br/books?id=WpdQAAAAMAAJ>>.
- [16] SOTTILE, M.; MATTSON, T. G.; RASMUSSEN, C. E. *Introduction to Concurrency in Programming Languages*. [S.l.]: Chapman & Hall, 2009. ISBN 1420072137, 9781420072136.
- [17] HEWITT, C. Viewing control structures as patterns of passing messages. *Artif. Intell*, Elsevier Science Publishers Ltd., Essex, UK, v. 8, n. 3, p. 323–364, 1977. ISSN 0004-3702. Disponível em: <[http://dx.doi.org/10.1016/0004-3702\(77\)90033-9](http://dx.doi.org/10.1016/0004-3702(77)90033-9)>.
- [18] AGHA, G. An algebraic theory of actors and its application to a simple object-based language. In: . Ole-Johan Dahl's Festschrift, volume 2635 of LNCS. Heidelberg, Germany: Springer, 2004. p. 26–57.
- [19] HALLER P.; ODERSKY, M. Actors that unify threads and events. In: . MURPHY, A. L.; VITEK, J. (Ed.). *Coordination Models and Languages*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 171–190. ISBN 978-3-540-72794-1.
- [20] GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Publishing Co., Boston, Massachusetts, USA, 1995.
- [21] ULLMAN, A. V. A. M. S. L. R. S. J. D. *Compilers: Principles, Techniques, and Tools*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
- [22] NETO, J. J. *Introdução à Compilação*. [S.l.]: Elsevier Editora Ltda, 2016.
- [23] HILS, D. D. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages and Computing*, Academic Press Limited, 1992.
- [24] MALONEY, J. et al. The scratch programming language and environment. *Journal of Visual Languages and Computing*, ACM Transactions on Computing Education, n. 16, 2010.
- [25] MIOLA, A. *Flutter Complete Reference*. [S.l.: s.n.], 2020.
- [26] GAYNOR, A. Rply. 2014. Disponível em: <<https://rply.readthedocs.io/>>.
- [27] LATTNER C.; ADVE, V. A compilation framework for lifelong program analysis & transformation. In: . Code Generation and Optimization (CGO). Palo Alto, California: Springer Berlin Heidelberg, 2004. v. 1, n. 1. ISBN 0-7695-2102-9.
- [28] LLVM.ORG. Llmv tutorial. Disponível em: <<https://llvm.org/docs/tutorial/index.html>>.
- [29] BENDERSKY, E. Pykaleidoscope. Disponível em: <<https://github.com/eliben/pykaleidoscope>>.

APÊNDICE A – IMAGENS DA INTERFACE DA APLICAÇÃO

São apresentadas algumas imagens ampliadas da interface do sistema desenvolvido, para facilitar a visualização.

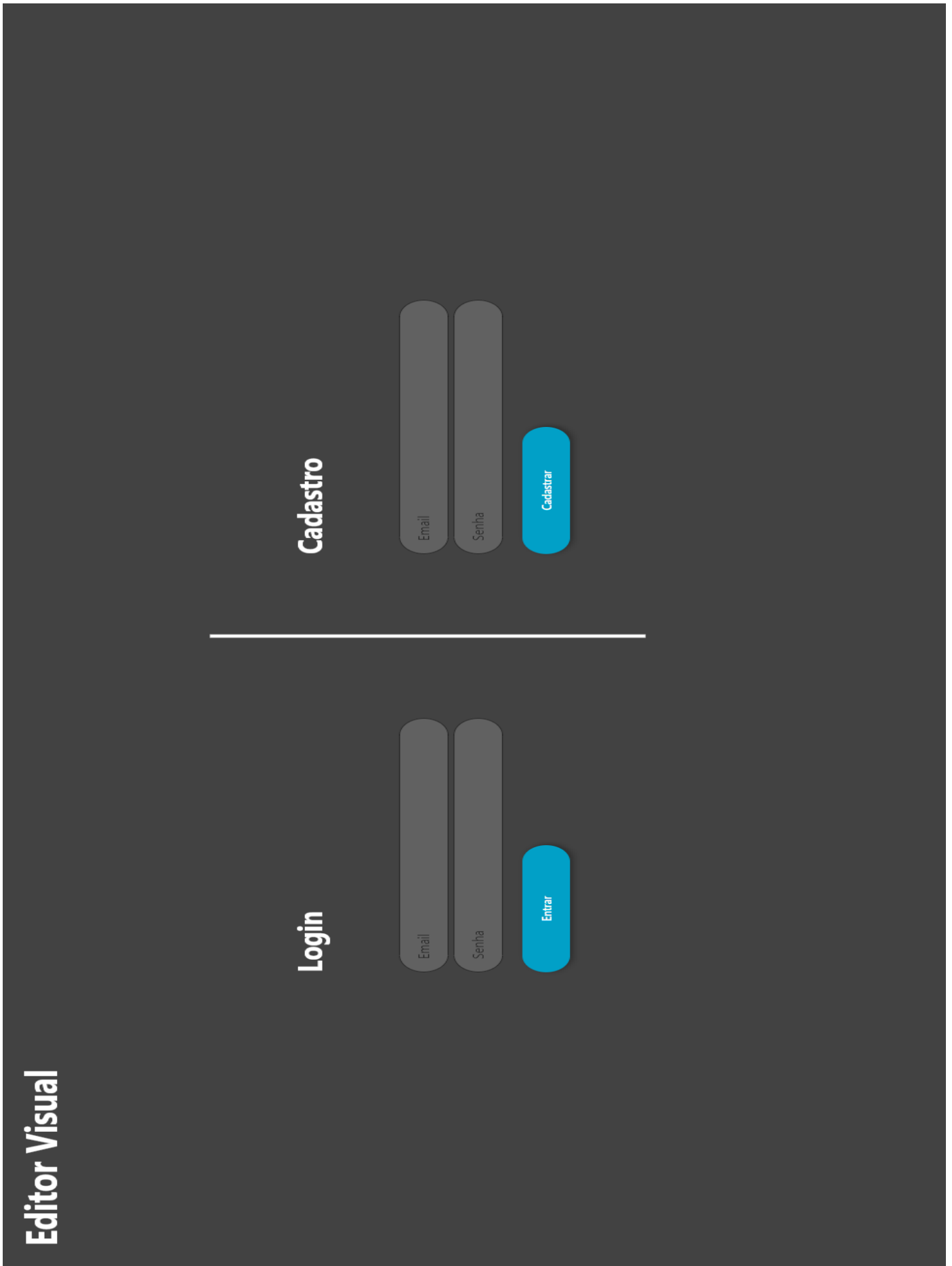


Figura 21: Detalhe da tela de Cadastro e Login no Sistema

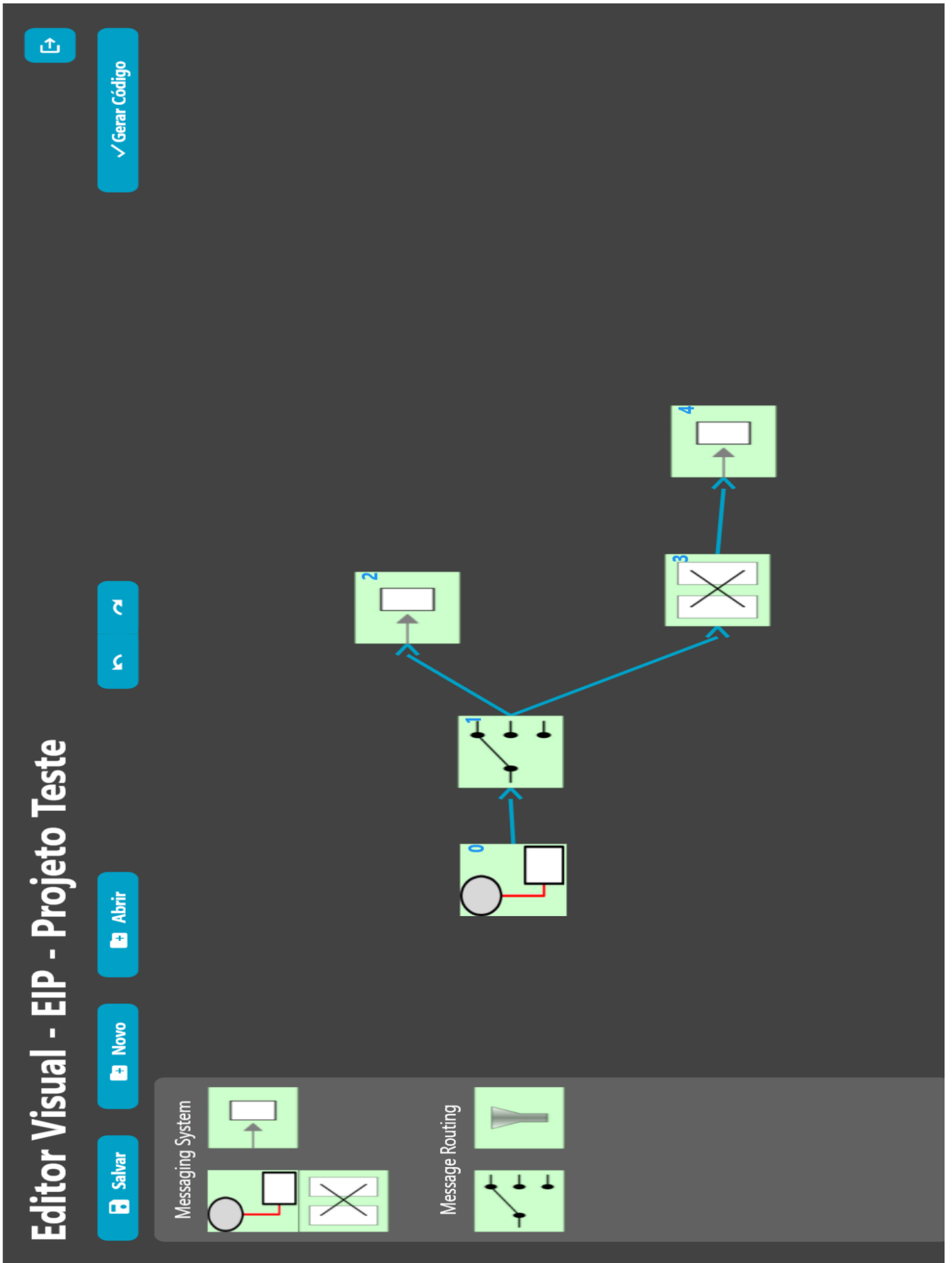


Figura 22: Detalhe da tela base da aplicação, contendo o canvas principal com o qual o usuário pode interagir

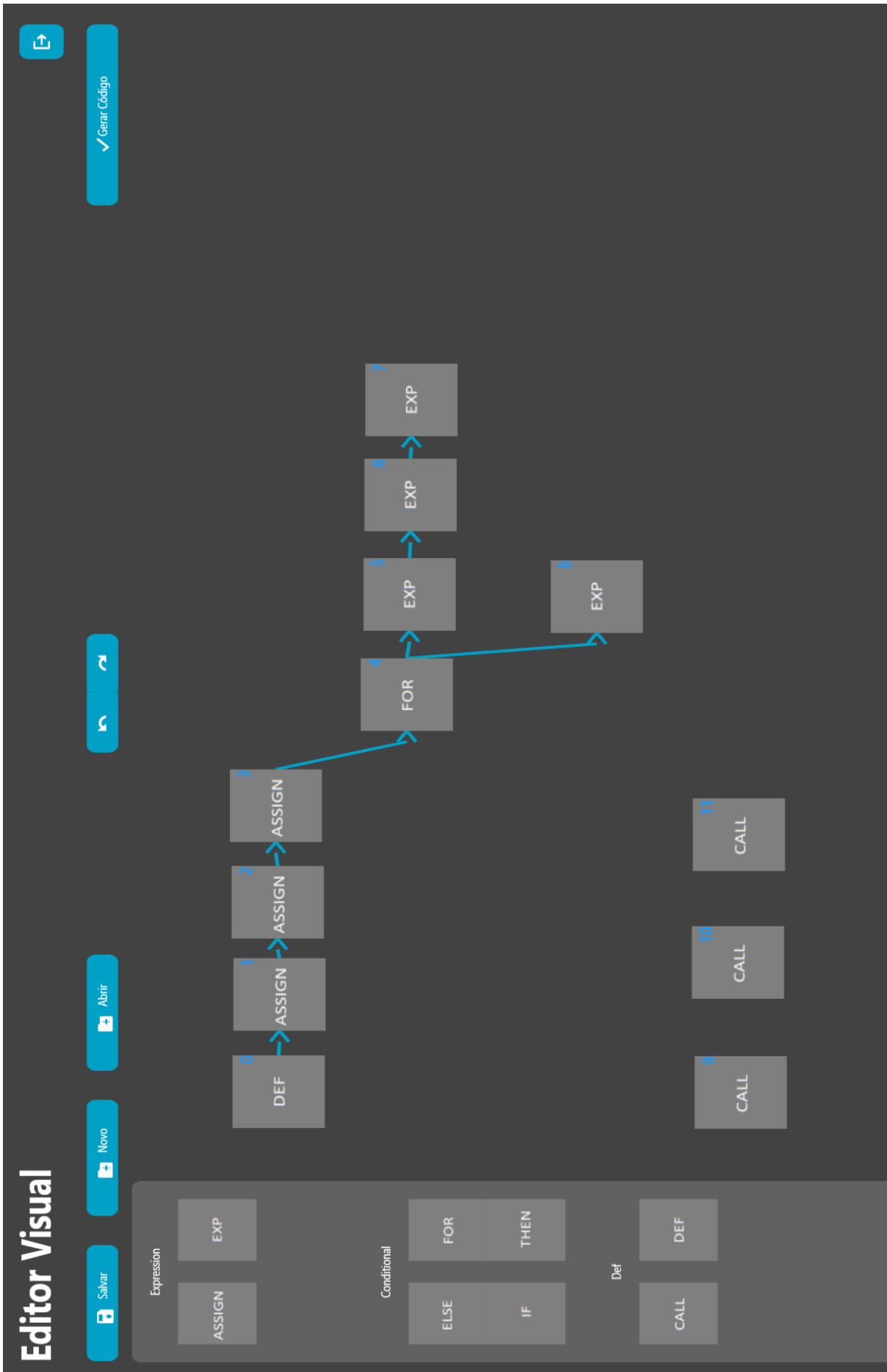


Figura 23: Detalhe do diagrama para a implementação da função de obtenção de números da sequência de Fibonacci.

APÊNDICE B – GERADOR DE CÓDIGO BASIC

Neste capítulo documenta-se uma terceira proposta de geração de código a partir de diagramas. Neste caso trata-se da Linguagem Dartmouth BASIC, descrita adiante.

O processo de implementação é bastante similar ao apresentado no Capítulo 7. Sendo assim, este apêndice apenas apresenta algumas informações sobre sua implantação e demonstração dos resultados.

Não se trata de uma funcionalidade mantida no projeto, sendo apenas documentada para melhor exemplificar a facilidade de expansão dos funcionamentos do projeto.

B.1 Alterações no Front-End

Os elementos visuais utilizados para representação da linguagem BASIC são os seguintes:

- Assign: Criação e atribuição de valores para variáveis.
- For: Definição de laços For, contendo uma variável de controle interno, com definição do seu valor inicial, valor de parada e incremento.
- If: Desvio condicional.
- Goto: Desvio incondicional.
- Print: Saída de texto.

B.2 Alterações no Back-End

A arquitetura utilizada para implementação do código BASIC é basicamente a mesma das anteriores. Alguns detalhes são mostrados adiante.

B.2.1 Parser

Utiliza como base um subconjunto da gramática do BASIC apresentada a seguir:

```

Program : BStatement {BStatement}
BStatement : "INTEGER" Assign | Print | Goto | Gosub | Return
            | IF | For | Next | Def | Remark
Assign : "LET" Var "=" Exp
Var : Id [ "OPENBRACKET" Exp { "COMMA" Exp} "CLOSEBRACKET" ]
Exp: Term {"PLUS" | "MINUS"} Term}
Term: Eb {"MUL" | "DIV" Eb}
Eb: ("PLUS" Eb | "MINUS" Eb | "INTEGER"
     | "LPAREN" Exp "RPAREN" | "INTEGER" | Var
     | FN ID LPAREN Exp RPAREN)
     | RND LPAREN Exp RPAREN
Print : "PRINT" Pitem {"COMMA" Pitem}
Put : "PUT" Pitem {"COMMA" Pitem}
Pitem : Exp | String
Read : "READ" Var
Goto : ("GOTO" | "GO" "TO") "INTEGER"
Gosub : "GOSUB" "INTEGER"
Return : "RETURN"
If : "IF" Exp ("EQ" | "NOTEQ" | "GT" | "LESS" | "GTEQ" | "LESSEQ")
     Exp "THEN" "INTEGER"
For : "FOR" Id "EQUAL" Exp "TO" Exp ["STEP" Exp]
Next : "NEXT" Id
Def : "DEF" "FN" Id "LPAREN" Id "RPAREN" "EQUAL" Exp
Dim : "DIM" Id "LPAREN" "INTEGER" {"COMMA" "INTEGER"} "RPAREN"
Remark : "REM" String
Id: letter{digit|letter}
String: QUOTE {character} QUOTE

```

Ao final do processamento do Parser, obtém-se a representação do diagrama elaborado na forma de uma AST, sendo possível dar continuidade ao processo de geração de código.

B.2.2 Gerador de Código Basic

Com a AST gerada, este componente realiza a transformação da AST para representação na linguagem BASIC definida no item anterior, sendo realizado de forma recursiva, da mesma forma como feito para as demais linguagens.

B.2.3 Compilador BASIC Para Linguagem de Máquina

Apenas para propósitos de testes, testou-se o código gerado em um compilador previamente implementado em outro projeto independente.

Este compilador foi realizado de maneira similar a apresentada no processo de construção dos componentes deste trabalho, utilizando as mesmas técnicas e ferramentas conceituas, apenas sendo implementado para a gramática específica mostrada anteriormente.

B.3 Exemplo de Funcionamento

Adiante mostram-se as imagens de um diagrama simples e seu respectivo código BASIC gerado.

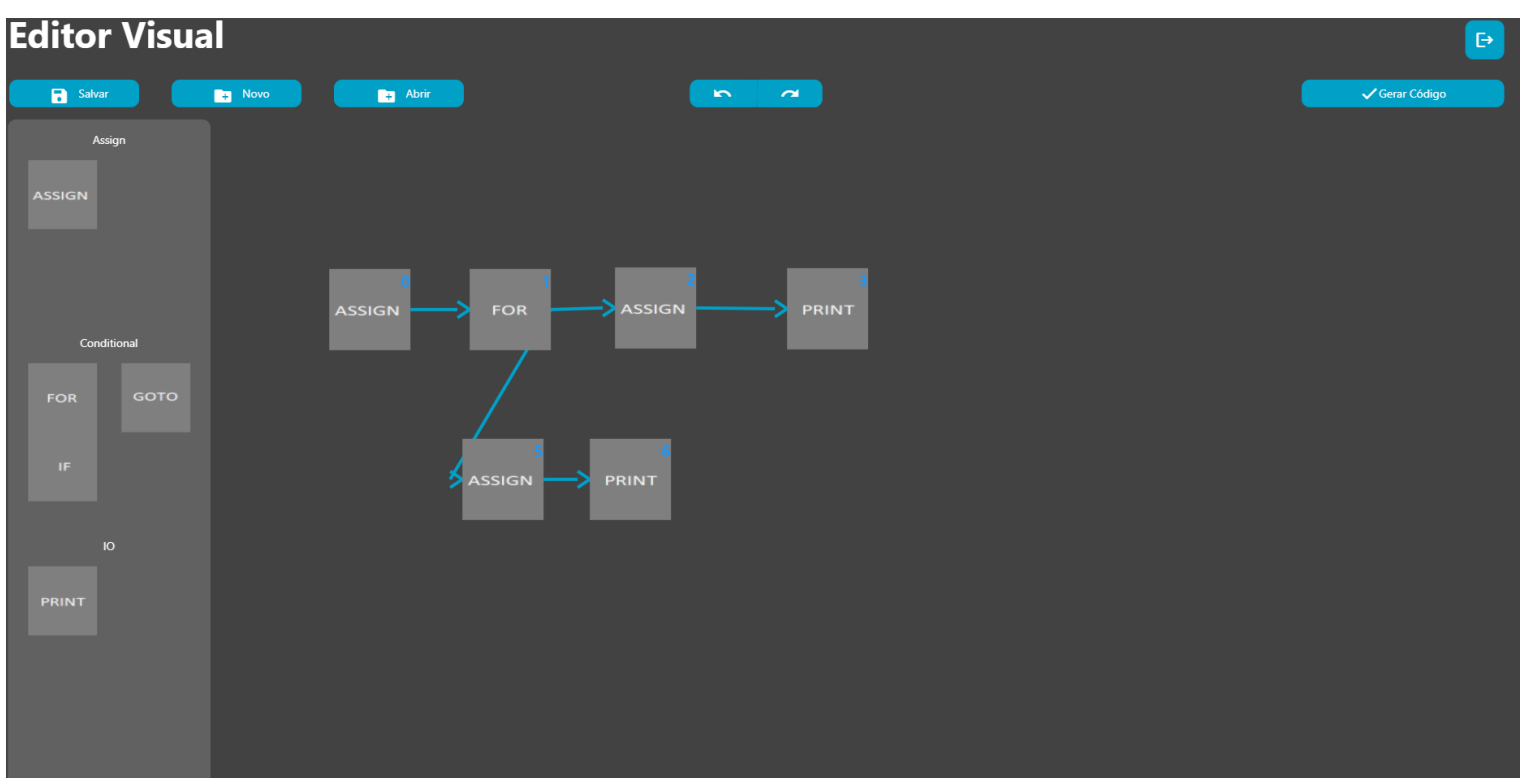


Figura 24: Simple diagrama para exemplificação da geração de código para BASIC.

```

Código Gerado

0 LET X = 1
1 FOR Y = 0 TO 10 INCREMENT 1
5 LET X = X+1
6 PRINT X
01 NEXT Y
2 LET Y = X
3 PRINT Y

```

Figura 25: Exemplo do código BASIC gerado.

Nota-se que os ID's dos componentes visuais foram utilizados como base para a numeração das linhas dos códigos BASIC, uma vez que são utilizados como labels, e as linhas não são reordenadas pelo compilador utilizado.