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 obtenção do Título de Engenheiro .

Área de Concentração:
Engenharia de Computação

Orientador:
Ricardo Luis de Azevedo da Rocha

São Paulo
23/06/2020

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 o Apache Camel por meio de uma interface gráfica, simples e acessível, gerando o código em Java correspondente.

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

SUMÁRIO

1 INTRODUÇÃO

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.

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 RedHat. 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 empregando Apache Camel conjuntamente com um gerador do código correspondente em Java. O gerador deverá ser desenvolvido de maneira a ser incrementalmente modificado para agregar outras linguagens como Scala,

ou Kotlin, por exemplo.

Como objetivos secundários define-se a aplicação em si, um editor de fluxos de integração Web baseado na estruturação do fluxo de integração em forma 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.

A aplicação deve gerar o código e o projeto na linguagem Java, utilizando-se o Camel, de forma a poder ser utilizado diretamente pelo usuário de forma direta, removendo a interação do usuário com o código propriamente dito.

Todo o software será 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.

É possível facilitar a criação de sistemas de integração é possibilitar a automação e facilitar a geração do código associado. Uma forma possível de abordagem é possibilitar a geração dos fluxos de integração por meio de diagramas, em especial utilizando diretamente os elementos EIP.

A ferramenta JBoss Fuse Tooling, densenvolvida pela Red Hat, é um bom exemplo deste conceito. Funcionando como uma interface para utilização do Apache Camel, implementada utilizando o conceito de diagramação. 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.

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 colabora para facilitar o acesso à ferramenta e possibilitar interações mais simples com o usuário.

2 ASPECTOS CONCEITUAIS

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 [9].

Infelizmente, obter a mesma velocidade de processamento provou ser extremamente difícil, como mostra a Lei de Amdahl [10]. 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 [11].

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 [12] afirmou sabiamente, “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 [13, 14].

O modelo de ator escolheu outra direção desde o início [15, 16], 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 [17].

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) [18] 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. Estas permitem criar uma sequencia 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.

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 no livro Compilers: Principles, Techniques, and Tools (A. V. Aho, M. S. Lam, R. Sethi, e J. D. Ullman) [19].

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.

O processo de análise da formação e estrutura do diagrama montado faz necessária a implementação de um parser, conceito importante dentro de compiladores.

A geração de uma linguagem intermediária que representa as informações do diagrama de forma a possibilitar a geração do código final também é um tema importante. A utilização da linguagem intermediária LLVM [20] é uma possibilidade para este projeto.

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

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.

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.

A comunicação com o back-end é feita via protocolo HTTP, utilizando-se do modelo REST.

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

3.2.1 Servidor WEB

Desenvolvido em Python com auxílio do framework Flask para a implementação da arquitetura REST.

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

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

3.2.2 Gerador de Código

Os módulos correspondentes a parte de geração de código do projeto são implementados utilizando Python, utilizando conceitos de compiladores, baseando-se em bibliotecas existentes para desenvolvimento de compiladores.

A biblioteca RPLY disponível para Python é capaz de gerar Parsers e analisadores Léxicos. Muitos de seus conceitos são aplicados ao desenvolvimento deste projeto.

3.2.3 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 execucão dos projetos finais.

4 METODOLOGIA DO TRABALHO

O desenvolvimento da aplicação deve ser realizado de forma incremental e continua, 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, constituida pelos livros Enterprise Integration Patterns [7], Patterns of Enterprise Application Architecture [6] e Camel in Action [21].

Outra parte importante do projeto é a geração de código, baseado nos conceitos associados a compiladores. O livro Compilers: Principles, Techniques, and Tools [19] 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, pretende-se codificar a interface e o servidor, incluindo a funcionalidade de

diagramação com elementos EIP e geração de código. Não serão 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.

Pretende-se implementar 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 pelos usuários.

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

Por fim, a possibilidade de aprimoramento da geração de código para se utilizar LLVM [20] e expandir a geração de código para outras linguagens também deve ser explorado neste momento do desenvolvimento.

5 ESPECIFICAÇÃO DO SISTEMA

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 fluxo de integração
- RF03 - Criar diagramas no estilo "flowchart"utilizando-se de elementos e padrões EIP
- RF04 - Editar e modificar a disposição dos elementos EIP utilizados
- RF05 - Possibilitar e configuração dos diferentes elementos EIP para criação das rotas do Apache Camel

5.2 Arquitetura Da Aplicação

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

A estrutura geral do sistema pode ser visualizada na imagem a seguir.

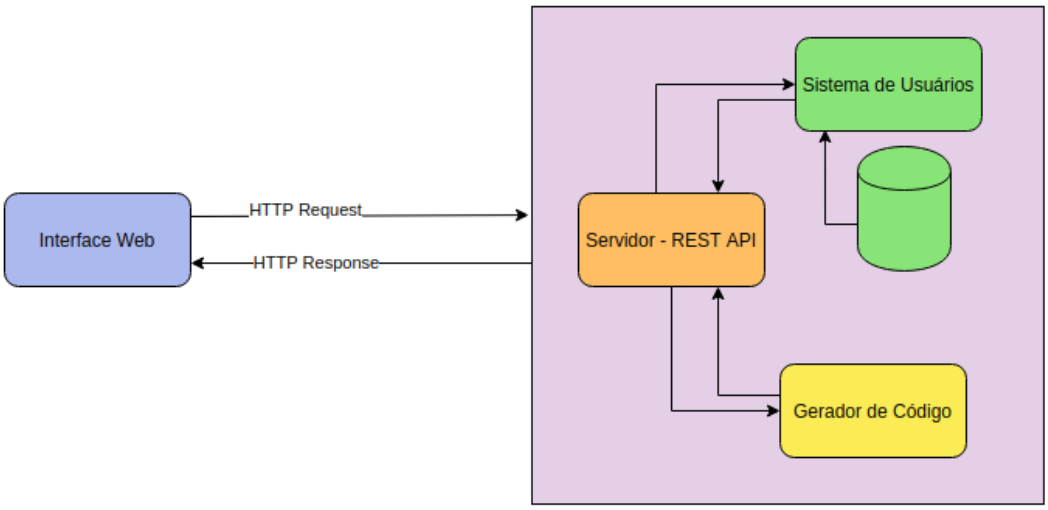

Figura 1: Arquitetura da Aplicação

5.3 Front-End

Representado na parte esquerda da imagem como interface Web.

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 imagem abaixo representa um protótipo da interface da aplicação implementada na primeira parte deste projeto.

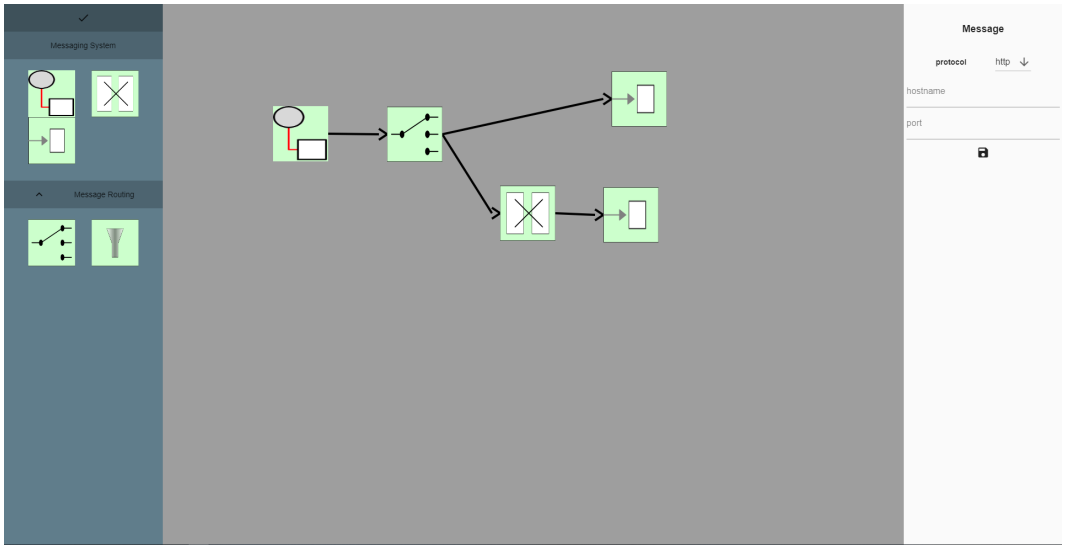

Figura 2: Protótipo da Interface com o Usuário

5.3.1 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, com cor cinza.

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

A Figura 2 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.

5.3.2 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.

As interfaces de salvamento e geração de código devem ser implementadas neste painel.

5.3.3 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 se pretende-se garantir essa qualidade é disponibilizando interfaces simples e intuitivas para os diferentes tipos de elementos EIP.

Mostra-se na Figura 2 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 3.

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.

A Figura 3 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.

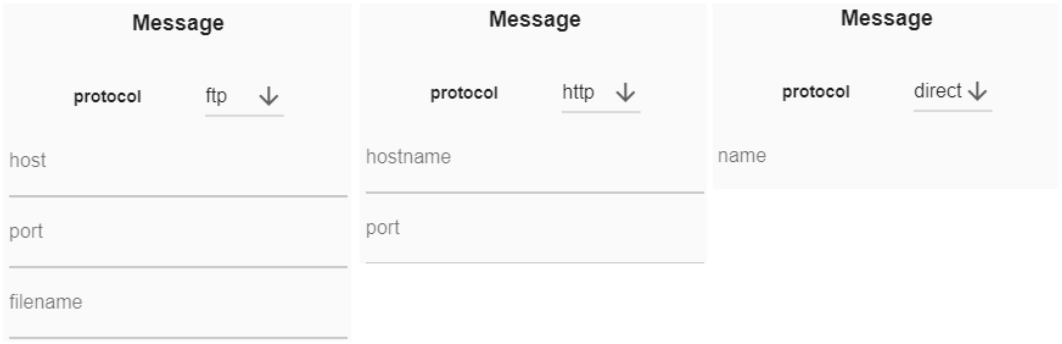

Figura 3: Diferentes apresentações do painel de configuração para diferentes protocolos para mensagens.

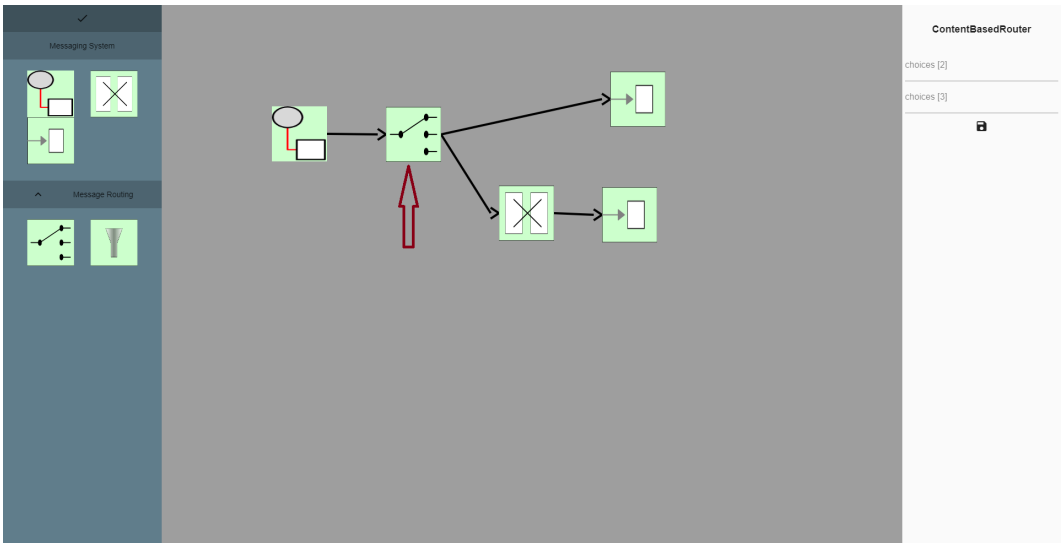

Figura 4: Exemplo do painel de configuração para o elemento Content Based Router.

5.4 Back-End

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

5.4.1 Servidor Web

Implementa os endpoints HTTP da API REST utilizada para a comunicação com o front-end. Utiliza-se da framework Flask em python, que permite sua execução de forma simples e direta.

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

5.4.2 Sistema de Usuários

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.

5.4.3 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.

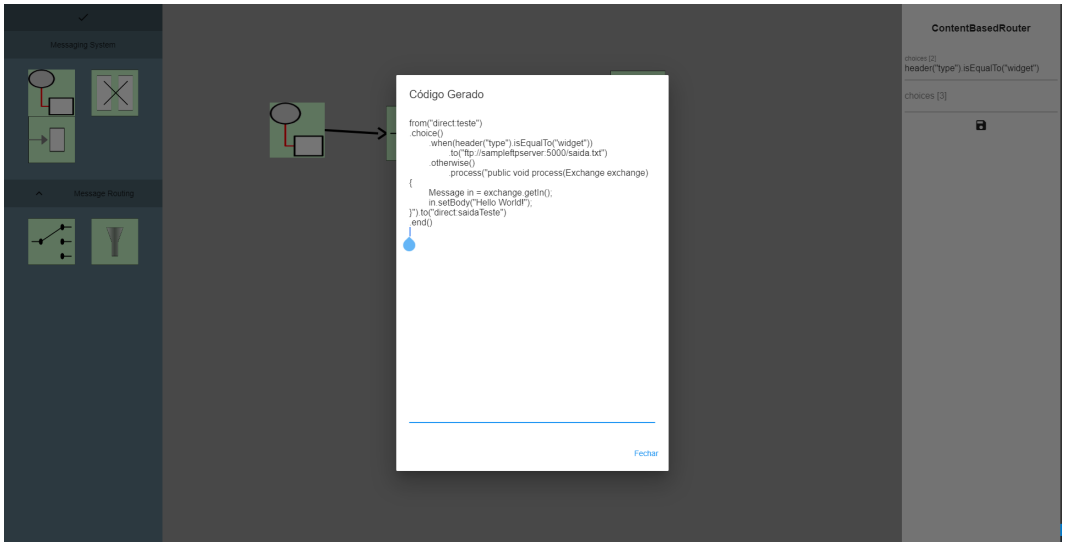

Figura 5: Exemplo de código gerado a partir do fluxo de integração de exemplo.

5.4.3.1 Gerador de Código

Pode-se observar na Figura 5 o resultado do gerador de código implementado para o protótipo utilizando o fluxo de integração mostrado na Figura 2. 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.

O sistema de geração de código é parte fundamental deste projeto e deve ser aprimorado conforme a continuidade do projeto. Sua implementação inicial 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. Ao terminar a busca em profundidade dos elementos do diagrama, o código da rota equivalente é gerado.

5.4.3.2 Parser

Pretende-se adicionar funcionalidades para verificação sintática do código gerado e de sua diagramação pelo meio de um parser. 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 formas 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 com inclusão do operador "*"(estrela) que denota um Fecho de Kleene, 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.

Para a primeira parte deste projeto foi construída uma gramática que representa as estruturas básicas encontradas nas rotas possíveis de serem montadas empregando padrões EIP e Aache Camel. Nota-se que não é uma gramática completa em relação a todos os possíveis elementos EIP disponíveis, mas sim apenas para aqueles que já 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 com letras maiúsculas são os não terminais, já os representados em letras minúsculas são os 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 deve ser completada e modificada ao longo da implementação do projeto com a expansão dos elementos EIP e das funcionalidades da aplicação.

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). 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] HALLER P.; SOMMERS, F. *Actors in Scala*. [S.l.]: USA: Artima Incorporation, 2012. ISBN 0981531652.

[10] 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>.

[11] 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.

[12] HEWITT, C. A historical perspective on developing foundations iinfo(tm) information systems: iconsult(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>.

[13] 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>.

[14] SOTTILE M.; MATTSON, T. G. R. C. E. *Introduction to Concurrency in Programming Languages*. [S.l.]: Chapman & Hall, 2009. ISBN 1420072137, 9781420072136.

[15] 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>.

[16] 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.

[17] 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.

[18] GAMMA RICHARD HELM, R. J. E.; VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Publishing Co., Boston, Massachusetts, USA, 1995.

[19] 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.

[20] 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.

[21] ANSTEY, C. I. J. 2nd. ed. [S.l.]: Manning Publications, 2018. ISBN 9781617292934.