UTILIZAÇÃO DOS CONCEITOS DE LOW CODE E NO CODE NA GERAÇÃO DE WEB SERVICES COM ARQUITETURA MDA

Alairton Dendena[1], Gustavo Bisognin [2]

Resumo: Modelos Específicos de Plataforma utilizados na Arquitetura Dirigida a Modelos geram artefatos com código-fonte que facilitam e aumentam a produtividade no desenvolvimento de software. Esta pesquisa desenvolve uma plataforma que extraí metadados de um banco de dados relacional gerando Web Services com uso dos conceitos de No-Code e Low-Code, transformando modelos e disponibilizando artefatos na linguagem de programação Java. Após uso da plataforma em bases de dados existentes, foram coletados dados de performance e analisados os artefatos gerados validando a execução dos mesmos. Com os resultados, notou-se maior sucesso nos métodos de busca e uma taxa menor em transações que validam regras de integridade do banco de dados.

Palavras-chave: Modelo Entidade-Relacionamento. Arquitetura Dirigida a Modelos. No/Low-Code. Web Services. API RESTful.

ABSTRACT: Platform Specific Models used in Model Driven Architecture generate artifacts with source code that facilitate and increase productivity in software development. This research develops a platform that extracts metadata from a relational database, generating Web Services using the concepts of No-Code and Low-Code, transforming models and making artifacts available in the Java programming language. After using the platform in existing databases, performance data were collected and the generated artifacts were analyzed, validating their execution. With the results, it was noticed greater success in search methods and a lower rate in transactions that validate database integrity rules.

Keywords: Entity-Relationship Model. Model Driven Architecture. No/Low-Code. Web Services. API RESTful.

[1] alairton.dendena@gmail.com
[2] gustavo@unesc.net

1 INTRODUÇÃO

No desenvolvimento de software é recorrente a necessidade de implementação de cadastros básicos associados a requisitos funcionais ou regras de negócio. Essas tarefas básicas consomem um tempo considerável no ciclo de vida do desenvolvimento de sistemas. (SOMMERVILLE, 2007).

A partir dos requisitos levantados para o desenvolvimento de uma aplicação ou sistema web, é possível definir a modelagem de dados usando o Modelo Entidade Relacionamento que representa a estrutura lógica do banco de dados, abordando objetos chamados entidades e os relacionamentos entres estes objetos (ARAÚJO, 2008).

A modelagem de dados é uma etapa essencial que eleva o nível de abstração no desenvolvimento, ajudando em seu planejamento e entendimento. Essa etapa pode ser ainda mais importante visando sanar problemas de estruturação de projeto e otimização da implementação das tarefas básicas conforme citado anteriormente. Para tanto, utiliza-se a abordagem da geração automática de código fonte a partir do modelo do sistema, promovendo-o a artefato principal do desenvolvimento.

A Arquitetura Dirigida a Modelos (MDA) desponta como uma solução emergente para os problemas inerentes ao desenvolvimento de sistemas, isto porque possui uma abordagem centrada no conceito de modelos, no relacionamento entre eles e nas transformações entre estes modelos. (MOREIRA; MRACK, 2003).

O seu principal objetivo é isolar a lógica de negócio da aplicação, da evolução e manutenção da tecnologia, e apresenta como proposta a construção de sistemas de forma rápida, consistente e independente de plataforma. O desenvolvimento se torna rápido porque a maior parte do código é gerado a partir das especificações de condições e transformações entre modelos, sendo que novos modelos compatíveis com uma plataforma específica são obtidos sem a necessidade de um novo processo de desenvolvimento. (MOREIRA; MRACK, 2003).

A base do MDA utiliza modelos como dados que podemos consultar, analisar, validar, simular e transformar em outros formatos úteis, como código executável. (FAVRE, 2010, tradução nossa).

No MDA um modelo é a representação de algum sistema cuja forma e conteúdo são definidos com base em um conjunto específico de informações. O

mapeamento de um modelo relacionado a um sistema pode ser explícito ou implícito e representa aspectos específicos de um determinado domínio. O modelo pode incluir metamodelos em sua composição, como por exemplo, um diagrama de classes, de sequência ou um protótipo de interface do usuário (OBJECT MANAGEMENT GROUP (OMG), 2010, tradução nossa).

Desta forma, com base nos estudos dos temas acima citados, este projeto visa o desenvolvimento de uma plataforma que utiliza modelos para geração de código-fonte de aplicações, minimizando o esforço e trabalho de codificação para nenhum ou próximo a isso, sendo essa uma abordagem diretamente relacionada aos conceitos de No-Code e Low-Code.

2 MATERIAIS E MÉTODOS

Conforme Wazlawick (2021), o presente trabalho proposto é uma pesquisa descritiva e bibliográfica com base tecnológica, caracterizada pelo levantamento de dados e informações, por meio do estudo de artigos, teses e livros sobre modelos, arquiteturas e processos relacionados ao tema selecionado.

Para obtenção dos metadados e seleção das ferramentas de desenvolvimento, será aplicado o MDA, ou seja, usando os conceitos de CIM, PIM e principalmente PSM.

O Modelo Independente de Computação (CIM) também é frequentemente referido como um modelo de negócio ou de domínio porque usa um vocabulário familiar aos especialistas no assunto. Ele apresenta exatamente o que se espera que o sistema faça, mas oculta todas as especificações relacionadas à tecnologia, independentemente de como o sistema está ou será implementado. O CIM desempenha um papel importante em preencher a lacuna que normalmente existe entre esses especialistas de domínio e os responsáveis pela implementação do sistema. Em uma especificação de MDA, os requisitos de CIM devem ser rastreáveis às construções PIM e PSM que os implementam e vice-versa (TRUYEN, 2006, tradução nossa).

Construído por especialistas em negócios e de modelagem trabalhando juntos, o Modelo Independente de Plataforma (PIM) expressa regras de negócios e funcionalidades não distorcidas, tanto quanto possível, pela tecnologia (SIEGEL; GROUP, 2001, tradução nossa).

O grau de independência em um PIM deve ser suficiente para permitir seu mapeamento para uma ou mais plataformas. Isso é comumente alcançado definindo um conjunto de serviços de forma que seja possível abstrair todos detalhes técnicos. (TRUYEN, 2006, tradução nossa).

A especificação de um sistema em termos de desenvolvimento e implementação é a abordagem utilizada em um Modelo Específico de Plataforma (PSM). Neste modelo deve-se levar em conta quais são as tecnologias disponíveis para uso na transformação de mapeamentos. Um PIM pode ser transformado em um ou mais PSM (HUANG; CHU, 2014, tradução nossa).

O MDA fornece especificações de como transformar um modelo PIM em um PSM específico, sendo que o modelo da plataforma de destino determina a natureza do mapeamento. Embora parte da transformação possa resultar de um exercício manual, a intenção é claramente automatizar o processo tanto quanto permitido pelo conjunto de ferramentas em uso. A especificação Meta Object Facility (MOF) fornecida pela OMG, é um padrão onde se escolhe a linguagem para definir as linguagens de modelagem. Podemos entender essa linguagem como a metalinguagem relacionada aos metamodelos utilizados na extração de metadados da modelagem selecionada. Neste padrão, os modelos podem ser exportados de uma aplicação, importados para outra, transportados pela rede, armazenados em um repositório e renderizados em diferentes formatos usados para gerar o código de aplicações (TRUYEN, 2006, tradução nossa).

A etapa final do desenvolvimento é a transformação de cada PSM em um modelo de código. A transformação de PSM em modelos de código é direta (HUANG; CHU, 2014, tradução nossa).

Quanto mais completa a semântica da aplicação e o comportamento em tempo de execução incluído no PSM, mais completos os artefatos gerados podem ser. Dependendo da maturidade e da qualidade do conjunto de ferramentas MDA, a geração de código varia de significativa a substancial ou, em alguns casos, até completa. Observe que mesmo a automação mínima simplifica o trabalho de desenvolvimento e representa um ganho significativo devido ao uso de uma arquitetura consistente para gerenciar os aspectos independentes e específicos da plataforma das aplicações (TRUYEN, 2006, tradução nossa).

Como todos sabemos, a redução da codificação manual e a detecção prévia de bugs podem aumentar drasticamente a probabilidade de entregar um

sistema de alta qualidade no prazo e dentro do orçamento (DUBY, 2003, tradução nossa).

A ferramenta de geração do código relacionada a última etapa do processo do MDA será desenvolvida utilizando a linguagem Java, bem como será necessário o uso de bibliotecas para criação dos arquivos de código e para leitura da metalinguagem relacionada a modelagem da estrutura do banco de dados relacional, a qual será convertida em uma aplicação. A aplicação selecionada para demonstração será um Web Service, e este terá a arquitetura REST.

De modo geral, conforme Figura 1, mediante a ferramenta será possível gerar projeto e código Java seguindo os padrões necessários para se disponibilizar uma API RESTful, tendo o mínimo ou nenhum esforço por parte do usuário que a utilizará.

Figura 1 – Caso de uso.

Fonte: Elaborado pelo autor.

Ao final, a API gerada poderá ser apresentada e o código-fonte poderá ser demonstrado e alterado para validação do processo de manutenção com foco na aplicação de regras de negócio.

2.1 EXTRAÇÃO DE METAMODELOS

Segundo Chen (1975, tradução nossa), o Modelo Entidade Relacionamento (MER) é composto basicamente por entidades e relacionamentos em um domínio específico de conhecimento baseado em algumas das informações semânticas importantes sobre o mundo real.

O MER que foi desenvolvido e publicado em 1976 por Peter Chen, continua sendo uma das técnicas de modelagem mais amplamente utilizadas no campo da Engenharia de Software. Indiscutivelmente, seu sucesso deriva em grande parte de sua aplicação ao projeto lógico de bancos de dados relacionais (GONZÁLEZ JIMÉNEZ, 2006, tradução nossa).

Um metamodelo é a abstração de uma linguagem de modelagem, e como tal, é um tipo especial de modelo. Pode ser entendido como a especificação do conjunto de todos os modelos possíveis expressos nesta linguagem de modelagem. Cada modelo deve estar em conformidade com seus metamodelos (OBJECT MANAGEMENT GROUP (OMG), 2010, tradução nossa).

2.1.1 Banco de dados

O desenvolvimento do projeto parte da seleção de um banco de dados para extração de suas informações como modelos e metamodelos os quais serão convertidos em código-fonte de aplicação. Tendo isso em mente, é necessário selecionar qual Banco de dados será utilizado, sua versão, bem como uma biblioteca de driver compatível com a plataforma e linguagem de programação de interesse, em nosso caso Java.

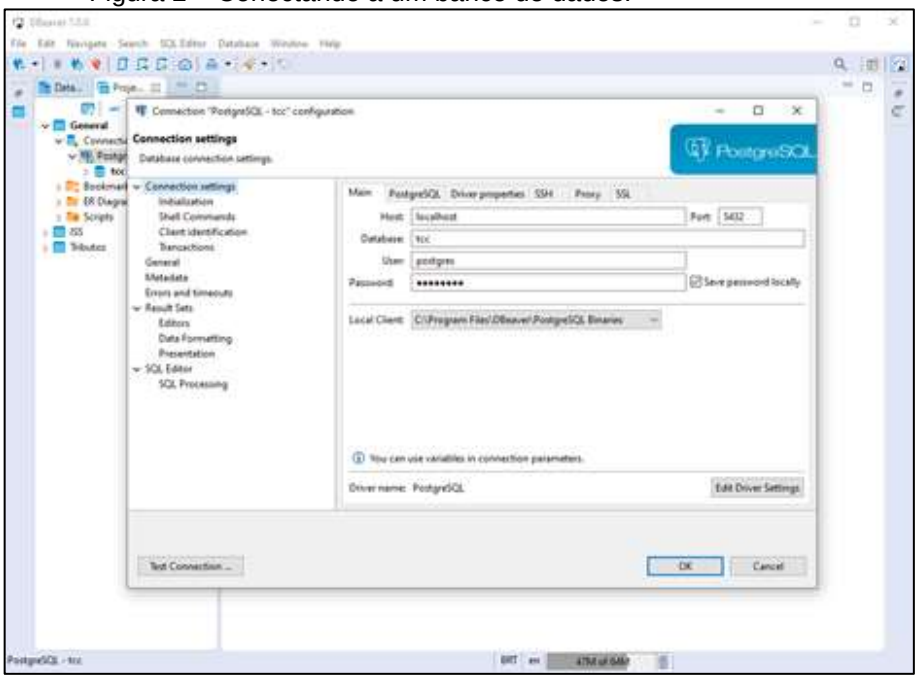
Para facilitar a acessibilidade ao banco de dados costuma-se usar um gerenciador. Neste projeto estarei utilizando o aplicativo desktop DBeaver[3], uma ferramenta de administração de banco de dados relacionais. Ele usa a interface de programação de aplicativo JDBC para interagir com os bancos de dados por meio de um driver.

O JDBC[4] fornece acesso universal a dados a partir da linguagem de programação Java. Usando a API JDBC, você pode acessar virtualmente qualquer fonte de dados, de bancos de dados relacionais a planilhas e arquivos simples. A tecnologia JDBC também fornece uma base comum na qual ferramentas e interfaces alternativas podem ser construídas.

[3] Mais informações em: https://dbeaver.io/about
[4] Mais informações em: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc

Conforme Figura 2, podemos nos conectar com um banco de dados desejado, para tanto, basta ter as credenciais de acesso ao mesmo. Neste projeto selecionaremos o PostgreSQL.

Figura 2 – Conectando a um banco de dados.



Fonte: Do autor.

PostgreSQL[5] é um poderoso banco de dados relacional de código aberto que usa e estende a linguagem SQL combinada com muitos recursos que armazenam e escalam com segurança as cargas de trabalho de dados mais complicadas.

2.1.2 Extração de metadados com JDBC

Além de recuperar os dados reais armazenados nas tabelas de um banco de dados, você também pode ler metadados, os quais possuem informações sobre os dados do banco, como nomes de tabelas, nomes de colunas, tipos de colunas, tamanhos de colunas dentre outros recursos.

A leitura desses metadados podem ser feitas utilizando a Linguagem de Consulta Estruturada (SQL) para bancos de dados relacionais, ou conforme nossa proposta, utilizando a biblioteca JDBC disponível no Kit de Desenvolvimento Java (JDK). Para tanto, após uma conexão ser estabelecida com o banco de dados, você

[5] Mais informações em: https://www.postgresql.org/about

pode recuperar metadados sobre o mesmo invocando o método getMetaData() no objeto Connection.

Figura 3 – Leitura de metadados.

```
Connection connection = DriverManager.getConnection(url, user, password);
DatabaseMetaData databaseMetaData = connection.getMetaData();
```

Fonte: Do autor.

O método getMetaData() exemplificado na Figura 3 retorna um objeto que implementa a interface DatabaseMetaData[6] que fornece uma variedade de métodos para ler informações abrangentes sobre o banco de dados como um todo.

2.2 GERAÇÃO DE CÓDIGO-FONTE

A engenharia de software é uma disciplina de engenharia cujo objetivo é o desenvolvimento econômico de sistemas de software. Atualmente, a geração automática de código desempenha um papel importante no desenvolvimento de software, pois ajuda a economizar tempo e facilita o uso de ferramentas e linguagens de programação. No entanto, também é importante que as ferramentas atendam a um conjunto de características de qualidade para que o desenvolvedor possa se sentir confiante ao usá-las. Todas essas abordagens, paradigmas de programação, ferramentas e técnicas facilitam o desenvolvimento de software e, portanto, melhoram a lucratividade dos sistemas de software (ROSALES-MORALES et al., 2015, tradução nossa).

Em plataformas de desenvolvimento com foco na geração de código, abordagens modulares de Low-Code permitem que desenvolvedores de software criem aplicações rapidamente, liberando-os da necessidade de escrever código linha por linha. Próximo a isso, o No-Code permite que analistas de negócios, administradores de escritórios e outros profissionais com perfil técnico possam criar aplicações com pouco ou nenhum conhecimento das linguagens de programação tradicionais.

Normalmente plataformas No-Code e Low-Code fornecem um ambiente de interação por meio da interface gráfica de usuário, a qual proporciona ao mesmo o controle das configurações e comportamentos da aplicação desenvolvida.

[6] Documentação: https://docs.oracle.com/javase/8/docs/api/java/sql/DatabaseMetaData.html

2.2.1 Framework

Para desenvolver a ferramenta de geração de código-fonte, bem como disponibilizar o artefato RESTful Web Service gerado em si, foi utilizado o Spring Framework.

O Spring[7] torna a programação em Java mais rápida, fácil e segura para todos. O foco do Spring em velocidade, simplicidade e produtividade o tornou o framework Java mais popular do mundo.

Criando uma estrutura de projetos padrão do Apache Maven[8] e adicionando as principais dependências junto ao Spring Framework, foi possível iniciar o desenvolvimento do gerador de código.

2.2.2 Dependências

No decorrer do desenvolvimento, novas dependências foram adicionadas ao projeto para que pudessem ser atendidas todas as especificidades com relação aos objetivos e comportamentos planejados para a plataforma.

Além da estrutura inicial do projeto, foram adicionadas dependências para controle e compressão de arquivo zip4j[9], geração de arquivos de origem .java com JavaPoet[10] e o conjunto de bibliotecas Springfox[11] que tem como objetivo automatizar a geração de especificações para APIs JSON.

2.2.3 JavaPoet

Com as informações obtidas por meio da extração de metadados com JDBC, e tendo conhecimento de toda a estrutura do projeto necessária para disponibilização de um artefato como resultado, foi iniciado o processo de codificação que efetuará a geração automática dos arquivos .java. Com a biblioteca open-source JavaPoet foi possível criar especificações que pudessem flexibilizar a geração de

[7] Mais informações em: https://spring.io/why-spring
[8] Mais informações em: https://maven.apache.org
[9] Repositório: https://github.com/srikanth-lingala/zip4j
[10] Repositório: https://github.com/square/javapoet
[11] Repositório: https://github.com/springfox/springfox

classes, funções e blocos de código conforme os princípios básicos necessários para se disponibilizar um Web Service.

Para atender esses princípios, foi criada a interface MdaGenerator com o método generate passando como parâmetro os metadados obtidos e implementando as classes que seriam geradas sendo elas, EntityGenerator, RepositoryGenerator, ServiceGenerator e ControllerGenerator.

Nesta etapa, além do conhecimento e estudo da biblioteca JavaPoet, foi fundamental a compreensão do conjunto de anotações, extensões e encapsulamentos de classes que estão presentes no Spring para disponibilização do Web Service como artefato que será gerado, mais precisamente uma API RESTful, bem como a documentação de especificações deste mesmo artefato com o uso do conjunto de bibliotecas Springfox.

Ao final, foram disponibilizados dois endpoints na API de geração de código, sendo elas /extract-metadata, responsável por receber a solicitação de criação de um novo artefato com as credenciais de conexão do banco de dados que é utilizada para extração das informações, e o endpoint /generate, que permite a edição e flexibilização dos dados obtidos, bem como uma validação do conteúdo que é disponibilizado para download em formato .zip para o solicitante.

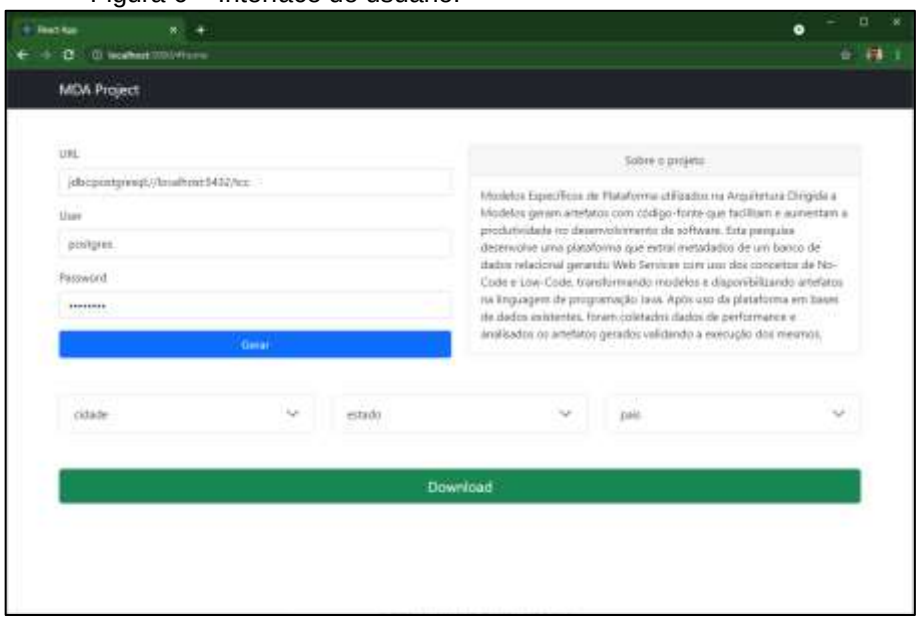
2.2.4 Interface do usuário

Com o intuito de fornecer uma interface de usuário amigável para a plataforma, foi utilizada a biblioteca JavaScript React[12] para criar uma aplicação web que se comunicará diretamente com a API de geração de código utilizando os endpoints citados anteriormente.

Na Figura 6 foi disponibilizado um formulário para adicionar as configurações do projeto que será criado, bem como as credenciais de acesso ao banco de dados. Após submissão do formulário, é possível obter a estrutura de dados relacionada a extração dos metadados que fica disponível para possível customização e download do artefato no botão ao final da página.

[12] Documentação: https://pt-br.reactjs.org/docs/getting-started.html

Figura 6 – Interface do usuário.



Fonte: Do autor.

2.3 API RESTFUL

Segundo Myers; Stylos (2016, tradução nossa) a Interface de Programação de Aplicações (API) expõe serviços e dados fornecidos por uma aplicação empregando um conjunto de recursos pré-definidos com métodos e objetos. O sucesso da API depende crucialmente de quão bem sua documentação atende às necessidades de informação dos desenvolvedores de software. Ao usar esses recursos, outras aplicações podem acessar os dados ou serviços sem ter que implementar os objetos e procedimentos subjacentes.

APIs são o centro para muitas arquiteturas de software modernas, pois fornecem abstrações de alto nível que facilitam as tarefas de programação, suportam a arquitetura de aplicações de software distribuídos e modulares bem como sua reutilização de código (ROBILLARD, 2009, tradução nossa).

Contemplado em uma API o termo RESTful é como citar a didática do termo orientado a objetos, ou seja, entende-se que uma linguagem, estrutura ou uma aplicação pode ser projetada de forma orientada a objetos, mas isso não a torna uma arquitetura orientada a objetos. Muitos Web Services usam HTTP, mas o usam de maneiras diferentes. Uma solicitação para um Web Service RESTful coloca as informações de execução no método HTTP e as informações de escopo na URI. Um

serviço RESTful orientado a recursos expõe uma URI para cada composição de dados que se deseja operar (BÖCK, 2012, tradução nossa).

2.3.1 Aplicação gerada

O artefato gerado pela plataforma resultou na API RESTful com as configurações pré-definidas pelo usuário solicitante, tanto quanto possível pela flexibilidade de personalização e uso do Spring Framework como enfoque no aumento da produtividade oferecido pelo mesmo dentro da linguagem de programação Java. O pattern disponibilizado pelo Spring, junto ao uso de anotações no código fonte gerado, tais como, @Entity, @Repository, @Service, @RestController, entre outras, possibilitou uma redução drástica de código boilerplate.

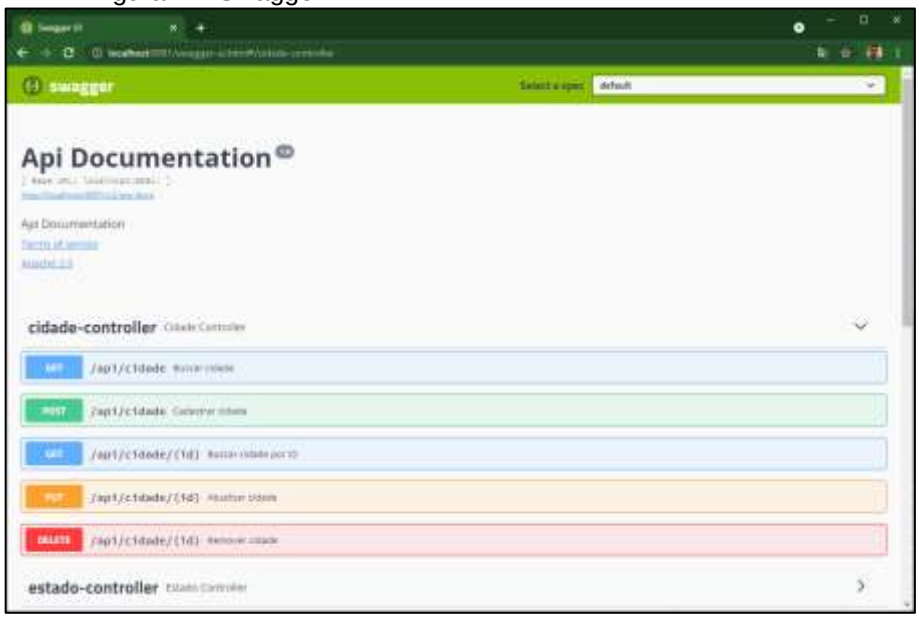
Ao descomprimir o conteúdo de download disponibilizado pela plataforma, e tendo um ambiente Java configurado é possível executar o artefato por meio de linhas de comando ou usando uma IDE com suporte a linguagem de programação Java. A execução e publicação do artefato gerado remete ao conceito No-Code, ou seja, não houve alteração do código gerado e o mesmo pode ser submetido ao uso em ambiente de homologação e produção. Caso seja necessária a adição de novas regras de negócio é possível editar o código-fonte gerado para que sejam atendidas as necessidades e comportamentos desejados antes da disponibilização da aplicação, tornando o conceito de Low-Code o resultado desse processo.

2.3.2 Documentação

Com o intuito de facilitar a compreensão e uso da API gerada pela plataforma e que se comunicará com outras aplicações foi disponibilizada toda a documentação referente à conversão dos modelos extraídos do banco de dados, bem como a exemplificação de uso dos métodos HTTP para consumo dos recursos. Para isso, conforme abordado no capítulo de geração de código-fonte, foi utilizado o conjunto de bibliotecas Springfox.

Na Figura 7 podemos observar a documentação da API, com os respectivos endpoints e modelos disponíveis para consumo acessível em /swagger-ui.html.

Figura 7 – Swagger.



Fonte: Do autor.

A documentação é gerada para o Swagger[13], um framework composto por diversas ferramentas que, independentemente da linguagem, auxilia a descrição, consumo e visualização de serviços de uma API REST.

3 RESULTADOS E DISCUSSÃO

Os resultados obtidos nesta pesquisa originaram-se da análise da plataforma desenvolvida mediante a geração de um artefato a partir de uma base de dados já consolidada. Sendo assim, foram considerados o número de entidades e atributos mapeados pela plataforma, bem como o tempo de geração das classes relacionadas aos modelos após sua conversão.

Conforme Tabela 1, foram processados um total de 3 entidades, com 19 atributos em um tempo total de 51 milissegundos. Esses valores podem variar dependendo dos recursos de hardware disponíveis para processamento por parte da plataforma.

[13] Mais informações em: https://swagger.io

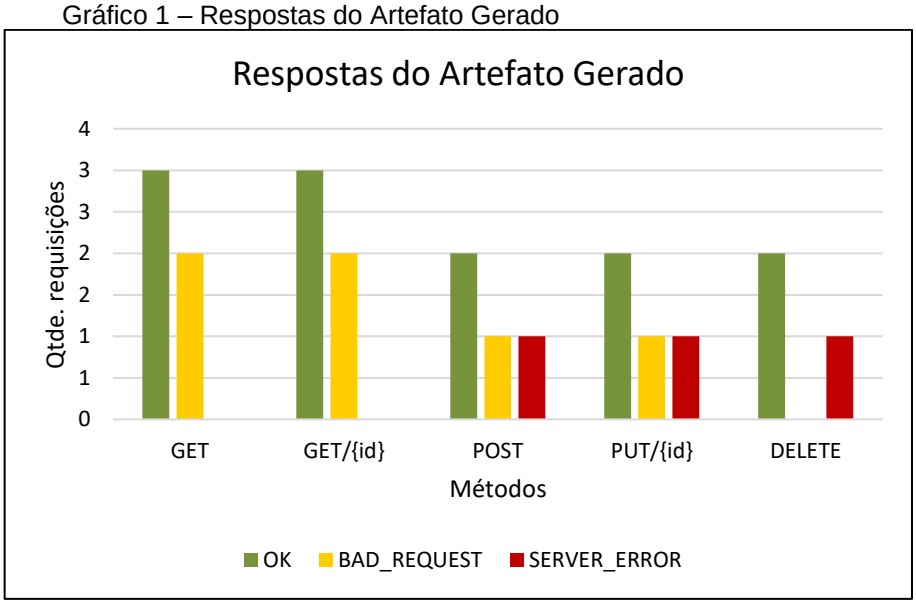
Tabela 1 – Tempo de processamento dos modelos.

Entidades	Atributos (un.)	Model (ms)	Resitory (ms)	Service (ms)	Controller (ms)	Total (ms)
Pais	5	4	5	1	3	13
Estado	6	4	2	3	3	12
Cidade	8	17	3	2	4	26

Fonte: Elaborado pelo autor.

Validou-se também a execução do Web Service como artefato gerado, para isso, foram aplicados testes que evidenciam através do código de status do protocolo HTTP quais requisições obtiveram sucesso, e quais apresentaram erros, sejam eles de integridade do banco de dados mapeado, de regras de negócio já previamente implementadas ou requisições com erros não tratados.

Gráfico 1 – Respostas do Artefato Gerado



Fonte: Elaborado pelo autor.

No Gráfico 1 os resultados obtidos foram de maior sucesso nos métodos GET, o qual apresenta uma listagem dos dados disponíveis, seguido do método GET/{id}, responsável por obter um registro específico da base de dados fazendo uso de sua chave primária. Os métodos POST, PUT/{id} e DELETE que respectivamente registram, atualizam e removem dados tiveram uma taxa menor de sucesso devido à regras de integridade aplicadas a nível de banco de dados.

Isso demonstra que o conceito de No-Code é mais efetivo na obtenção de dados envolvendo consultas que não envolvem transações. Outro fato relevante é que a criação de novos registros bem como a remoção podem ser impactados diretamente

por chaves estrangeiras de outras entidades que se relacionam diretamente com a entidade a qual teve seu modelo convertido e código-fonte gerado.

A aplicação do conceito de Low-Code se mostrou mais efetiva nos métodos transacionais como POST, PUT e DELETE que tiveram a necessidade de ajuste do código gerado na tipagem de dados e na criação de enumerados para representação de dados simples como identificadores numéricos e de caracteres como 1 e 0 (verdadeiro e falso) ou S e N (sim e não) por exemplo.

Uma abordagem de Model-Driven Architecture (MDA) também é proposta no artigo realizado na Universidade Pública de Pesquisa em L'Aquila, Itália e publicado na Computers 2020, que simplifica a modelagem, o projeto, a implementação e a integração de aplicações ao definir o software principalmente no nível do modelo. Adota-se a Unified Modeling Language (UML), como linguagem de modelagem. A tradução automática de diagramas UML para o código orientado a objetos é altamente desejável porque elimina as chances de introdução de erros humanos. Além disso, a geração automática de código ajuda os arquitetos de software a entregar o software no prazo. Nessa abordagem, as transformações automáticas em todos os níveis do MDA são baseadas em metamodelos para duas das construções mais importantes da UML, ou seja, casos de uso e classes. Uma ferramenta própria chamada xGenerator realiza as transformações até o código-fonte Java. A arquitetura das aplicações Web geradas respeita uma variante do conhecido padrão Model-View-Controller (MVC). O trabalho conclui uma longa pesquisa, na qual implementação e validação principalmente de regras de negócio para aplicações Web voltada para bancos foram reunidas. (PAOLONE et al., 2020, tradução nossa).

O artigo publicado na Cluster Computing em 2019 aponta que tendências recentes no ciclo de vida de desenvolvimento de software tratam muito de processos automáticos que levam à redução de tempo e custo. Na Arquitetura Dirigida a Modelos (MDA), os modelos de linguagem de modelagem unificada (UML) são a espinha dorsal de qualquer software em desenvolvimento. A transformação do modelo lógico de previsão proposta, automatiza a transformação de dois modelos do diagrama de sequência. Cada modelo carrega as mesmas informações dos outros modelos em aspectos diferentes para visualizar as restrições de requisitos em dimensões diferentes. Para tirar proveito disso, o diagrama de sequência foi considerado na abordagem MDA para gerar outros modelos automaticamente. Informações como elementos, atributos, relacionamentos do diagrama de sequência são extraídas

usando o analisador de modelo de objeto XML. As informações extraídas do diagrama de sequência combinadas com as regras de predição lógicas pré definidas, geram os elementos e relacionamentos de outros modelos. O resultado das informações transformadas é processado pela estrutura PlantUML para produzir o modelo desejado (MYTHILY; VALARMATHI; DURAI, 2019, tradução nossa).

O artigo realizado na Universidade Federal de São Carlos e publicado em 2020 no Software Quality Journal, apresenta uma alternativa para realizar uma Modernização Orientada à Modularidade (MOM), cujo objetivo é reestruturar características modulares ruins, conhecidas como uma deficiência de sistemas legados. Isso pode ser alcançado com o uso da Modernização Dirigida a Arquitetura (ADM), onde em seu processo de reengenharia são considerados o uso dos metamodelos e conceitos do MDA. O ADM também oferece um metamodelo denominado Structured Metrics Metamodel (SMM), cujo objetivo é padronizar a especificação de métricas, processos de mensuração e os resultados das análises. As métricas são mais difíceis de se especificar porque medem elementos que não estão presentes no código-fonte. De um ponto de vista mais técnico, é fornecida uma ferramenta com suporte de biblioteca que pode ser usada para mensurar a disseminação de características em sistemas representados por KDM. A biblioteca desenvolvida é especificada em SMM e pode ser facilmente reutilizada por ferramentas baseadas em ADM. A abordagem é útil em cenários de modernização para melhorar a modularização de sistemas de software, por exemplo, um processo de modernização que aplica refatorações a fim de modularizar o código de persistência que está espalhado entre funções, componentes e linhas de código (MARTÍN et al., 2020, tradução nossa).

O estudo realizado na Universidade Tecnológica Federal do Paraná e publicado em 2012 no Journal of Computer Science 8, fornece uma visão geral sobre o refinamento de modelo e investiga duas abordagens para o refinamento baseados na Atlas Transformation Language (ATL), conhecido como modo de refinamento e sobreposição de módulo. ATL é uma linguagem amplamente adotada para resolver problemas de transformação de modelo na abordagem MDA. O estudo apresenta os resultados comparativos obtidos a partir da análise das abordagens, enfatizando os benefícios de sua aplicação. Ao final conclui-se que o uso crescente do MDA para o projeto de sistemas de software possibilitou pesquisas sobre como os desenvolvedores podem se beneficiar de abordagens que realizam o refinamento do

modelo. As principais vantagens apontadas pelo Modo Refino são a facilidade de uso e rapidez de execução. Por outro lado, esta técnica possui restrições severas, tais como, incompatibilidade com blocos de ação e de regras. Essas restrições muitas vezes dificultam o desenvolvimento de transformações de modelo mais complexas (SOARES et al., 2012, tradução nossa).

A pesquisa publicada em 2013 no Journal of Systems and Software, feita na Universidade Estadual do Centro Oeste e na Universidade Tecnológica Federal, ambas no Paraná, traz resultados estatísticos sobre o uso da modelagem UML e abordagens baseadas em modelos para projetos de software embarcados no Brasil. A pesquisa fornece evidências sobre a maturidade do uso da UML e das abordagens orientadas a modelos, como são empregadas, quais e onde estão os profissionais que as utilizam. Aspectos técnicos, organizacionais e sociais foram investigados e documentados por meio de um método descritivo de pesquisa. Tais aspectos parecem refletir as opiniões dos engenheiros de software sobre como eles percebem o impacto do uso de UML e abordagens orientadas por modelo na produtividade e qualidade no desenvolvimento de software embarcado. Os resultados mostram que a maioria dos participantes está claramente ciente do valor da abordagem de modelagem, embora a pratique apenas em um grau limitado. A maioria dos entrevistados que faz uso de abordagens baseadas em modelos atesta que a produtividade e a portabilidade são as principais vantagens de seu uso. O artigo reúne respostas que ilustram aspectos derivados da experiência prática de engenheiros de software que participaram de uma pesquisa envolvendo 209 desenvolvedores brasileiros de software embarcado (AGNER et al., 2013, tradução nossa).

Em todas as pesquisas citadas acima, os autores utilizaram a Arquitetura Dirigida a Modelos como forma de otimizar processos para que pudessem reduzir o tempo de desenvolvimento e a complexidade das soluções desenvolvidas, usando em maior parte a linguagem de modelagem unificada (UML) como fonte de extração dos metadados.

4 CONCLUSÃO

Nesta pesquisa desenvolveu-se uma plataforma aplicando o uso do Modelo Específico de Plataforma (PSM) da Arquitetura Dirigida a Modelos como forma de obtenção do código-fonte para geração de Web Services na linguagem de

programação Java. A transformação entre os modelos partiu da extração dos metadados de uma base de dados já consolidada, seguido da programação de especificações e uso de biblioteca de terceiros para geração e disponibilização dos artefatos pela plataforma.

Os conceitos de No-Code e Low-Code junto a arquitetura dirigida a modelos se mostraram úteis em automatizar o processo de geração de código-fonte, principalmente em suas abordagens de condicionamento das estruturas geradas. É importante ressaltar que apesar da seleção específica de uso da linguagem de programação Java, bem como de Web Services como artefato de resultado, o uso dos conceitos e da arquitetura podem ser aplicados em qualquer linguagem ou artefato de destino.

Quanto aos resultados obtidos foi possível efetuar a conversão e geração de código-fonte de todas as bases de dados disponibilizadas até o momento, indicando um resultado satisfatório. Para avaliação de uso da plataforma desenvolvida, foram coletadas informações relacionadas a quantidade de modelos e atributos mapeados, bem como o tempo de geração das classes definidas para execução do artefato usando uma base de dados específica. Após disponibilização do Web Service, o mesmo foi submetido a testes de requisição, validando os códigos de status e retornos gerados pela API RESTful.

Em pesquisas futuras recomenda-se o uso dos conceitos de No-Code e Low-Code junto em uma nova arquitetura, obtendo como resultado de artefato uma linguagem ou plataforma distinta da abordagem utilizada nesta pesquisa, proporcionando até mesmo a possibilidade de comparação entre as plataformas. Orienta-se também o uso da Inteligência artificial na geração de código-fonte empregando os conceitos abordados neste artigo.

REFERÊNCIAS

AGNER, Luciane Telinski Wiedermann et al. **A Brazilian survey on UML and model-driven practices for embedded software development**. Journal of Systems and Software, v. 86, n. 4, p. 997–1005, 2013. Disponível em: <https://doi.org/10.1016/j.jss.2012.11.023>. Acesso em: 19 Jun. 2021.

BÖCK, Heiko. **RESTful Web Services**, 2012. Disponível em: <https://doi.org/10.1007/978-1-4302-4102-7_32>. Acesso em: 19 Jun. 2021.

CHEN, Peter Pin Shan. **The entity-relationship model: Toward a unified view of data**. ACM SIGIR Forum, v. 10, n. 3, p. 9, 1975. Disponível em: <https://doi.org/10.1145/1095277.1095279>. Acesso em: 19 Jun. 2021.

DUBY, CK. **Accelerating Embedded Software Development with a Model Driven Architecture®**. Pahtfinder Solutions, 2003. Disponível em: <http://omg.net/mda/mda_files/MDA_overview.pdf>. Acesso em: 19 Jun. 2021.

FAVRE, Liliana María. **Model Driven Architecture (MDA)**, n. June, p. 15–33, 2010. Disponível em: <https://doi.org/10.4018/978-1-61520-649-0.ch002>. Acesso em: 19 Jun. 2021.

GONZÁLEZ JIMÉNEZ, Luis. **REERM: Reenhancing the entity-relationship model**. Data and Knowledge Engineering, v. 58, n. 3, p. 410–435, 2006. Disponível em: <https://doi.org/10.1016/j.datak.2005.05.004>. Acesso em: 19 Jun. 2021.

HUANG, Yen Chieh; CHU, Chih Ping. **Developing web applications based on model driven architecture**. International Journal of Software Engineering and Knowledge Engineering, v. 24, n. 2, p. 163–182, 2014. Disponível em: <https://doi.org/10.1142/S0218194014500077>. Acesso em: 19 Jun. 2021.

MARTÍN, Daniel San et al. **Specification and use of concern metrics for supporting modularity-oriented modernizations**. Software Quality Journal, v. 28, n. 3, p. 1087–1111, 2020. Disponível em: <https://doi.org/10.1007/s11219-020-09528-9>. Acesso em: 19 Jun. 2021.

MOREIRA, E.; MRACK, P.. **Sistemas Dinâmicos Baseados em Metamodelos**. In II Workshop de Computação e Gestão da Informação (WCOMPI2003), 2, 2003, Santa Cruz do Sul. Disponível em: <http://3layer.com.br/confluence/download/attachments/3571742/sistemasDinamicosEmTempoDeExecucao.pdf>. Acesso em: 19 Jun. 2021.

MYERS, Brad A.; STYLOS, Jeffrey. **Improving API usability.** Communications of the ACM, v. 59, n. 6, p. 62–69, 2016. Disponível em: <https://doi.org/10.1145/2896587>. Acesso em: 19 Jun. 2021.

MYTHILY, M.; VALARMATHI, M. L.; DURAI, C. Anand Deva. **Model transformation using logical prediction from sequence diagram: an experimental approach**. Cluster Computing, v. 22, n. s5, p. 12351–12362, 2019. Disponível em: <https://doi.org/10.1007/s10586-017-1618-5>. Acesso em: 19 Jun. 2021.

OBJECT MANAGEMENT GROUP (OMG). **The MDA Foundation Model**. SparxSystems, p. 1–9, 2010. Disponível em: <http://www.omg.org/cgi-bin/doc?ormsc/10-09-06.pdf>. Acesso em: 19 Jun. 2021.

PAOLONE, Gaetanino et al. **Automatic code generation of mvc web applications**. Computers, v. 9, n. 3, p. 1–29, 2020. Disponível em: <https://doi.org/10.3390/computers9030056>. Acesso em: 19 Jun. 2021.

ROBILLARD, Martin P. **What makes APIs hard to learn? answers from developers**. IEEE Software, v. 26, n. 6, p. 27–34, 2009. Disponível em: <https://doi.org/10.1109/MS.2009.193>. Acesso em: 19 Jun. 2021.

ROSALES-MORALES, Viviana Yarel et al. **An analysis of tools for automatic software development and automatic code generation**. Revista Facultad de Ingenieria, v. 2015, n. 77, p. 75–87, 2015. Disponível em: <https://doi.org/10.17533/udea.redin.n77a10>. Acesso em: 19 Jun. 2021.

SIEGEL, Jon; GROUP, the OMG Staff Strategy. **Developing in OMG's Model-Driven Architecture**. Management, v. 6, p. 1–12, 2001. Disponível em: <https://www.omg.org/cgi-bin/doc?omg/2001-12-01>. Acesso em: 19 Jun. 2021.

SOARES, Inali Wisniewski et al. **Model Refinement in the Model Driven Architecture Context**. Journal of Computer Science, v. 8, n. 8, p. 1205–1211, 2012. Disponível em: <https://doi.org/10.3844/jcssp.2012.1205.1211>. Acesso em: 19 Jun. 2021.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed, p.18-281, São Paulo: Pearson Addison Wesley, 2007.

TRUYEN, Frank. **The Fast Guide to Model Driven Architecture The Basics of Model Driven Architecture**. Cephas Consulting Corp, p. 12, 2006. Disponível em: <http://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf>. Acesso em: 19 Jun. 2021.

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação**. 3.ed. Rio de Janeiro: LTC, 2021.