UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
INSTITUTO DE GEOCIÊNCIAS E CIÊNCIAS EXATAS – RIO CLARO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DYNASCHEMA: UMA BIBLIOTECA PARA EVOLUÇÃO DE BANCO DE DADOS RELACIONAL PARA O DOMÍNIO DE SOFTWARE AUTOADAPTATIVO

GABRIEL NAGASSAKI CAMPOS

RIO CLARO-SP
2024

UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

GABRIEL NAGASSAKI CAMPOS

DynaSchema: Uma Biblioteca para Evolução de Banco de Dados Relacional para o Domínio de Software Autoadaptativo

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Frank José Affonso

Rio Claro – SP

2024

C198d

Campos, Gabriel Nagassaki
DynaSchema : uma biblioteca para evolução de banco de dados relacional para o domínio de software autoadaptativo / Gabriel Nagassaki Campos. -- Rio Claro, 2024
152 f.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Geociências e Ciências Exatas, Rio Claro
Orientador: Frank José Affonso

1. Evolução do esquema de dados. 2. Software autoadaptativo. 3. Biblioteca. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Geociências e Ciências Exatas, Rio Claro. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

GABRIEL NAGASSAKI CAMPOS

DynaSchema: Uma Biblioteca para Evolução de Banco de Dados Relacional para o Domínio de Software Autoadaptativo

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Comissão Examinadora

- Prof. Dr. Frank José Affonso (Orientador)
  Departamento de Estatística, Matemática Aplicada e Computação (DEMAC)
  Universidade Estadual Paulista "Júlio De Mesquita Filho" (UNESP) – Câmpus de Rio Claro – SP
- Profa. Dra. Marcela Xavier Ribeiro
  Departamento de Computação
  Universidade Federal de São Carlos – SP
- Prof. Dr. Rogério Eduardo Garcia
  Departamento de Matemática e Computação
  Universidade Estadual Paulista "Júlio De Mesquita Filho" (UNESP) – Câmpus de Presidente Prudente – SP

Conceito: Aprovado.

Rio Claro (SP), 27 de março de 2024.

AGRADECIMENTOS

Agradeço a Deus por todas as graças conquistadas ao longo da vida, que inclui esta jornada de mestrado acadêmico. Também agradeço à intercessão de Nossa Senhora, Santa Mãe de Deus, que atende de maneira misericordiosa a todas as nossas suplicas.

Agradeço a minha família por encorajar minha carreira acadêmica, provendo todo o suporte necessário para minha dedicação exclusiva aos estudos.

Agradeço ao Prof. Dr. Frank José Afonso, pela parceria, paciência e sabedoria em orientar minha formação acadêmica, desde a iniciação científica, graduação, e agora, a pós-graduação.

Agradeço aos membros da banca examinadora pela avaliação cuidadosa e pelos comentários valiosos que contribuíram com este projeto de mestrado acadêmico.

Agradeço a todos os colaboradores da Universidade Estatual Paulista "Júlio de Mesquita Filho" que colaboraram direta ou indiretamente com minha formação acadêmica, desde a graduação até a pós-graduação.

Agradeço pelo apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES N. 88887.670715/2022-00) na pós-graduação, além do Projeto LINEAS – Lineamentos e Zonas de Fraturas Continentais: Possíveis Corredores de Deformação da Bacia de Santos e seu Embasamento (FUNDUNESP N. 2700/2017) e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP N. 2019/21510-3) na graduação.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Vários problemas podem surgir quando a adaptação de software é realizada de maneira manual, principalmente com relação à inserção involuntária de incertezas por parte dos desenvolvedores. Portanto, a adaptação manual é um tipo de atividade propensa a erros que pode consumir muito tempo, esforço e dinheiro. Nesse contexto, o Software Autoadaptativo (em inglês, *Self-adaptive Software* – SaS) é um tipo especial de sistema de software que é capaz de se modificar em tempo de execução para lidar com as mudanças no seu ambiente operacional. De acordo com essa característica, o SaS pode ser adotado com uma solução para lidar com a automação da adaptação de software ao mesmo tempo que minimiza o envolvimento dos desenvolvedores. Na literatura científica podem ser encontrados vários exemplos de SaS relacionados com diferentes domínios de software, porém os tipos de SaS associados com a persistência de dados ainda são pouco investigados. Tais tipos de SaS abordam um cenário de execução em que os ciclos de autoadaptação estrutural do sistema quebram a compatibilidade com o esquema de banco de dados que foi estabelecido na etapa de projeto. Tendo em vista esse problema, foi conduzido um mapeamento sistemático da literatura (em inglês, *Systematic Mapping Study* – SMS) para estabelecer um panorama abrangente sobre a evolução do esquema de dados no domínio de SaS. A partir dos resultados desse SMS, foram levantadas as evidências sobre a importância desse tema de pesquisa, porém não aplicadas de maneira direta no domínio de SaS. Assim, o principal objetivo deste projeto de mestrado acadêmico é contribuir com a área de SaS através da proposição da biblioteca DynaSchema, que consiste em uma nova solução para lidar com a evolução do esquema de dados nesse domínio de software. Em síntese, essa solução atua como uma camada intermediária entre o SaS e seu banco de dados relacional, permitindo que o SaS possa adaptar seu esquema de dados ao mesmo tempo que a biblioteca gerencia as mudanças necessárias no banco de dados. Além disso, vale destacar que o SMS conduzido nesta dissertação de mestrado pode ser utilizado como um norte para o desenvolvimento de soluções que lidam com a evolução do esquema de dados. Por fim, foi elaborado um estudo de caso para avaliar a biblioteca DynaSchema, onde foram explorados cenários de adaptação que lidam com a evolução do esquema de dados do SaS.

Palavras-chave: evolução do esquema de dados; software autoadaptativo; biblioteca; dynaschema.

ABSTRACT

Several problems can arise when software adaptation is performed manually, mainly regarding developers' involuntary insertion of uncertainties. Therefore, manual adaptation is an error-prone activity that consumes time, effort, and money. In this context, Self-adaptive Software (SaS) is a special type of software system capable of modifying itself at runtime to deal with changes in its operating environment. According to this feature, SaS can be adopted as a solution to handle the automation of software adaptation while minimizing developer involvement. Several examples of SaS related to different software domains can be found in the scientific literature, but the SaS types associated with data persistence are still slightly investigated. Such systems address an execution scenario in which the system's structural self-adaptation cycles break up compatibility with the database schema established in the design stage. Because of this problem, a Systematic Mapping Study (SMS) was conducted to establish a comprehensive overview of the evolution of the data schema in the SaS domain. Based on the results of this SMS, evidence was retrieved about the importance of this research topic, but not directly applied in the SaS domain. Thus, the main goal of this academic master's project is to contribute to the SaS area by proposing the DynaSchema library, which consists of a new solution to deal with the evolution of the data schema in this software domain. In short, this solution acts as an intermediate layer between SaS and its relational database, enabling SaS to adapt its data schema at the same time as the library manages the necessary changes in the database. Furthermore, it is worth highlighting that the SMS conducted in this master's dissertation can be used as a guide to support the development of solutions that deal with the evolution of the data schema. Finally, a case study was developed to evaluate the DynaSchema library, where adaptation scenarios that deal with the evolution of the SaS data schema were explored.

Keywords: data schema evolution; self-adaptive software; library; dynaschema.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CEP	Código de Endereçamento Postal
CE	Critério de Exclusão
CI	Critério de Inclusão
CNPJ	Cadastro Nacional da Pessoa Jurídica
CPF	Cadastro de Pessoa Física
DDL	*Data Definition Language*
DSLModeler4SaS	*Domain-Specific Language Modeler for Self-adaptive Software*
DSL	*Domain-Specific Language*
eLanguage	*entity Language*
ETL	*Extraction, Transformation and Loading*
FB	Frente de Busca
IBM	*International Business Machines*
IE	Inscrição Estadual
JDK	*Java Development Kit*
JPA	*Java Persistence API*
MAPE-K	*Monitor, Analyze, Plan, Execute over Knowledge*
MAPE	*Monitor, Analyze, Plan, Execute*
OLAP	*On-Line Analytical Processing*
ORM	*Object-Relational Mapping*
QP	Questão de Pesquisa
RA4SaS	*Reference Architecture for Self-adaptive Software*
RA4SelfMobApps	*Reference Architecture for Self-adaptive Service-oriented Mobile Applications*
SAS	*Self-adaptive Software*
SDL	*Storage Definition Language*
SGBD	*Sistema Gerenciador de Banco de Dados*
SMS	*Systematic Mapping Study*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
VDL	*View Definition Language*
XML	*eXtensible Markup Language*
ZDT	*Zero-Downtime*

SUMÁRIO

1 Introdução

1.1 Contextualização

Os sistemas de software têm desempenhado um papel fundamental na nossa sociedade, sendo responsáveis pela automatização da execução de tarefas em diferentes segmentos, a saber: instituições púbicas, privadas e financeiras; sistemas de comunicação; aeroportos; meios de entretenimento; entre outras áreas de aplicação. Além disso, tais sistemas têm sido projetados para atuar em situações adversas e/ou sujeitas às mudanças, permitindo a manutenção da sua disponibilidade de execução em 24/7 (ou seja, 24 horas por dia e 7 dias por semana), sem interrupções ou intervenções humanas. De acordo com essas demandas, é essencial que esses sistemas de software possuam a capacidade de lidar com as incertezas, cujas causas podem variar desde mudanças no seu ambiente operacional até em alterações nas necessidades de seus usuários. O Software Autoadaptativo (em inglês, *Self-adaptive Software* – SaS) é um tipo especial de sistema de software que é capaz de lidar com as mudanças em tempo de execução para atender as novas necessidades dos usuários ou reagir de forma autônoma às modificações em seu ambiente de execução (por exemplo, degradação de qualidade, sobrecarga de operações, entre outras adversidades) (CHENG *et al.*, 2009; LEMOS *et al.*, 2013; WEYNS, 2019).

Vários problemas podem surgir quando a adaptação de software é realizada de maneira manual, principalmente devido à inserção involuntária de incertezas por parte dos desenvolvedores (ou seja, dos seres humanos). Nesse sentido, pode-se dizer que a adaptação manual de software é uma atividade propensa a erros que pode consumir muito tempo, esforço e dinheiro (HILLENBRAND *et al.*, 2020). Tendo em vista esse cenário problemático, abordagens automatizadas de adaptação têm sido adotadas com uma alternativa viável para maximizar a rapidez de implementação do SaS e, em paralelo, minimizar o envolvimento dos seres humanos (CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009; LEMOS *et al.*, 2013). Por outro lado, a preservação da integridade entre os modelos gerados na fase de projeto e o código fonte do SaS durante seus ciclos de autoadaptação em tempo de execução ainda é um desafio que foi pouco investigado pela comunidade científica de SaS e pelos profissionais da indústria que trabalham com esse tipo de sistema de software (HILLENBRAND *et al.*, 2020; CHENG *et al.*, 2009; AFFONSO; PASSINI; NAKAGAWA, 2019; WEYNS, 2019; LEMOS *et al.*, 2013).

Uma iniciativa da comunidade de SaS tem mantido um repositório que é atualizado ao longo dos anos para fornecer exemplos, desafios e soluções para a área de SaS (VOGEL, 2023). Os exemplos de SaS contidos nesse repositório estão relacionados com diferentes domínios de software, a saber: serviços *web*, veículos autônomos, sistemas ciber-físicos, processamento de vídeo, entre outras áreas de aplicação. Embora esse repositório seja uma referência para a comunidade SaS, outros tipos desse sistema de software também podem ser encontrados na literatura. Como exemplo, esta dissertação de mestrado se concentra especificamente nos tipos de SaS que precisam armazenar suas informações em um banco de dados relacional. Nesse cenário, um SaS pode passar por ciclos de autoadaptação estrutural que quebram a compatibilidade com o esquema de banco de dados que foi estabelecido na sua etapa de projeto. Para lidar com esse problema de integridade, esse SaS deve apresentar um mecanismo para lidar com as diferentes versões do seu esquema de dados em tempo de execução (HILLENBRAND *et al.*, 2020; STÖRL; KLETTKE; SCHERZINGER, 2020). Portanto, esse mecanismo deve alterar o esquema de banco de dados para acompanhar os ciclos de autoadaptação do SaS ao longo do tempo.

1.2 Motivação

Existem vários estudos na literatura científica que exploram diferentes aspectos de um SaS. Tais estudos contextualizam os pontos principais que definem esse tipo de sistema de software (BRUN *et al.*, 2009; CHENG *et al.*, 2009; KRUPITZER *et al.*, 2015; SALEHIE; TAHVILDARI, 2009; WEYNS *et al.*, 2013), as propriedades de autoadaptação para resolver um problema específico (KRUPITZER *et al.*, 2015; SALEHIE; TAHVILDARI, 2009), as técnicas para o controle do processo de autoadaptação (BRUN *et al.*, 2009; CHENG *et al.*, 2009; SALEHIE; TAHVILDARI, 2009; WEYNS *et al.*, 2013), as questões de engenharia envolvidas no desenvolvimento de uma entidade de software (KRUPITZER *et al.*, 2015; SALEHIE; TAHVILDARI, 2009), entre outros assuntos.

Baseado no contexto exposto, Affonso e Nakagawa (2013) propuseram a arquitetura de referência RA4SaS (do inglês, *Reference Architecture for Self-adaptive Software*) para apoiar o desenvolvimento de um SaS. Essa arquitetura tem sido mantida pelo orientador deste projeto de mestrado acadêmico, servindo de base para desenvolvimento de outras pesquisas no domínio de SaS. Como exemplo, pode-se citar o *framework* para apoiar à tomada de decisão em um SaS (AFFONSO *et al.*, 2015), a ferramenta de modelagem para apoiar à tomada de decisão fornecida pelo *framework* anterior (AFFONSO; LEITE; NAKAGAWA, 2016), o gerador de código fonte baseado em *templates* (BENATO; AFFONSO; NAKAGAWA, 2017) e a arquitetura de referência para a criação de aplicativos móveis autoadaptativos orientados a serviços (AFFONSO; PASSINI; NAKAGAWA, 2019). O tema desta dissertação de mestrado está inserido no contexto da linha de pesquisa de aprimoramento da arquitetura de referência RA4SaS, cuja última evolução permitiu que o SaS possa raciocinar sobre a persistência dos seus dados. Nesse sentido, um SaS que utiliza a arquitetura RA4SaS está preparado para autoadaptar seu modelo de dados, porém o esquema do banco de dados não acompanha essas mudanças de maneira automática.

Motivado pela problemática apresentada nesta seção, pelo cenário exposto na Seção 1.1 e pela necessidade de se obter uma visão geral sobre a temática desta dissertação de mestrado, foi conduzida uma revisão da literatura sobre a evolução do esquema de dados para o domínio de SaS. Para isso, foi adotada a técnica de mapeamento sistemático (em inglês, *Systematic Mapping Study* – SMS), que permite uma avaliação mais completa e justa sobre um tópico de pesquisa através de uma metodologia confiável, rigorosa, auditável e isenta de influências pessoais dos pesquisadores (KITCHENHAM; DYBA; JORGENSEN, 2004; KITCHENHAM; CHARTERS, 2007; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Em relação aos interesses de investigação, esse SMS buscou traçar um panorama abrangente em relação aos atributos não funcionais, modelos de dados, estratégias de migração e aspectos de *design* que foram abordados no projeto das soluções que lidam com a evolução do esquema de dados.

Dentre os resultados desse SMS, vale destacar a seleção de 17 estudos primários (listados na Tabela 1 do Apêndice A) para sistematizar os conhecimentos sobre a temática de pesquisa explorado nesta dissertação de mestrado. A partir de uma análise geral, pode-se dizer que a evolução do esquema de dados no domínio de SaS é um tema amplo e precisa ser investigado com maior profundidade. Nesse sentido, o panorama reportado no SMS do Capítulo 3 apresenta uma importante contribuição para a comunidade de SaS e seus *stakeholders*. Embora os estudos selecionados nesse SMS apresentem um conjunto comum de características sobre diferentes aspectos de investigação (ou seja, domínios de aplicação, modelos de dados, estratégias de migração e atributos não funcionais), não foi observado um consenso sobre o que deve ser aceito como uma solução no contexto da evolução do esquema de dados no domínio de SaS.

Diante do conteúdo reportado nesta seção, a persistência de dados para o domínio de SaS e os problemas relacionadas à evolução do esquema de dados desse tipo de sistema são assuntos que vêm sendo investigados nos últimos anos e ainda possuem potencial de exploração pela comunidade científica. Nesse sentido, pode-se dizer que nenhuma das iniciativas, encontradas na investigação conduzida neste projeto de mestrado acadêmico, contempla totalmente as lacunas e temas em aberto que foram expostas nesta seção, a saber: (i) *a dependência entre a estrutura do SaS e o esquema de banco de dados*; (ii) *a evolução do esquema de banco de dados*; (iii) *a migração dos dados da versão antiga do esquema de baco de dados para a nova versão*; (iv) *a manutenção simultânea das versões antiga e nova do esquema de banco de dados em um estado transitório* e (v) *a capacidade de recuperação em relação a uma operação mal executada*. Diante do exposto, tais características foram consideradas como fatores motivadores para a proposição de uma nova solução para a evolução do esquema de dados no domínio de SaS.

1.3 Objetivos

De acordo com o cenário exposto nas Seções 1.1 e 1.2, este projeto de mestrado acadêmico tem o objetivo de desenvolver uma nova solução para apoiar o processo de evolução do esquema de dados no domínio de SaS. Com base nos resultados da investigação da literatura conduzida e reportada no Capítulo 3, foi proposta uma biblioteca que atua como uma camada intermediária entre o SaS e seu banco de dados relacional, permitindo que o SaS possa adaptar seu esquema de dados ao mesmo tempo que a biblioteca gerencia as mudanças que devem ser feitas no banco de dados para acompanhar essa adaptação. Vale salientar que essa biblioteca lida com um tipo de banco de dados relacional, que adere aos resultados anteriores de nosso grupo de pesquisa e fornece os recursos identificados na investigação conduzida nesta dissertação de mestrado. Além disso, vale destacar que essa biblioteca não abordou outros tipos de banco de dados devido aos níveis elevados de complexidade desse tipo de solução, além da restrição de tempo para a conclusão de um projeto de mestrado acadêmico.

Conforme reportado neste capítulo, foi conduzido uma revisão da literatura científica para assegurar a originalidade da solução proposta neste projeto de mestrado acadêmico e estabelecer uma visão geral do seu tema de pesquisa. Portanto, outro objetivo geral desta dissertação de mestrado consiste na apresentação de um panorama detalhado sobre o projeto de soluções que são capazes de lidar com a evolução do esquema de dados no domínio de SaS. Com base no conteúdo apresentado no Capítulo 3, acredita-se que os resultados dessa revisão possam nortear os pesquisadores e/ou os profissionais da indústria no desenvolvimento desse tipo de solução e/ou no projeto de novas soluções para a área de SaS ou outros domínios vizinhos.

1.4 Organização da Dissertação

Neste capítulo foi apresentado o contexto em que este projeto de mestrado acadêmico está inserido, as motivações para o desenvolvimento do trabalho proposto e seus objetivos principais. O conteúdo restante desta dissertação de mestrado foi organizado em cinco capítulos. Inicialmente, no Capítulo 2 são reportados os conceitos e as definições abordados neste projeto de mestrado. Em seguida, no Capítulo 3 é apresentada uma visão geral sobre a evolução do esquema de dados para o domínio de SaS através de uma revisão sistemática da literatura. No Capítulo 4 são abordados os detalhes de projeto e *design* da biblioteca DynaSchema, que consiste em uma nova solução para lidar com o processo da evolução do esquema de dados de um SaS. No Capítulo 5 foi conduzido um estudo de caso para avaliar os requisitos de projeto, as funcionalidades e o comportamento dessa biblioteca. Por fim, no Capítulo 6 são apresentadas as conclusões desta dissertação de mestrado e as perspectivas para trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos e definições envolvidos na elaboração da DynaSchema, a biblioteca desenvolvida neste projeto de mestrado acadêmico. Inicialmente, na Seção 2.1 é discutida a importância da arquitetura de software, sendo apresentados os princípios dos estilos e padrões arquiteturais, assim como os conceitos sobre arquitetura de referência. Em seguida, na Seção 2.2 são reportados os conceitos sobre os sistemas autoadaptativos, além dos pontos principais relacionados ao desenvolvimento desse tipo de sistema. Na Seção 2.3 é descrita a arquitetura de referência RA4SaS que contextualiza o domínio de aplicação onde a biblioteca DynaSchema está inserida. Na Seção 2.4 são relatados os conceitos envolvidos em uma abordagem de sistema de banco de dados para gerenciar as informações de um sistema de software. Por fim, na Seção 2.5 são apresentadas as considerações finais deste capítulo.

2.1 Arquitetura de Software

A **arquitetura de software** pode ser definida como uma descrição sobre a organização de um sistema computacional e as relações entre seus componentes de software (FOWLER, 2006; PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2019). É importante salientar que a arquitetura não é o software operacional, mas uma representação em alto nível que guia o desenvolvimento de um sistema computacional em vários aspectos (FOWLER, 2006; PRESSMAN; MAXIM, 2016). Além disso, essa representação em alto nível pode apresentar um aspecto subjetivo, pois a arquitetura reflete uma compreensão compartilhada entre os desenvolvedores de um mesmo projeto de software, cujas características devem atender às necessidades específicas do sistema computacional que está sendo projetado (FOWLER, 2006; PRESSMAN; MAXIM, 2016).

Uma arquitetura de software pode ser descrita através de um conjunto de artefatos que reflete algumas perspectivas sobre um sistema computacional (PRESSMAN; MAXIM, 2016). Tais artefatos podem fornecer diferentes visões sobre a arquitetura do sistema, como: (i) *a metáfora do esquema*, que descreve os componentes do sistema e as informações arquiteturais de maneira clara aos projetistas e engenheiros de software; (ii) *a metáfora da linguagem*, que facilita a comunicação entre os *stakeholders* envolvidos no projeto do sistema; (iii) *a metáfora da decisão*, que permite o balanceamento de diferentes fatores, como custo e desempenho, por exemplo; e (iv) *a metáfora da literatura*, que possibilita a transferência de conhecimento através da documentação das soluções arquiteturais construídas no passado. Vale ressaltar que existem várias maneiras de se especificar uma arquitetura de software, cujas partes importantes podem mudar durante o ciclo de vida do sistema (FOWLER, 2006).

O projeto de uma arquitetura de software tem o objetivo de compreender como o sistema computacional deve ser organizado e qual deve ser sua estrutura geral (SOMMERVILLE, 2019). Essa atividade envolve um processo criativo para atender os requisitos funcionais e não funcionais da aplicação. Infelizmente, não existe uma metodologia padronizada para a criação de um projeto arquitetural e sua elaboração depende do domínio em que o sistema de software está inserido, do nível de experiência dos arquitetos de sistema envolvidos no projeto e dos requisitos específicos da aplicação. Além disso, os estilos e padrões arquiteturais e as arquiteturas de referência podem ser utilizadas no projeto de uma arquitetura de software para descrever uma organização de modo previsível aos projetistas e engenheiros de software (PRESSMAN; MAXIM, 2016).

Os **estilos arquiteturais** podem ser utilizados para estabelecer uma estrutura global do sistema (PRESSMAN; MAXIM, 2016). Cada categoria de estilo é caracterizada por um conjunto de quatro aspectos, a saber: (i) um conjunto de componentes responsável por executar uma determinada funcionalidade do sistema; (ii) um conjunto de conectores que permite a comunicação, coordenação e cooperação entre esses componentes; (iii) as restrições de integração entre o sistema e seus componentes; e (iv) a semântica da organização do sistema através de suas partes constituintes para obter uma visão geral sobre as propriedades do estilo arquitetural.

A Arquitetura Modular é um exemplo de estilo arquitetural que divide os sistemas de software em módulos (PRESSMAN; MAXIM, 2016). Nesse estilo é adotado alguma estratégia ou padrão modular para a estruturação de uma aplicação, cuja noção intuitiva de organização já é historicamente utilizada por bons desenvolvedores de software. No contexto da biblioteca DynaSchema, foi possível dividir suas funcionalidades em componentes bem definidos, assim foi adotado um estilo arquitetural modular para estruturar sua organização interna. Vale salientar que a estrutura global de um sistema computacional não precisa se limitar somente a uma organização modular, dessa maneira podem ser adotados outros estilos arquiteturais mais adequados às necessidades de cada projeto de software, como: (i) a Arquitetura em Camadas que organiza as funcionalidades do sistema computacional em um número arbitrário de camadas (SOMMERVILLE, 2019), estabelecendo uma hierarquia em que os níveis mais internos se aproximam progressivamente do conjunto de instruções de máquina (PRESSMAN; MAXIM, 2016); (ii) a Arquitetura Cliente-Servidor que estrutura uma aplicação em um conjunto de servidores responsáveis pelo fornecimento das funcionalidades que serão consumidas pelos clientes desse sistema de software (SOMMERVILLE, 2019); e (iii) a Arquitetura de Chamadas e Retornos que pode ser classificada em duas categorias principais (PRESSMAN; MAXIM, 2016), onde a primeira define uma hierarquia de chamadas em que um programa principal chama um ou mais subprogramas e estes últimos podem chamar outros subprogramas, já a segunda estabelece a comunicação do programa principal e dos subprogramas através de chamadas de procedimentos remotos; entre outros estilos. Outro estilo arquitetural que vem ganhando popularidade é a Arquitetura Limpa (do inglês, *Clean Architecture*) (MARTIN, 2019), pois estabelece uma dependência de código fonte de fora para dentro, onde a camada externa contém os detalhes de implementação de baixo nível e as camadas internas estabelecem as regras mais abstratas de alto nível. Vale salientar que existem outros estilos arquiteturais com princípios semelhantes aos da Arquitetura Limpa, como a Arquitetura Onion (PALERMO, 2008) e a Arquitetura Hexagonal (COCKBURN, 2005).

Em relação aos **padrões arquiteturais**, na visão de Sommerville (2019) os termos estilos e padrões possuem o mesmo significado, pois ambos descrevem a organização de um sistema computacional. Por outro lado, Pressman e Maxim (2016) tratam os dois termos de maneira distinta. Na visão dos respectivos autores, um estilo arquitetural é uma organização imposta ao sistema inteiro e um padrão arquitetural é algo menos abrangente, pois estabelece uma regra para lidar com algum problema específico que possui um conjunto de limitações e restrições. A origem dos padrões está relacionada com a obra de Gamma *et al.* (2000), que retratam os padrões como uma maneira de apresentar, compartilhar e reutilizar conhecimentos.

De acordo com Fowler (2006), os padrões arquiteturais buscam sintetizar a essência das soluções que foram descobertas na prática. Tais soluções são baseadas nas observações das situações da vida real e muitas vezes não são ideias originais. É importante salientar que os padrões constituem soluções "semiprontas" que devem ser modificadas para atender as necessidades de um determinado projeto. Por esse motivo, um mesmo padrão arquitetural pode ser utilizado várias vezes sem que sua implementação seja exatamente a mesma. Além disso, os padrões arquiteturais podem ser utilizados para comunicar uma ideia e para estabelecer um vocabulário comum entre os desenvolvedores de um mesmo projeto. Como exemplos, são apresentados quatro padrões arquiteturais utilizados no projeto da biblioteca DynaSchema:

Padrão *Strategy*. Este padrão estabelece uma interface comum para encapsular os algoritmos de uma mesma família, permitindo que cada uma dessas estratégias (ou seja, cada algoritmo) possa ser utilizada de maneira intercambiável (GAMMA *et al.*, 2000). Como exemplo, o padrão *Strategy* foi utilizado na biblioteca DynaSchema para resolver o problema da existência de diferentes dialetos SQL (do inglês, *Structured Query Language*). Nesse contexto, a sintaxe de uma mesma instrução pode ser diferente dependendo do banco de dados utilizado, exigindo que a biblioteca DynaSchema esteja preparada para suportar novos dialetos SQL (ou seja, estratégias de sintaxe SQL) com o mínimo de alterações.

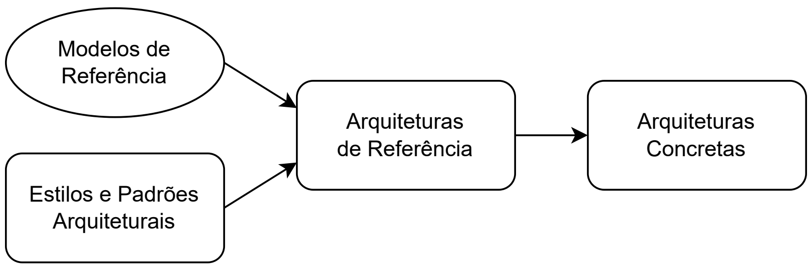
Padrão *Factory Method*. Este padrão permite a criação de objetos através de uma interface abstrata, delegando o processo de instanciação para suas extensões concretas (GAMMA *et al.*, 2000). Esse tipo de organização promove um desacoplamento entre o cliente e os objetos, pois a interface fica encarregada de encapsular dos detalhes de instanciação desses objetos. Como exemplo, uma versão mais simplifica do padrão *Factory Method* foi utilizado na biblioteca DynaSchema para estabelecer um mecanismo de criação dos objetos de dialeto SQL, cujo comportamento pode ser configurado dinamicamente através de um arquivo de metadados. Nesse sentido, o dialeto que será utilizado pela biblioteca deve ser especificado no arquivo de metadados, assim definindo a estratégia de sintaxe dos objetos de dialeto SQL que serão instanciados.

Padrão *Observer*. Este padrão viabiliza um meio de notificar e atualizar os dependentes de um objeto quando esse último muda de estado (GAMMA *et al.*, 2000). Para isso, é estabelecido um tipo de dependência "um para muitos", onde o objeto observado só conhece a interface dos seus objetos observadores. Nesse sentido, tal organização promove um acoplamento mínimo e abstrato entre o objeto de interesse e seus dependentes. Como exemplo, o padrão *Observer* foi utilizado para estabelecer o mecanismo de notificação entre os utilizadores da biblioteca DynaSchema e a emissão dos seus eventos internos.

Padrão *Facade*. Este padrão tem o objetivo de simplificar a utilização de um subsistema através da definição de uma interface com um nível mais alto de abstração (GAMMA *et al.*, 2000). Nesse sentido, o cliente pode acessar os recursos desse subsistema de maneira unificada por meio de uma única interface abstrata, assim facilitando e simplificando sua utilização. Como exemplo, a biblioteca DynaSchema adotou o padrão *Facade* para obter três benefícios principais, a saber: (i) estabelecer um único "ponto de acesso" aos recursos da biblioteca, simplificando sua utilização pelo cliente final; (ii) encapsular a complexidade de instanciação dos módulos da biblioteca, evitando que o cliente tenha que se preocupar com os detalhes de implementação; e (iii) encapsular a maneira como as funcionalidades da biblioteca são oferecidas, evitando que o cliente tenha que saber como coordenar as chamadas dos módulos da biblioteca DynaSchema.

Por fim, a **arquitetura de referência** pode ser definida como uma organização que pode ser reutilizada em diversos sistemas computacionais (ANGELOV; GREFEN; GREEFHORST, 2012; BASS; CLEMENTS; KAZMAN, 2003), sendo elaborada a partir de modelos de referência e estilos e padrões arquiteturais, conforme ilustrado na Figura 1. Nesse sentido, uma arquitetura concreta pode ser entendida como a instância de uma arquitetura de referência para um contexto específico. A natureza genérica da arquitetura de referência possibilita sua aplicação em diversas situações, constituindo um mecanismo de uniformização de arquiteturas concretas.

Figura 1 – Processo de obtenção de uma arquitetura concreta



Fonte: Adaptado de Angelov, Grefen e Greefhorst (2012).

Com o aumento da importância e complexidade dos sistemas computacionais, as arquiteturas de referência têm tido grande relevância no desenvolvimento de software. Esse tipo de arquitetura permite a definição e a adoção de boas práticas na construção dos sistemas, além de apoiar a evolução dos mesmos. Nesse sentido, as arquiteturas de referência têm sido projetadas para atender diferentes domínios de software, inclusive o software autoadaptativo. Como exemplo, a RA4SaS (AFFONSO; NAKAGAWA, 2013) e a RA4SelfMobApps (do inglês, *Reference Architecture for Self-adaptive Service-oriented Mobile Applications*) (AFFONSO; PASSINI; NAKAGAWA, 2019) são arquiteturas de referência que auxiliam o desenvolvimento de sistemas autoadaptativos em domínio geral e específico (ou seja, aplicações móveis orientadas a serviços), respectivamente. Além disso, é importante salientar que a biblioteca DynaSchema está inserida no contexto da arquitetura de referência RA4SaS.

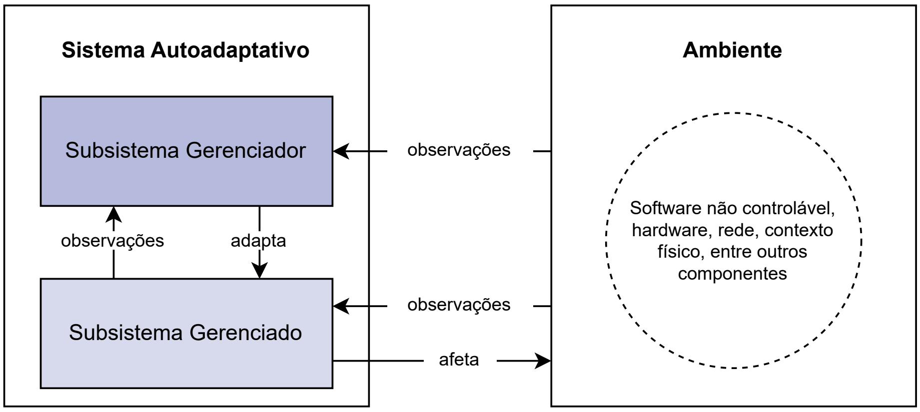
2.2 Sistemas Autoadaptativos

O cenário da evolução contínua exige que os sistemas de software se tornem cada vez mais versáteis, flexíveis, resilientes, confiáveis, eficientes em termos de energia, personalizáveis, auto-otimizáveis, entre outras características (CHENG *et al.*, 2009). Tais necessidades salientam a alta demanda pela redução da complexidade, automação do gerenciamento, robustez e atendimento de todos os requisitos desejados (SALEHIE; TAHVILDARI, 2009). Diante desse contexto, os sistemas autoadaptativos representam uma alternativa para lidar com o cenário da evolução contínua. Esse tipo de sistema é capaz de se modificar em tempo de execução, permitindo a alteração automática da sua dinâmica interna em resposta às mudanças no ambiente operacional e nas próprias entidades de software internas.

Um **sistema autoadaptativo** (ou seja, uma SaS) é capaz de se modificar automaticamente em resposta às mudanças no seu ambiente operacional (KRUPITZER *et al.*, 2015). A modificação da dinâmica interna do SaS pode ser feita através de alterações nos seus vários artefatos ou atributos (SALEHIE; TAHVILDARI, 2009), cujo nível de automação pode acontecer sem nenhuma ou com o mínimo de intervenção humana (BRUN *et al.*, 2009). Além disso, as mudanças no contexto de execução do SaS podem ocorrer devido a falhas, variabilidade nos recursos disponíveis, alteração das prioridades do usuário, entre outros fatores (WEYNS *et al.*, 2013). Para lidar com essas adversidades, esse sistema possui certas características, conhecidas como propriedades *self-**, que fornecem algum grau de variabilidade para a aplicação e ajudam a superar os desvios esperados durante seu funcionamento (SALEHIE; TAHVILDARI, 2009).

Na Figura 2 é ilustrado o contexto de funcionamento de um SaS, sendo composto por dois elementos principais, a saber: (i) *o ambiente*; e (ii) *o próprio sistema autoadaptativo* (WEYNS *et al.*, 2013). O ambiente constitui a parte do mundo externo com o qual um SaS interage e cujos efeitos dessa interação serão observados e avaliados. Esse contexto pode corresponder tanto as entidades físicas do ambiente quanto as entidades internas do próprio SaS. Por outro lado, o sistema autoadaptativo pode ser dividido em duas partes, a saber: (i) *um subsistema gerenciado*, que é a parte da aplicação que contém a lógica de negócio e as funcionalidades de seu domínio; e (ii) *um subsistema gerenciador*, que é a parte da aplicação que contém a lógica de adaptação, além de ser responsável pela administração do subsistema gerenciado. É importante salientar que a separação entre o subsistema gerenciador e o gerenciado nem sempre é clara. Dessa maneira, a implementação de um sistema autoadaptativo pode condensar as lógicas de adaptação e de negócio em uma única aplicação executável de software.

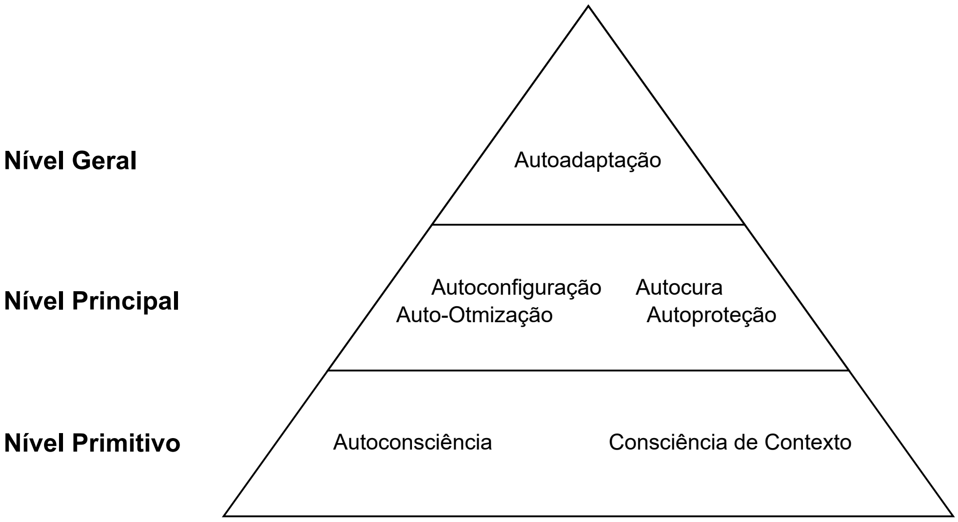
Figura 2 – Ambiente operacional de um SaS



Fonte: Adaptado de Weyns *et al.* (2013).

Além dos aspectos organizacionais, um SaS possui algumas características, conhecidas como **propriedades *self-****, que auxiliam a aplicação na tratativa dos desvios causados pelas mudanças constantes no seu ambiente operacional (SALEHIE; TAHVILDARI, 2009). Na Figura 3 é apresentada uma organização hierárquica dessas propriedades em torno de três níveis, a saber: (i) *geral*; (ii) *principal*; e (iii) *primitivo*.

Figura 3 – Organização hierárquica das propriedades *self-**



Fonte: Adaptado de Salehie e Tahvildari (2009).

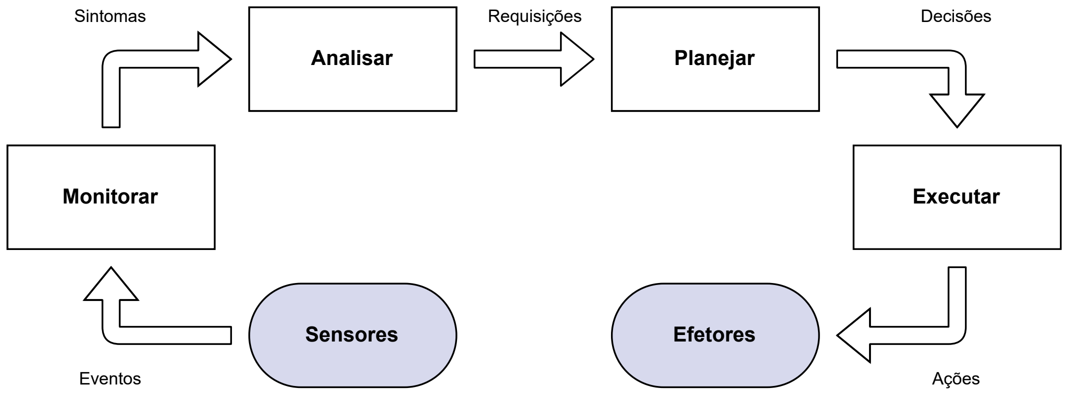
No **nível geral** estão contidas as propriedades globais de um SaS, como autogerenciamento, automanutenção, autocontrole, entre outras características. No **nível principal** estão contidas as quatro propriedades padrões definidas pela IBM (do inglês, *International Business Machines*) de acordo com os mecanismos de autoadaptação biológica, a saber: (i) autoconfiguração, que é a capacidade de reconfiguração dinâmica com base na instalação, atualização, integração e (de)composição das entidades de software; (ii) autocura, que é a capacidade de descobrimento, diagnóstico e reação aos distúrbios do sistema e seu ambiente; (iii) auto-otimização, que é a capacidade de gerenciamento do desempenho e a alocação de recursos para o atendimento dos requisitos do sistema; e (iv) autoproteção, que é a capacidade de detecção das violações de segurança e a recuperação dos seus efeitos. Finalmente, no **nível primitivo** estão contidas as propriedades de caráter básico de um SaS, como automonitoramento e autosituação. Além dessas características, outras duas propriedades são essenciais para alcançar o comportamento autoadaptativo, a saber (KRUPITZER *et al.*, 2015): (i) autoconsciência, que é a capacidade de monitoração dos recursos, estado e comportamento do sistema; e (ii) consciência do contexto, que é a capacidade de avaliação do ambiente em que o sistema está inserido.

Os **laços de adaptação** possuem um papel central na maneira como os sistemas autoadaptativos são conceitualizados e projetados (WEYNS *et al.*, 2013). Nesse sentido, o laço de controle fornece um mecanismo genérico para a autoadaptação (BRUN *et al.*, 2009) e provê uma visão mais ampla do que acontece dentro do sistema e seu ambiente (SALEHIE; TAHVILDARI, 2009). Na Figura 4 é ilustrada a organização de um laço de controle MAPE (do inglês, *Monitor, Analyze, Plan, Execute*), onde os retângulos com os cantos arredondados representam as interfaces com o sistema e o ambiente, as setas descrevem o fluxo de dados e os retângulos com os cantos retos simbolizam os processos do laço de controle. Os sensores e efetores (também conhecidos como atuadores) são componentes essenciais para um SaS e a instrumentação dessas interfaces constitui o primeiro passo para a sua construção (SALEHIE; TAHVILDARI, 2009). Os sensores são responsáveis pelo monitoramento das entidades de software, cujos dados coletados refletem o estado do sistema. Por outro lado, os efetores estabelecem o mecanismo para aplicar as mudanças no sistema em tempo de execução. Uma descrição de cada um dos quatro processos do laço de controle MAPE é apresentada a seguir:

1. **Monitorar:** este processo é responsável por coletar os dados dos sensores e correlacionar as informações obtidas, permitindo a identificação dos padrões de comportamento e sintomas do sistema (SALEHIE; TAHVILDARI, 2009). Tal procedimento se preocupa com questões sobre a taxa de amostragem, a confiabilidade dos dados coletados, o formato dos eventos emitidos pelos sensores, entre outras preocupações (CHENG *et al.*, 2009);

2. **Analisar:** este processo é responsável pela análise dos sintomas fornecidos pela atividade

Figura 4 – Laço de controle MAPE



Fonte: Adaptado de Salehie e Tahvildari (2009).

de monitoramento e pelo histórico do sistema, permitindo a identificação de uma transição de estado do sistema em resposta às mudanças percebidas (SALEHIE; TAHVILDARI, 2009). Tal procedimento se preocupa com questões sobre a maneira de inferência do estado atual do sistema, a possibilidade de reutilização de estados anteriores, a validação e verificação dos dados contidos nos sintomas, a fidelidade do modelo estabelecido com o mundo real, entre outras preocupações (CHENG *et al.*, 2009);

3. **Planejar:** este processo é responsável pela definição do plano de mudança, permitindo a identificação do que precisa ser mudado e como as alterações devem ser feitas para alcançar o melhor resultado (SALEHIE; TAHVILDARI, 2009). Esse planejamento deve ser baseado em critérios para possibilitar a comparação entre as diferentes maneiras de se aplicar as mudanças e permitir a escolha da solução mais adequada ao contexto percebido no processo de análise. Tal procedimento se preocupa com as questões sobre a maneira de inferência do estado futuro do sistema, como uma decisão é tomada, quais são as prioridades de adaptação, entre outras preocupações (CHENG *et al.*, 2009); e

4. **Executar:** este processo é responsável pela aplicação das ações definidas na atividade de planejamento, permitindo a adaptação do sistema (SALEHIE; TAHVILDARI, 2009). Essas ações são executadas através dos efetores disponíveis e podem ser coordenadas por meio de fluxos de trabalho predefinidos ou mapeamento de atividades. Tal procedimento se preocupa com as questões sobre quando a adaptação é segura, se a alteração planejada ajuda o sistema a atingir uma meta global, se o plano de ação pode ser executado com os efetores disponíveis, entre outras preocupações (CHENG *et al.*, 2009).

2.3 Reference Architecture for Self-adaptive Software

A arquitetura de referência **RA4SaS** tem o objetivo de apoiar o desenvolvimento de sistemas autoadaptativos de propósito geral (AFFONSO; NAKAGAWA, 2013). Essa arquitetura de referência se baseia em três conceitos principais, a saber: (i) o laço de adaptação MAPE-K (do inglês, *Monitor, Analyze, Plan, Execute over Knowledge*) (IBM, 2005), que permite a administração das entidades de software no ambiente de execução; (ii) a reflexão computacional (MAES, 1987), que fornece os meios para modificar as entidades de software em tempo de execução; e (iii) a abordagem de adaptação externa (SALEHIE; TAHVILDARI, 2009), que organiza a lógica de adaptação em duas camadas, a saber: (1) *o supervisor*, que contém a lógica da adaptação; e (2) *o supervisionado*, que possui as entidades de software autoadaptativas. Além disso, é importante salientar que essa arquitetura de referência trabalha com uma modalidade de adaptação controlada, onde o desenvolvedor deve especificar o nível da adaptação de cada entidade de software. Esses níveis são utilizados para definir quais alterações uma entidade de software pode implementar em tempo de execução.

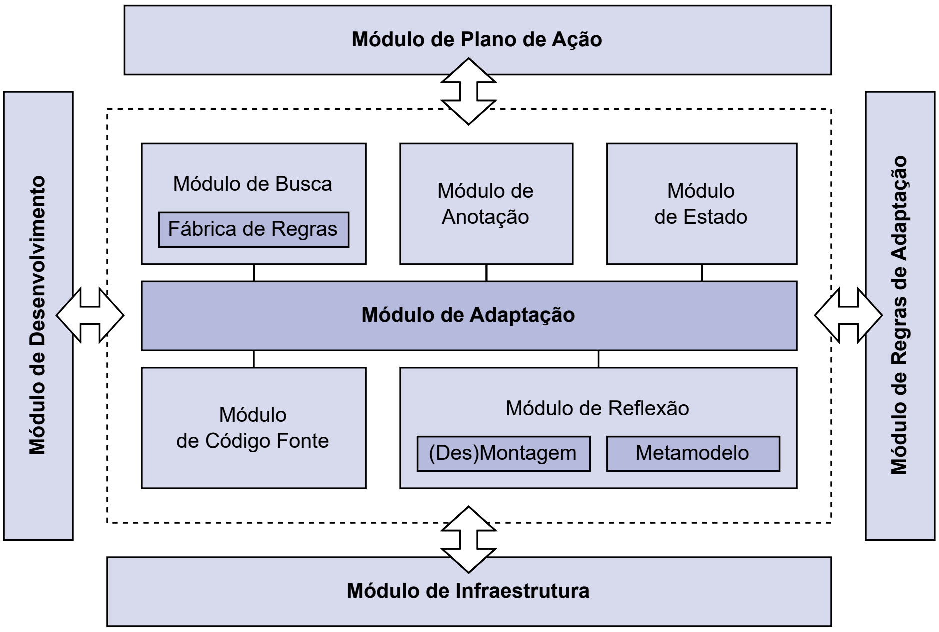
Conforme ilustrado na Figura 5, a arquitetura de referência RA4SaS é organizada em cinco módulos principais, a saber: (i) o módulo de Desenvolvimento fornece as diretrizes para a criação das entidades de software do SaS; (ii) o módulo de Plano de Ação auxilia o processo de adaptação do sistema; (iii) o módulo de Regras de Adaptação é responsável por analisar as entidades de software e extrair suas regras de adaptação; (iv) o módulo de Infraestrutura fornece os mecanismos para adaptar as entidades de software; e (v) o módulo de Adaptação constitui o "coração" da arquitetura de referência RA4SaS, sendo responsável pelo gerenciamento do processo de adaptação do SaS que é apoiado por outros módulos auxiliares.

O **módulo de Adaptação** concentra grande parte da complexidade da arquitetura de referência RA4SaS, pois representa o núcleo de adaptação do SaS. De acordo com a estrutura ilustrada na Figura 5, esse módulo é composto por outros cinco módulos internos, a saber:

Módulo de Pesquisa. Este módulo auxilia o processo de adaptação, sendo responsável pela busca das entidades de software no ambiente de execução. Dessa maneira, o método de pesquisa pode utilizar como parâmetros os conceitos e as notações que definem um metamodelo, além das informações técnicas contidas nos *templates* de geração de código.

Módulo de Anotação. Este módulo apoia os desenvolvedores na definição do nível de adap-

Figura 5 – Módulos da arquitetura de referência RA4SaS



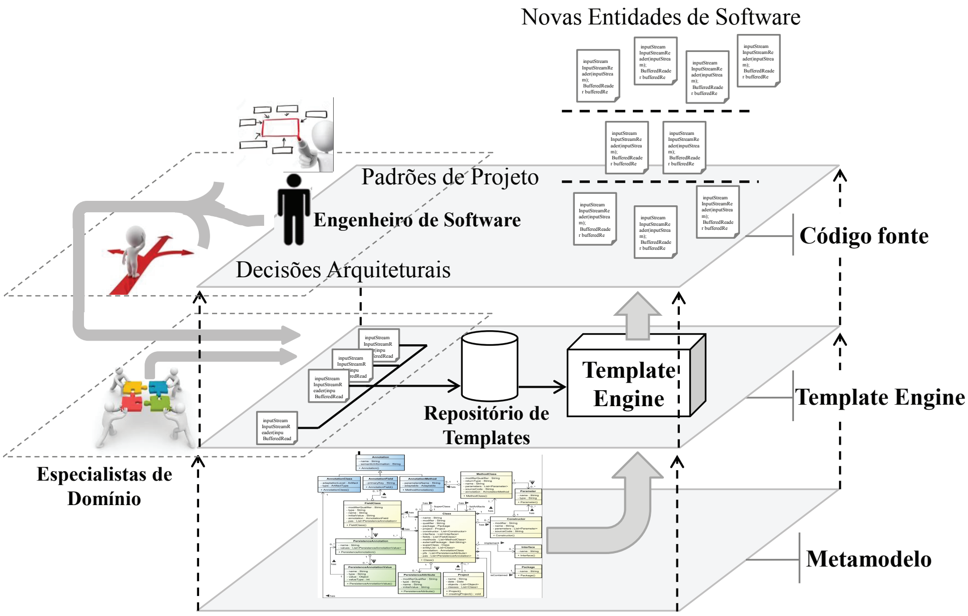
Fonte: Adaptado de Affonso e Nakagawa (2013).

tação das entidades de software ao longo da fase de desenvolvimento. Nesse sentido, anotações para classes, atributos e métodos são utilizadas para marcar os níveis de adaptação dessas estruturas. Posteriormente, o módulo de Reflexão utiliza as informações contidas nessas anotações para identificar o que pode ser modificado (ou seja, removido e/ou adicionado) nas entidades software do SaS.

Módulo de Código Fonte. Este módulo tem a função de gerar o código fonte das entidades de software adaptadas, baseando-se nas informações contidas no metamodelo do módulo de Reflexão e nos *templates* de geração de código. Assim, os engenheiros de software e especialistas de domínio podem concentrar seus esforços na criação dos *templates* de geração de código, estabelecendo os padrões arquiteturais aplicados no sistema e o nível de adaptação das entidades de software. Dessa maneira, o processo de geração das novas entidades, ilustrado na Figura 6, se baseia na lógica estabelecida nos *templates* e nas mudanças contidas no metamodelo (BENATO; AFFONSO; NAKAGAWA, 2017).

Módulo de Estado. Este módulo é responsável pelo armazenamento do estado atual das entidades de software. As informações de estado são armazenadas em um arquivo XML (do inglês, *eXtensible Markup Language*) (ou seja, serializadas), podendo ser recuperadas e reinseridas (ou seja, desserializadas) após o processo de adaptação das entidades.

Figura 6 – Processo de geração do código fonte das entidades de software



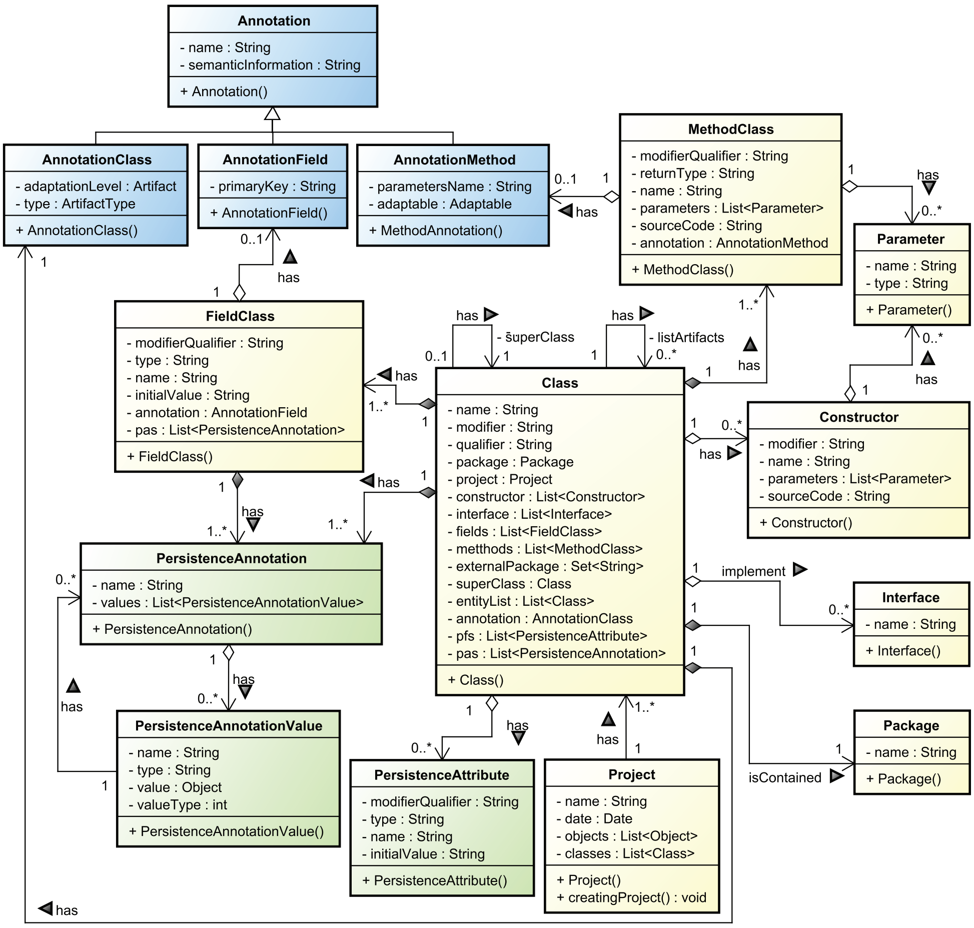
Fonte: (BENATO; AFFONSO; NAKAGAWA, 2017).

Módulo de Reflexão. Este módulo auxilia o processo de “montagem” e “desmontagem” das entidades de software. O processo de “desmontagem” gera um metamodelo com as informações extraídas pelo módulo de Reflexão, tendo como base as anotações contidas nas entidades. Nesse metamodelo, ilustrado na Figura 7, as informações podem ser adicionadas ou removidas para então serem enviadas novamente para o módulo de Código Fonte, permitindo a criação das novas entidades de software ou alteração das existentes.

A ferramenta de modelagem **DSLModeler4SaS** (do inglês, *Domain-Specific Language Modeler for Self-adaptive Software*) foi criada para apoiar o desenvolvimento de SaSs baseados na RA4SaS. Tal ferramenta fornece um ambiente completo para a criação e gerenciamento tanto das entidades de software quanto dos *templates* para a geração de código fonte. Dentro da ferramenta, esses artefatos podem ser organizados em projetos individuais, permitindo que cada projeto possa ser configurado de acordo com as necessidades de um trabalho em particular, além de armazenar os modos de aplicação dos *templates* para a geração de código fonte do SaS.

Na Figura 8 é ilustrada a interface gráfica da DSLModeler4SaS. No lado esquerdo da interface está localizada a árvore de projetos, onde são listadas as entidades de software **(1)** e os *templates* de código fonte **(2)** de cada projeto. Na parte central da interface se encontra o editor de código para o desenvolvimento das entidades e dos *templates* **(3)**. Vale salientar que esse editor é organizado em abas, permitindo que o desenvolvedor acesse cada entidade ou *template*

Figura 7 – Metamodelo de uma entidade de software

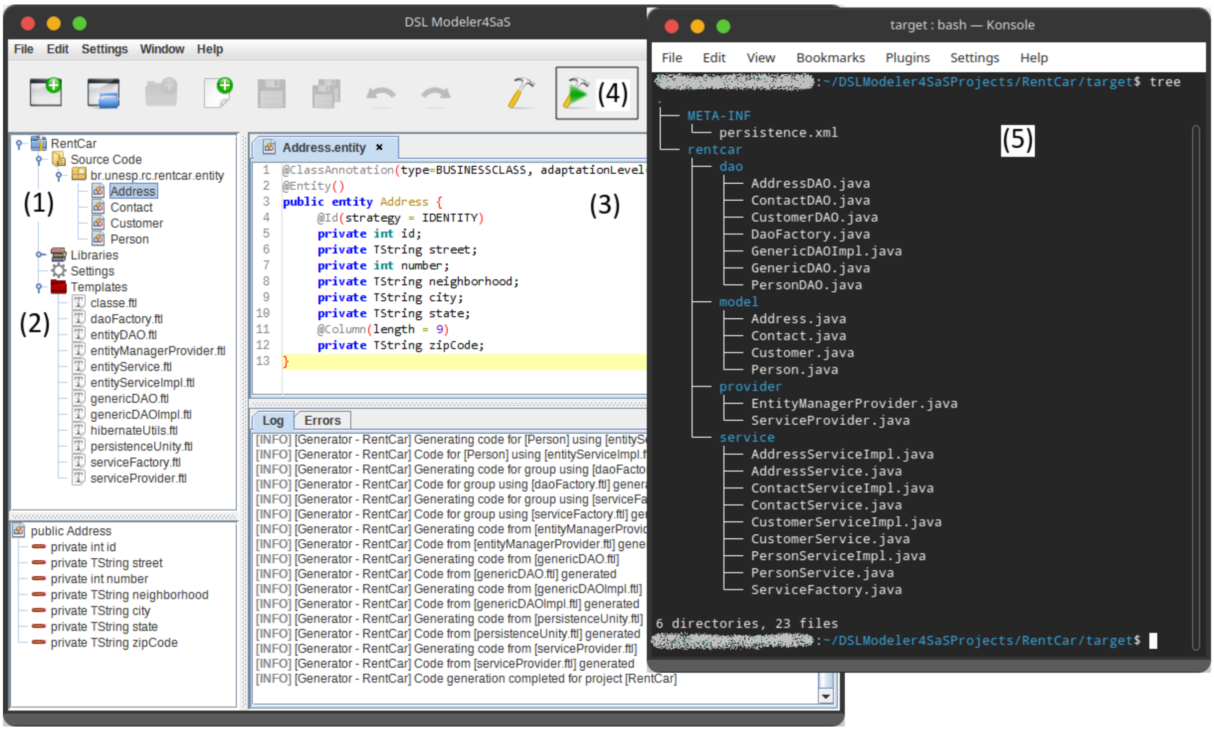


Fonte: (BENATO; AFFONSO; NAKAGAWA, 2017).

do projeto de maneira individual. Para gerar o código fonte do SaS, o desenvolvedor de software deve pressionar o botão com uma imagem de martelo e um triângulo verde **(4)**, localizado na parte superior da interface da DSLModeler4SaS. Dessa maneira, as entidades do projeto serão convertidas em metamodelos (Figura 7), sendo enviados para o módulo de Código Fonte da RA4SaS em conjunto com os *templates* de código fonte. Em seguida, o código do SaS será gerado **(5)** de acordo com o processo que foi ilustrado na Figura 6.

Por fim, vale destacar que a ferramenta de modelagem DSLModeler4SaS possui um papel importante no contexto da RA4SaS, pois permite a automação de grande parte do processo de criação e manutenção de um SaS baseado nessa arquitetura de referência, reduzindo o esforço empregado pelo desenvolvedor de software. Nesse sentido, é importante lembrar que a biblioteca DynaSchema, desenvolvida neste projeto de mestrado, considera sua utilização no contexto da modelagem de entidades de software no ambiente da ferramenta DSLModeler4SaS.

Figura 8 – Ferramenta de modelagem DSLModeler4SaS



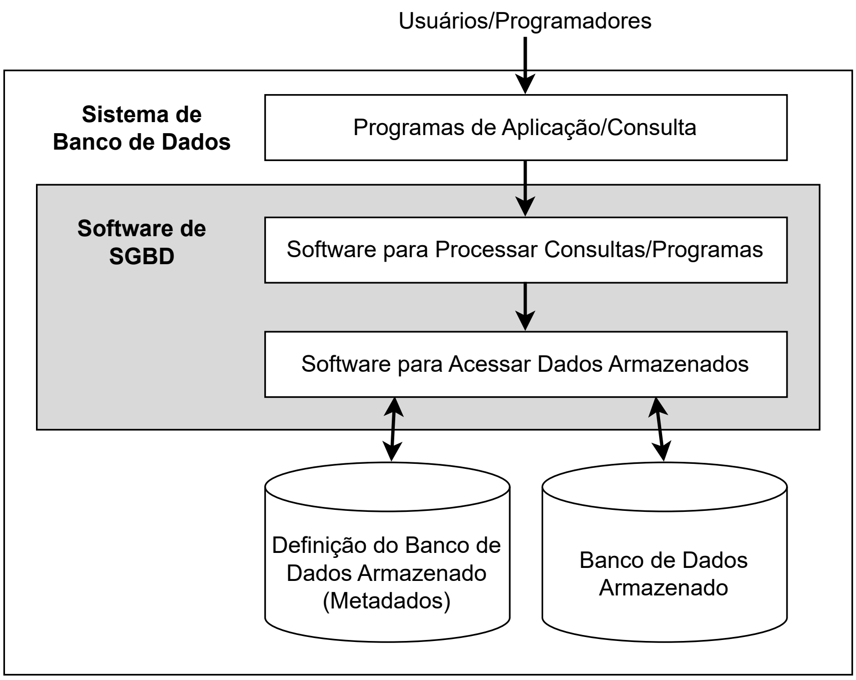
Fonte: Elaborada pelo autor.

2.4 Sistemas de Banco de Dados

Antes da invenção dos sistemas de banco de dados, uma aplicação típica para o processamento de dados deveria lidar com a manipulação de arquivos no formato de armazenamento imposto pelo sistema operacional (SILBERSCHATZ; KORTH; SUDARSHAN, 2021). Essa abordagem de organização pode apresentar vários problemas, tais como: (i) redundância e inconsistência, que podem surgir quando partes de uma aplicação armazenam seus dados de forma independente, possivelmente com informações duplicadas (ou seja, réplicas iguais) ou inconsistentes (ou seja, cópias com informações diferentes); (ii) atomicidade, onde a aplicação deve garantir que uma sequência de operações sobre os dados deve ser executada de um só vez ou desfeita em sua totalidade na ocorrência de algum erro; e (iii) acesso concorrente, onde os dados podem ser acessados e modificados simultaneamente por diversos usuários, podendo afetar a consistência das informações; entre outros desafios. Tais problemas podem ser solucionados com a utilização de um sistema de banco de dados, pois essa abordagem fornece uma visão abstrata dos dados através da ocultação dos detalhes do seu gerenciamento e armazenamento (SILBERSCHATZ; KORTH; SUDARSHAN, 2021). Nesse sentido, um sistema de banco de dados é uma solução de software para armazenar informações e fornecer uma interface conveniente para os usuários atualizarem e consultarem essas informações (DATE, 2004).

A organização interna de um **sistema de banco de dados** pode ser visualizada de maneira simplificada na Figura 9. De acordo com essa visão, o sistema de banco de dados pode ser definido pela união de dois componentes principais (ELMASRI; NAVATHE, 2019). O primeiro é o banco de dados, que viabiliza a integração das informações em um único repositório compartilhado, permitindo a criação de visões personalizadas sobre os dados para atender as necessidades de cada usuário do sistema (DATE, 2004). O segundo é o SGBD (*Sistema Gerenciador de Banco de Dados*), que estabelece uma camada de software entre o sistema de banco de dados e seus usuários. Esse software fornece várias funcionalidades de gerenciamento, como as operações inserção, atualização, remoção e consulta dos dados, por exemplo. Portanto, o SGBD fornece aos usuários do sistema um meio eficaz de interagir com o banco de dados, escondendo seus detalhes de implementação no nível de hardware. É importante ressaltar que o termo SGBD também é frequentemente utilizado como um sinônimo de um produto de banco de dados, como o MySQL da Oracle e SQLServer da Microsoft, por exemplo. Além disso, de acordo com a arquitetura ilustrada na Figura 9, é possível notar que as informações sobre a estrutura do banco de dados podem ser armazenadas na forma de metadados.

Figura 9 – Arquitetura simplificada de um sistema de banco de dados



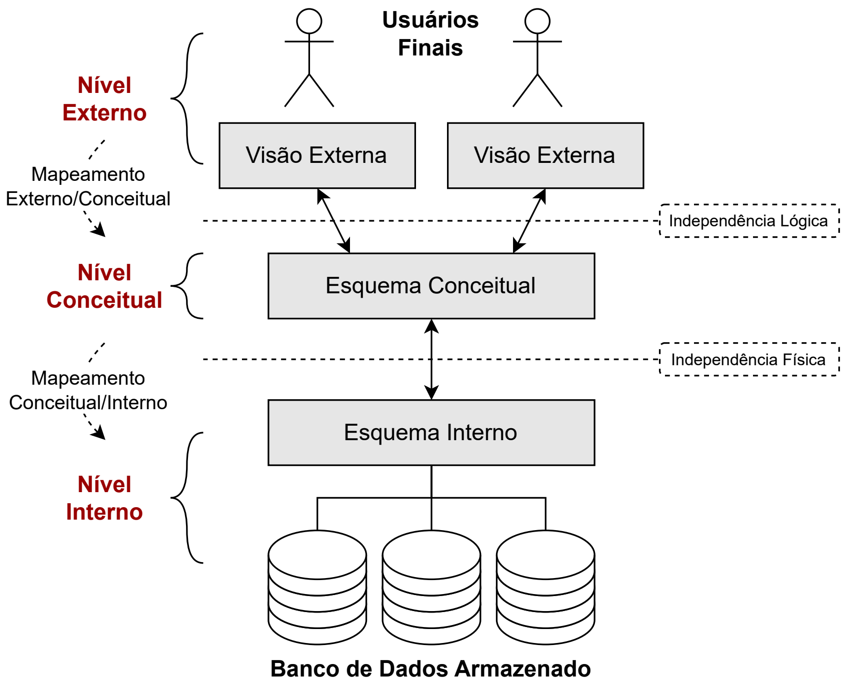
Fonte: Adaptado de Elmasri e Navathe (2019).

De acordo com Elmasri e Navathe (2019), um **banco de dados** pode ser definido como uma coleção de dados relacionados, sendo equivalente a um antigo armário de arquivamento (DATE, 2004). Seu tamanho e grau de complexidade podem variar desde um catálogo atualizado de maneira manual até um sistema computadorizado mantido por uma empresa (ELMASRI; NAVATHE, 2019). A conceitualização de um banco de dados também envolve a utilização de um modelo de dados para descrever a estrutura, os relacionamentos, a semântica e as restrições dos dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2021).

O crescimento da *World Wide Web* na década de 1990 exigiu a utilização de sistemas de banco de dados que pudessem suportar o processamento de um grande volume de transações com alta confiabilidade e disponibilidade de funcionamento de 24/7 (ou seja, 24 horas por dia e 7 dias por semana) (SILBERSCHATZ; KORTH; SUDARSHAN, 2021). Tais necessidades foram supridas pelos SGBDs relacionais e desde então são tradicionalmente adotados no desenvolvimento de qualquer tipo de sistema de informação. Entretanto, os SGBDs não relacionais (NoSQL) vem ganhando popularidade desde a década de 2010 devido aos benefícios de escalabilidade e disponibilidade trazidos por esse tipo de sistema de banco de dados. Vale salientar que tanto os bancos de dados relacionais quanto os não relacionais têm sido utilizados para outras finalidades, além de armazenar informações de propósito geral. Diante desse cenário, os *data warehouses* são tipos especiais de bancos de dados que são adotados na inteligência de negócios das empresas para apoiar a tomada de decisões (DATE, 2004). Tais bancos de dados são populados com grandes quantidades de informação sobre diversas áreas do negócio que variam com o tempo e não podem ser alterados ou excluídos. Esses dados estão sujeitos a intensas cargas de trabalho para produzir os relatórios de análise comercial. Nesse contexto está inserido o processamento OLAP (do inglês, *On-Line Analytical Processing*) que é responsável pela criação, gerenciamento, análise e geração desses relatórios de inteligência.

Na visão de Elmasri e Navathe (2019), um **SGBD** pode ser definido com um programa de computador para criar e manter um banco de dados. Nesse sentido, esse programa tem a finalidade de fornecer um ambiente adequado e eficiente para os usuários manipularem esse banco (SILBERSCHATZ; KORTH; SUDARSHAN, 2021). Além disso, o SGBD é responsável por controlar todo o acesso ao banco de dados (DATE, 2004), fornecendo funcionalidades de proteção contra falhas de hardware ou software, manutenção do sistema e suporte evolutivo para a mudança dos requisitos ao longo do tempo (ELMASRI; NAVATHE, 2019). Antes da invenção dos sistemas de banco de dados, as aplicações antigas geralmente implementavam no seu código fonte a maneira como seus dados eram estruturados e acessados fisicamente (DATE, 2004). Essa estratégia é pouco flexível, pois a aplicação deve ser modificada sempre que a estrutura física dos dados precisa ser alterada. A utilização de uma abordagem de SGBD evita esse problema, pois os efeitos colaterais causados pela alteração na estrutura ou no método de acesso aos dados são isolados da aplicação. Essa característica é chamada de independência de dados e os SGBDs podem alcançar esse comportamento através de uma arquitetura de três esquemas, conforme ilustrado na Figura 10. Além disso, os SGBDs devem apresentar uma interface para que os usuários possam criar e alterar o esquema do banco de dados de acordo com suas necessidades (ELMASRI; NAVATHE, 2019). Nesse sentido, uma maneira de executar essas ações consiste na utilização das sublinguagens de dados para manipular os três esquemas (DATE, 2004). Uma breve descrição de cada nível da arquitetura de três esquemas e de sua respectiva sublinguagem de manipulação é apresentada a seguir:

Figura 10 – Arquitetura de três esquemas para um SGBD



Fonte: Adaptado de Elmasri e Navathe (2019).

Esquema Interno. Este esquema descreve a estrutura do banco de dados em um nível de abstração mais baixo (DATE, 2004), especificando os detalhes físicos para o armazenamento dos dados e seus caminhos de indexação (ELMASRI; NAVATHE, 2019). A SDL (do inglês, *Storage Definition Language*) pode ser utilizada para definir os detalhes do esquema interno, permitindo que os administradores e projetistas de banco de dados especifiquem os métodos de indexação e a maneira de mapear os dados armazenados no banco.

Esquema Conceitual. Este esquema descreve a estrutura de todos os dados contidos no banco de dados (DATE, 2004), se concentrando na maneira como as entidades, tipos de dados, relacionamentos, operações e restrições são representadas (ou seja, os metadados) (ELMASRI; NAVATHE, 2019). Além disso, esse esquema também pode detalhar as restrições de segurança e integridade dos dados (DATE, 2004). A DDL (do inglês, *Data Definition Language*) pode ser utilizada para definir a estrutura do esquema conceitual e controlar seus mapeamentos com o esquema interno (ELMASRI; NAVATHE, 2019).

Esquema Externo. Este esquema descreve a estrutura das visões sobre o conteúdo do banco de dados (ELMASRI; NAVATHE, 2019). Em geral, os usuários do banco só estão interessados em partes dos dados, fazendo com que esse esquema seja útil para estabelecer as visões personalizadas para cada usuário (DATE, 2004). A VDL (do inglês, *View Definition Language*) pode ser utilizada para especificar a estrutura do esquema externo, permitindo que os administradores e projetistas de banco de dados definiam as visões dos usuários e controlem os mapeamentos com o esquema conceitual (ELMASRI; NAVATHE, 2019).

Por fim, vale destacar que os SGBDs fornecem algum suporte para descrever o esquema do banco de dados de acordo com a arquitetura de três esquemas que foi ilustrada na Figura 10, mas a separação entre os três níveis geralmente não é explicita, além de não haver na prática nenhuma distinção clara entre as sublinguagens de cada nível (ELMASRI; NAVATHE, 2019). Como exemplo, a linguagem SQL utilizada pela maioria dos SGBDs relacionais condensa todas as funcionalidades das sublinguagens DDL e VDL. Além disso, grande parte dos SGBDs também utiliza a DDL para definir a estrutura dos esquemas interno, conceitual e externo, ou seja, a DDL geralmente incorpora as funcionalidades da SDL e VDL.

Fazendo um paralelo entre os conceitos apresentados nesta seção e este projeto de mestrado, um dos desafios enfrentados pela biblioteca DynaSchema é a manutenção da integridade das informações do banco de dados de acordo com a evolução promovida pelos ciclos de autoadaptação do SaS. Nesse sentido, o processo de migração dos registros contidos na versão antiga do esquema de dados do SaS para a nova versão deve ser consistente, evitando a perda de informações ou a cópia de dados espúrios. Além disso, podem existir casos que exigem o mapeamento de dados entre as versões de esquema de dados antiga e nova, como no caso da alteração do tipo de dado do campo de alguma tabela do banco de dados, por exemplo.

O gerenciamento do esquema de banco de dados é também outro problema enfrentado pela biblioteca DynaSchema. Nesse sentido, essa biblioteca deve ser capaz de expressar para o SGBD quais as mudanças devem ser feitas no esquema para acompanhar os ciclos de autoadaptação do SaS. Nesse sentido, a biblioteca DynaSchema deve saber como criar, alterar e excluir as tabelas e colunas presentes nesse esquema de banco de dados. A execução dessas operações não é uma tarefa trivial, pois os SGBDs não apresentam uma sintaxe padronizada da linguagem SQL. Isso exige que a biblioteca tenha que conhecer qual o SGBD está sendo utilizado pelo SaS para poder executar as instruções SQL de maneira correta.

2.5 Considerações Finais

Este capítulo foi planejado para apresentar os principais conceitos e fundamentos que estão envolvidos na elaboração da biblioteca DynaSchema desenvolvida neste projeto de mestrado. Inicialmente, na Seção 2.1 foi apresentado o tema da arquitetura de software que fornece a base de conhecimento para o desenvolvimento de qualquer sistema de software. Em seguida, nas Seções 2.2 e 2.3 foram abordados os conceitos sobre SaS e uma visão geral da RA4SaS, respectivamente, apresentando os principais assuntos relacionados ao tema de pesquisa deste projeto de mestrado. Finalmente, na Seção 2.4 foram reportados os principais conceitos envolvidos em uma abordagem de sistema de banco de dados. Esse assunto combinado com a área de SaS representa a temática de interesse deste projeto de mestrado acadêmico.

3 Evolução do Esquema de Dados para o Domínio de Sistemas Autoadaptativos

Neste capítulo é apresentada uma visão geral sobre a evolução do esquema de dados para o domínio de SaS. Esse tipo de software tem se mostrado relevante no cenário atual de evolução e aprimoramento contínuo dos sistemas de software (CHENG *et al.*, 2009), pois sua capacidade de autoadaptação em tempo de execução permite lidar com as mudanças que ocorrem dentro do sistema e em seu ambiente operacional (SALEHIE; TAHVILDARI, 2009). Muitos trabalhos científicos da comunidade de SaS se concentram na investigação dos aspectos construtivos desse tipo de software, além de explorarem questões de engenharia e boas práticas. Por outro lado, a persistência de dados para o domínio de SaS e os problemas relacionados à evolução do esquema de dados são assuntos pouco explorados na literatura científica e merecem uma investigação mais aprofundada. Com relação a organização deste capítulo, seu conteúdo foi estruturado em três seções. Inicialmente, na Seção 3.1 é apresentado o mapeamento sistemático da literatura sobre a evolução do esquema de dados para o domínio de SaS. Em seguida, na Seção 3.2 são discutidos os resultados desse mapeamento com base na análise de 17 estudos primários selecionados. Por fim, na Seção 3.3 é realizado um fechamento sobre a revisão apresentada neste capítulo.

3.1 Mapeamento Sistemático da Literatura

As áreas de pesquisa em SaS e evolução do esquema de dados são extensas e complexas, sendo normalmente abordadas na literatura científica de maneira separada e sob uma variedade de aspectos. O tema abordado neste projeto de mestrado acadêmico pode ser considerado como uma intersecção entre esses domínios, fazendo-se necessária uma pesquisa para descobrir como a comunidade científica e a indústria de software têm lidado com essas áreas de interesse em conjunto. Para obter um ponto de vista abrangente, foi adotado o método de mapeamento sistemático da literatura (ou seja, SMS), pois essa técnica permite a construção de um conhecimento estruturado a partir da síntese, avaliação e interpretação das evidências relacionadas a um tema de interesse (KITCHENHAM; DYBA; JORGENSEN, 2004; KITCHENHAM; CHARTERS, 2007; PETERSEN; VAKKALANKA; KUZNIARZ, 2015).

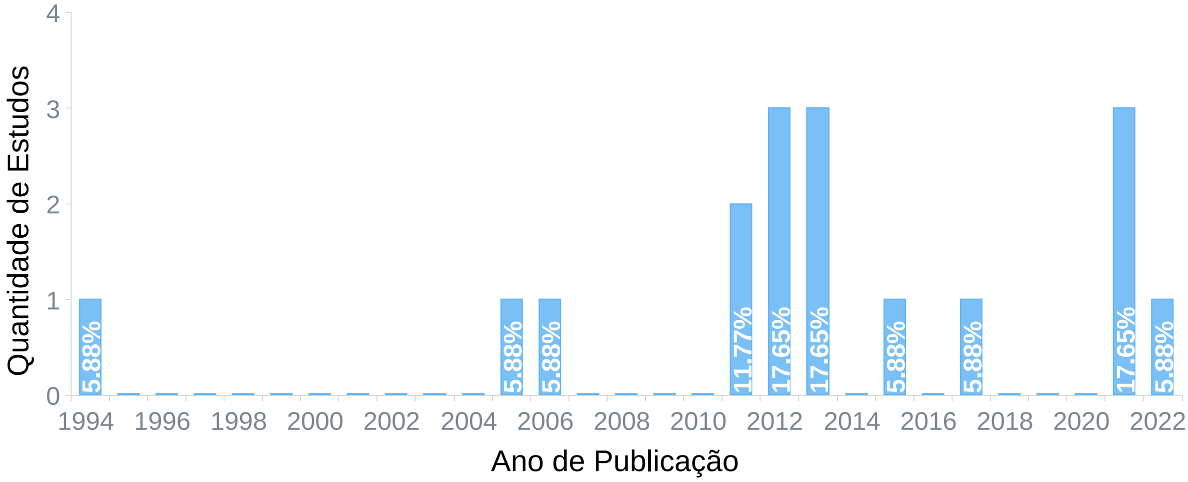
Inicialmente, como maneira de obter um panorama sobre as publicações que abordam a evolução do esquema de dados para o domínio de SaS, foi conduzido um levantamento de estudos secundários que seguiu as diretrizes propostas por Petersen, Vakkalanka e Kuzniarz (2015). Tal levantamento teve como objetivo a identificação de outros trabalhos de SMS semelhantes, além de descobrir como esse tema tem sido abordado na literatura científica. Como resultado, não foi encontrado nenhum estudo secundário sobre essa temática. A maioria dos trabalhos tratou o domínio de SaS de maneira isolada, sem considerar a evolução do seu esquema de dados. Esse cenário está em conformidade com a revisão elaborada por Rahm e Bernstein (2006), cujo resultado indicou uma maior concentração de estudos que envolvem os domínios de evolução do esquema de banco de dados e sistemas orientados a objetos.

Em seguida, devido à originalidade do tema de pesquisa deste projeto de mestrado, foi conduzido outro levantamento considerando somente os estudos primários que abordaram esse assunto de interesse, como publicações em periódicos, artigos de conferências e estudos de caso, por exemplo. Tal levantamento teve como objetivo a identificação de como esses estudos individuais abordam a evolução do esquema de dados para o domínio de SaS e quais soluções foram propostas para lidar com esse problema. O protocolo de pesquisa adotado na seleção dos estudos primários relevantes pode ser consultado no Apêndice B. Como resultado desse mapeamento, 17 estudos primários foram selecionados para compor o SMS sobre a evolução do esquema de dados para o domínio de SaS, cujas referências podem ser consultadas na Tabela 1 do Apêndice A. A seguir, nas Seções 3.1.1 a 3.1.7 são apresentadas as respostas para cada QP (Questão de Pesquisa) elaborada para o mapeamento conduzido neste projeto de mestrado.

3.1.1 **QP1:** Quais são os aspectos demográficos dos estudos?

Esta QP teve como objetivo a identificação de um panorama geral sobre a procedência dos estudos primários levantados por este SMS. O primeiro dado demográfico pode ser visto na Figura 11, que ilustra a distribuição da quantidade de estudos por ano de publicação (**QP1.1**). É possível observar que os estudos selecionados cobrem um período de publicação entre os anos de 1994 e 2022, totalizando um intervalo de 29 anos de pesquisa, considerando os anos do intervalo. Além disso, pode-se notar que tais estudos possuem uma tendência de publicação que oscila entre períodos de maior e menor produção. De acordo com o gráfico ilustrado na Figura 11, foram observados três intervalos de maior concentração de estudos, a saber: (i) *2011 a 2013*, com oito estudos (47,06%); (ii) *2021 e 2022*, com quatro estudos (25,53%); e (iii) *2005*

Figura 11 – Distribuição dos estudos por ano de publicação



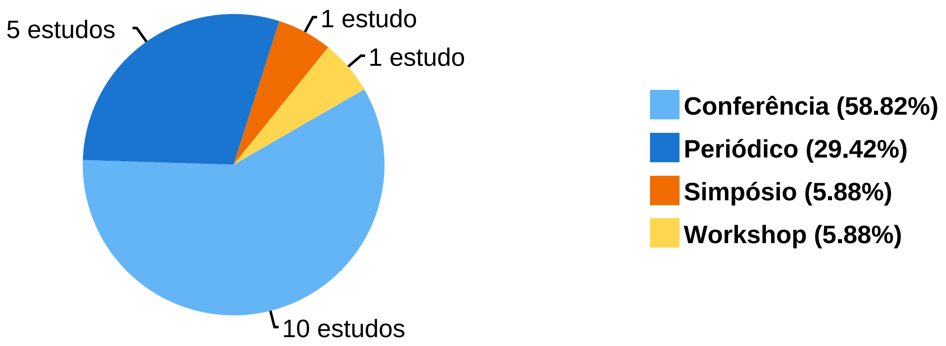
Ano	1994	2005	2006	2011	2012	2013	2015	2017	2021	2022
Estudos	E14	E13	E09	E01 E16	E04 E06 E08	E02 E10 E11	E12	E17	E03 E07 E15	E05

Fonte: Elaborada pelo autor.

e 2006, com dois estudos (11,76%). Dentre esses intervalos, o período entre 2011 a 2013 se destaca com a maior produção de estudos, porém o intervalo entre os anos de 2021 e 2022 merece destaque, pois contém a segunda maior concentração de estudos e indica uma tendência atual para publicações científicas relacionadas à temática do SMS conduzido neste projeto de mestrado acadêmico. Por outro lado, os anos de 1994, 2015 e 2017 forneceram publicações pontuais com um estudo (5,88%) para cada ano. De modo geral, essa evidencias revelam a relevância do tema de pesquisa abordado neste SMS, que possui um marco histórico no ano de 1994 e tem sido trabalhado mais ativamente a partir do ano de 2005 até o ano de 2022.

O segundo dado demográfico pode ser observado na Figura 12, que ilustra a distribuição da quantidade de estudos pelo tipo de publicação (**`QP1.2`**). Ao todo, foram identificados quatro tipos de publicação, a saber: (i) *conferência*; (ii) *periódico*; (iii) *simpósio* e (iv) *workshop*. A maior parte dos estudos está contida em duas categorias principais de publicação, a saber: (1) *conferência*, com 10 estudos (58,82%) publicados em *proceedings* de conferências; e (2) *periódico*, com cinco estudos (29,42%) publicados em revistas científicas. Além disso, para as publicações realizadas em simpósios e *workshops*, foi encontrado somente um estudo (5,88%) para cada um desses tipos de publicação. De acordo com a distribuição ilustrada na

Figura 12 – Distribuição dos estudos por tipo de publicação



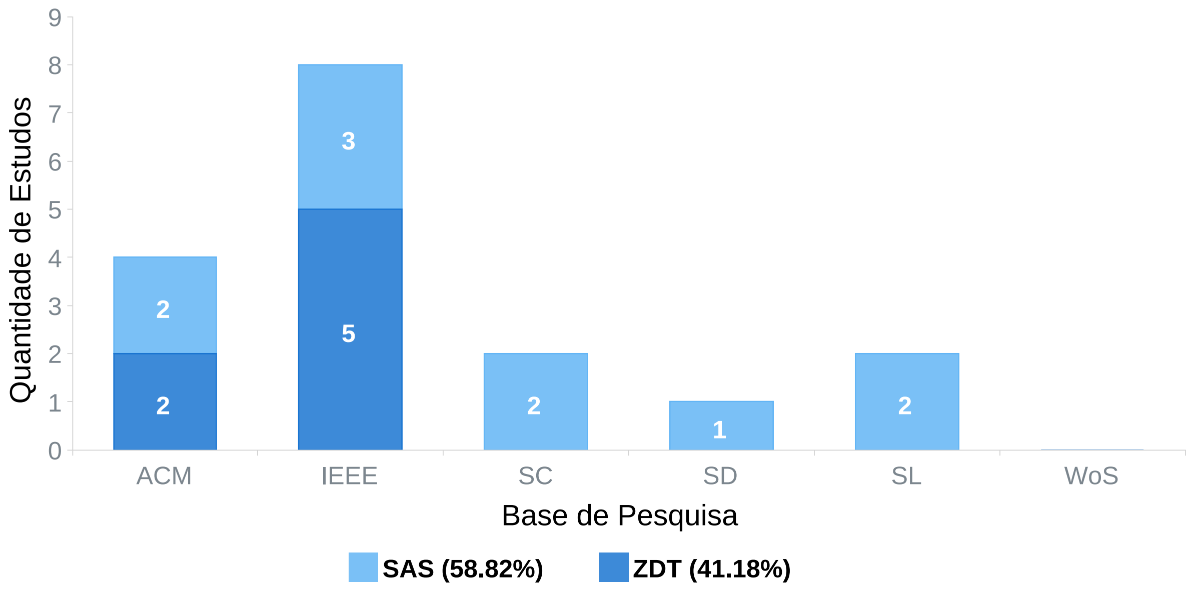
Tipo de Publicação	Estudos
Conferência	E01, E03, E06, E07, E08, E09, E10, E13, E14, E17
Periódico	E02, E04, E05, E11, E15
Simpósio	E16
Workshop	E12

Fonte: Elaborada pelo autor.

Figura 12, pode-se observar uma concentração de estudos publicados em conferências e periódicos (88,24%). Tais evidencias salientam a maturidade dos estudos relacionados com o tema de pesquisa do SMS conduzido neste projeto de mestrado. Como a palavra maturidade pode abranger diferentes aspectos de um trabalho acadêmico, neste SMS esse termo se refere aos estudos que apresentaram uma solução sobre o tema central de investigação e algum tipo de avaliação para a solução proposta. Com relação aos 10 estudos publicados em conferências, somente dois estudos (`E08` e `E14`) não abordaram algum tipo de avaliação da sua solução para a evolução do esquema de dados. De maneira semelhante, dentre as cinco publicações em periódicos, também foram encontrados dois estudos (`E02` e `E04`) que não reportaram uma maneira de avaliar a ideia proposta por sua solução. Além disso, vale lembrar que os estudos publicados em periódicos geralmente são mais robustos do que os publicados em conferências, pois costumam apresentar um embasamento teórico mais elaborado, além de revelar o funcionamento da sua solução proposta em mais detalhes.

O terceiro dado demográfico pode ser visualizado na Figura 13, que ilustra a distribuição da quantidade de estudos por base digital de publicação e FB (Frente de Busca) (**`QP1.3`**). Conforme mencionado no Apêndice B, foram estabelecidas duas FBs para ampliar a cobertura do SMS conduzido neste projeto de mestrado, a saber: (i) `SAS`, para encontrar os estudos específicos do domínio de SaS; e (ii) `ZDT`, para encontrar os estudos relacionados a outras áreas

Figura 13 – Distribuição dos estudos por base de pesquisa e frente de busca

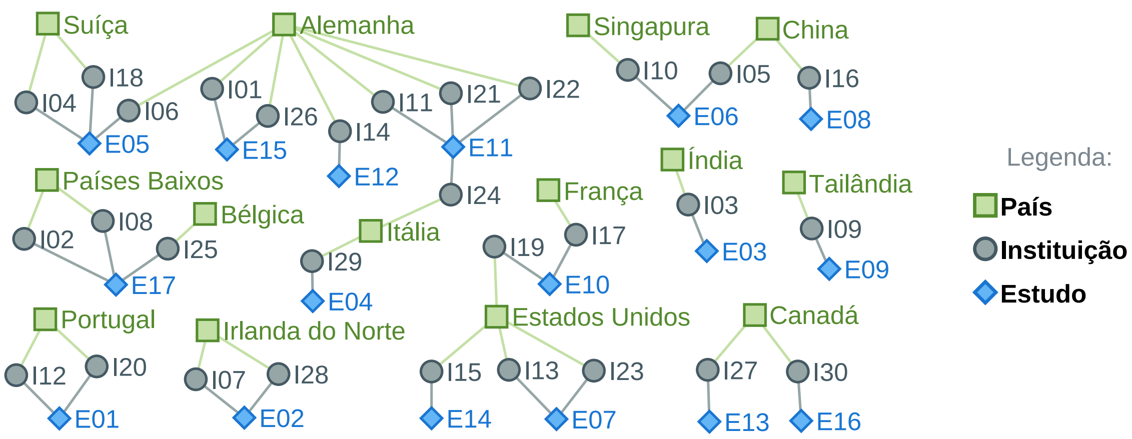


Fonte: Elaborada pelo autor.

de pesquisa que compartilham as mesmas necessidades e preocupações de um SaS (como os sistemas com tempo de inatividade zero, em inglês *zero-downtime*, por exemplo). Além disso, por questões de melhoria de visualização, os nomes das bases de pesquisa foram referenciados através da convenção de siglas apresentada na Tabela 2 do Apêndice B. Na Figura 13 pode-se observar que ambas as FBs contribuíram de maneira igualitária na cobertura das publicações, sendo que a `SAS` retornou 10 estudos (58,82%) e a `ZDT` obteve sete estudos (41,18%). Com relação a primeira FB, foram encontrados 10 estudos que aparecem de maneira mais distribuída entre cinco bases de pesquisa, a saber: (i) 3 estudos para *IEEE Xplore* (IEEE); (ii) 2 estudos para as bases *ACM Digital Library* (ACM), *Scopus* (SC) e *SpringerLink* (SL); e (iii) 1 estudo para *ScienceDirect* (SD). Por outro lado, para a segunda FB foram identificados sete estudos que estão concentrados em somente em duas bases de pesquisa, a saber: (i) 2 estudos para ACM; e (ii) 5 estudos para IEEE. De acordo com o gráfico apresentado na Figura 13, nenhum estudo relevante foi encontrado na base *Web of Science* (WoS), tanto para `SAS` quanto para `ZDT`. Diante desse cenário, tais evidencias indicam a existência de diferentes trabalhos acadêmicos que estão alinhados com a temática deste SMS, cujas publicações estão distribuídas entre diferentes bases de pesquisa e não representam um interesse específico de um veículo de publicação particular.

Por fim, o quarto dado demográfico pode ser visualizado na Figura 14, que ilustra a distribuição dos estudos de acordo com as instituições dos pesquisadores envolvidos e seus respectivos países de origem (**`QP1.4`**). Ao todo, foram identificadas 30 instituições diferentes, sendo

Figura 14 – Distribuição dos estudos por instituição e país



Instituições	
I01 - Darmstadt University of Applied Sciences	I16 - Tsinghua University
I02 - Delft University of Technology	I17 - Universite de Grenoble
I03 - Devi Ahilya University	I18 - University of Bern
I04 - ETH Zürich	I19 - University of California
I05 - Fudan University	I20 - University of Coimbra
I06 - Hamburg University of Applied Sciences	I21 - University of Dresden
I07 - IBM United Kingdom Limited	I22 - University of Magdeburg
I08 - Magnet.me	I23 - University of Maryland
I09 - Mahidol University	I24 - University of Milano
I10 - National University of Singapore	I25 - University of Namur
I11 - Philipps-University Marburg	I26 - University of Rostock
I12 - Polytechnic Institute of Coimbra	I27 - University of Toronto
I13 - ServiceNow	I28 - University of Ulster
I14 - Technische Universität Dresden	I29 - Università Roma Tre
I15 - The University of Michigan	I30 - Université du Québec à Montréal

Fonte: Elaborada pelo autor.

27 universidades e três empresas (*IBM United Kingdom Limited*, *Magnet.me* e *ServiceNow*). Em relação à distribuição geográfica, tais instituições foram localizadas em três continentes diferentes, a saber: (i) *América*, com cinco estudos (29,41%) relacionados aos países Canadá e Estados Unidos; (ii) *Ásia*, com quatro estudos (23,53%) relacionados aos países China, Índia, Singapura e Tailândia; e (iii) *Europa*, com nove estudos (52,94%) relacionados aos países Alemanha, Bélgica, França, Irlanda do Norte, Itália, Países Baixos, Portugal e Suíça. De acordo com o gráfico ilustrado na Figura 14, vale salientar que o estudo `E10` está relacionado com duas instituições diferentes, sendo uma europeia e outra americana. Além disso, dos 17 estudos selecionados neste mapeamento, nove trabalhos (52,94%) estão relacionados com pelo menos duas instituições diferentes (`E01`, `E02`, `E05`, `E06`, `E07`, `E10`, `E11`, `E15` e `E17`). Diante desse cenário, pode-se dizer que os pesquisadores que trabalham com uma temática relacionada a este SMS estão espalhados em diferentes regiões do planeta. Tal evidência indica que esse tema de pesquisa possui interesse global e não se concentra em um país específico.

3.1.2 QP2: Quais tipos de soluções são abordadas pelos estudos?

Esta QP teve como objetivo a classificação dos tipos de soluções identificados nos 17 estudos primários levantados por este SMS. O tema da evolução do esquema de dados foi abordado de diferentes maneiras em cada um desses estudos, cujas ideias centrais foram classificadas em torno de seis tipos de soluções distintas, a saber:

1. **Ferramenta:** neste tipo estão incluídas as soluções que abordaram a implementação e utilização de uma ferramenta para apoiar o processo de evolução do esquema de dados em um sistema de software existente. Tais soluções também estabeleceram um cenário de uso que especifica o domínio de aplicação, como operar, quando utilizar e em que parte do processo de evolução deve ser aplicada a ferramenta. Como exemplos desse tipo de solução, pode-se citar um utilitário que é capaz de alterar o esquema e os dados de um banco de dados de acordo com um conjunto de instruções de modificação escritas em um arquivo de metadados (MARKS; STERRITT, 2013), um programa que pode traduzir um esquema objeto-relacional em código Java orientado a objetos e entre outros tipos de traduções entre modelos com a migração de seus dados correspondentes (ATZENI *et al.*, 2012), um *plugin* que permite atualizar o esquema de dados de uma aplicação Java em execução (PUKALL *et al.*, 2013), entre outros exemplos de soluções;

2. ***Framework*:** neste tipo estão incluídas as soluções de natureza mais complexa que utilizaram uma infraestrutura de componentes para apoiar o processo de evolução do esquema de dados em um sistema de software. Tais componentes geralmente possuem uma funcionalidade bem definida, cujo trabalho conjunto permite a evolução do esquema de dados do sistema alvo. Como exemplo, Beurer-Kellner *et al.* (2022) criaram uma abordagem que permite o compartilhamento de dados entre serviços *web* que estão em constante mudança. Essa abordagem utiliza uma linguagem para definir a interface de dados, outra linguagem para especificar as funções de migração de dados e um ambiente operacional que permite a execução automática dessas funções de migração. Por outro lado, Götz e Kühn (2015) desenvolveram um novo tipo de mapeamento objeto-relacional (em inglês, *Object-Relational Mapping* – ORM) que automatiza as alterações no banco de dados conforme o esquema de dados de uma aplicação Java evolui. O funcionamento desse ORM se baseia na utilização um transformador de *bytecode* para identificar as ações de persistência de dados, uma base Prolog para armazenar as informações do esquema de

dados e um gerenciador de persistência para manipular o banco de dados;

3. ***Middleware*:** neste tipo estão incluídas as soluções que introduziram um novo componente entre um sistema de software existente e seus dados. Tal componente atua como uma camada intermediária, fazendo com que o sistema alvo precise utilizá-lo para acessar seus dados. Além disso, esse componente de *middleware* também é responsável pelo gerenciamento do esquema de dados, permitindo que o sistema alvo possa evoluir ao mesmo tempo que o *middleware* se encarrega de fazer as mudanças necessárias no esquema de dados. Como exemplos desse tipo de solução, pode-se citar um gerenciador para *data warehouse* que disponibiliza as informações de negócio em tempo real (SANTOS; BERNARDINO; VIEIRA, 2011), um *middleware* que ajusta as consultas SQL de uma aplicação de acordo com as versões de esquemas de dados presentes nessa aplicação e no seu banco de dados (SONG *et al.*, 2012), um gerenciador para lidar com a evolução de esquema e a migração dos dados de um banco de dados não relacional (HILLENBRAND *et al.*, 2021), um *middleware* responsável por permitir que várias versões de uma aplicação armazenem seus esquemas e compartilhem os dados comuns em um banco de dados relacional (JONG; DEURSEN; CLEVE, 2017), entre outros exemplos de soluções;

4. **Processo:** neste tipo estão incluídas as soluções que estabeleceram uma sequência de passos ou operações para executar o processo de evolução do esquema de dados em um sistema de software existente. Dessa maneira, essa sequência de passos constituí uma única grande tarefa que pode ser aplicada a cada vez que o esquema de dados do sistema alvo precise evoluir. Vale ressaltar que a sequência de passos do processo pode ser restrita a um ambiente operacional específico ou pode depender de outros tipos de tecnologias. Como exemplo desse tipo de solução, pode-se citar um processo de migração de banco de dados sem tempo de inatividade que se baseia nas tecnologias e ferramentas fornecidas pelo ambiente *Oracle Real Application Clusters* (WANG; DU; LIU, 2012);

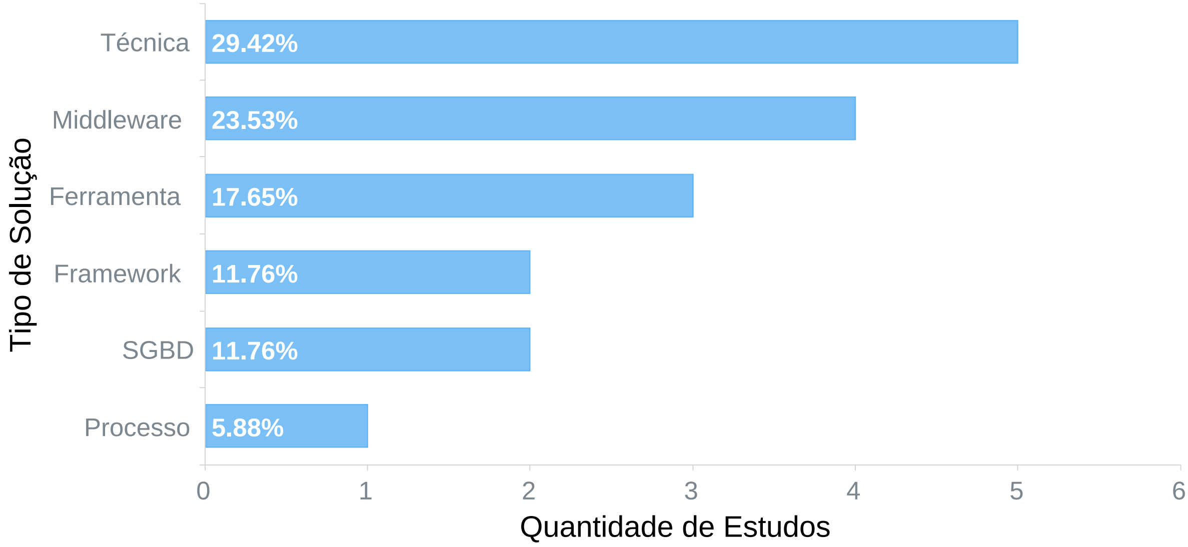
5. **SGBD:** neste tipo estão incluídas as soluções que propuseram a criação de um SGBD que implementa a evolução do esquema de dados como funcionalidade nativa. Esse tipo de solução geralmente se baseia no reaproveitamento do código fonte de um SGBD existente, cuja implementação é modificada para fornecer as funcionalidades que apoiam a evolução do esquema de dados. Como exemplos desse tipo de solução, pode-se citar uma extensão do banco de dados PostgreSQL que controla a evolução do seu esquema de dados através

da captura das instruções DDL (BHATTACHERJEE *et al.*, 2021), um banco de dados SQLite modificado que implementa um novo comando SQL para criar novas versões de esquema de dados (NEAMTIU *et al.*, 2013), entre outros exemplos de soluções; e

6. **Técnica:** neste tipo estão incluídas as soluções que desenvolveram uma metodologia para tratar o processo de evolução do esquema de dados em um sistema de software existente. Essas metodologias geralmente possuem uma natureza mais genérica e podem se basear na utilização de diferentes tecnologias, técnicas, ferramentas, entre outras dependências. Como exemplos desse tipo de solução, pode-se citar uma abordagem de migração de dados que combina a transferência de um *snapshot* (ou seja, a cópia de um instantâneo) do banco de dados e a captura em tempo real dos dados que estão sendo modificados (NAMDEO; SUMAN, 2021), uma técnica para o versionamento temporal de dados que precisam ser consultados por processos OLAP (MITRPANONT; FUGKEAW, 2006), um método de migração de dados para a alocação de bancos de dados sob emanda (SOUNDARARAJAN; AMZA, 2005), uma solução para a evolução do esquema de dados em um banco de dados orientado a objetos (RA; RUNDENSTEINER, 1994), uma abordagem autoadaptativa que permite a troca de dados após um processo de evolução do esquema de dados (ASSOUDI; LOUNIS, 2011), entre outros exemplos de soluções.

Na Figura 15 pode-se observar a distribuição da quantidade de estudos por tipo de solução. Dentre os seis tipos de soluções que foram identificadas, três tipos concentraram a maior quantidade de estudos, a saber: (i) *técnica*, com cinco estudos (29,42%); (ii) *middleware*, com quatro estudos (23,53%); e (iii) *ferramenta*, com três estudos (17,65%). Além disso, os tipos banco de dados e *framework* contribuíram cada um com dois estudos (11,76%) e somente um estudo (5,88%) apresentou uma solução do tipo processo. Diante desse cenário, observa-se uma predominância um pouco maior de soluções que abordam técnicas mais genéricas para tratar a evolução do esquema de dados. Além disso, é importante notar uma segunda tendência de soluções que introduzem um *middleware* entre o sistema alvo e seus dados. Tal tipo de solução possui uma predisposição mais natural, pois o *middleware* fica encarregado de administrar o processo de evolução do esquema dados, removendo essa responsabilidade de gerenciamento do sistema alvo. Por fim, pode-se dizer que as ferramentas são úteis para apoiar o processo de evolução do esquema de dados, porém podem ser consideras soluções mais limitadas dependendo do seu domínio e cenário de aplicação. Com base nessas observações, tais evidencias revelam a existência de diferentes tipos de soluções que estão alinhadas com o tema do SMS conduzido

Figura 15 – Distribuição dos estudos por tipo de solução



Tipo de Solução	Estudos
Técnica	E03, E09, E13, E14, E16
Middleware	E01, E06, E15, E17
Ferramenta	E02, E04, E11
Framework	E05, E12
SGBD	E07, E10
Processo	E08

Fonte: Elaborada pelo autor.

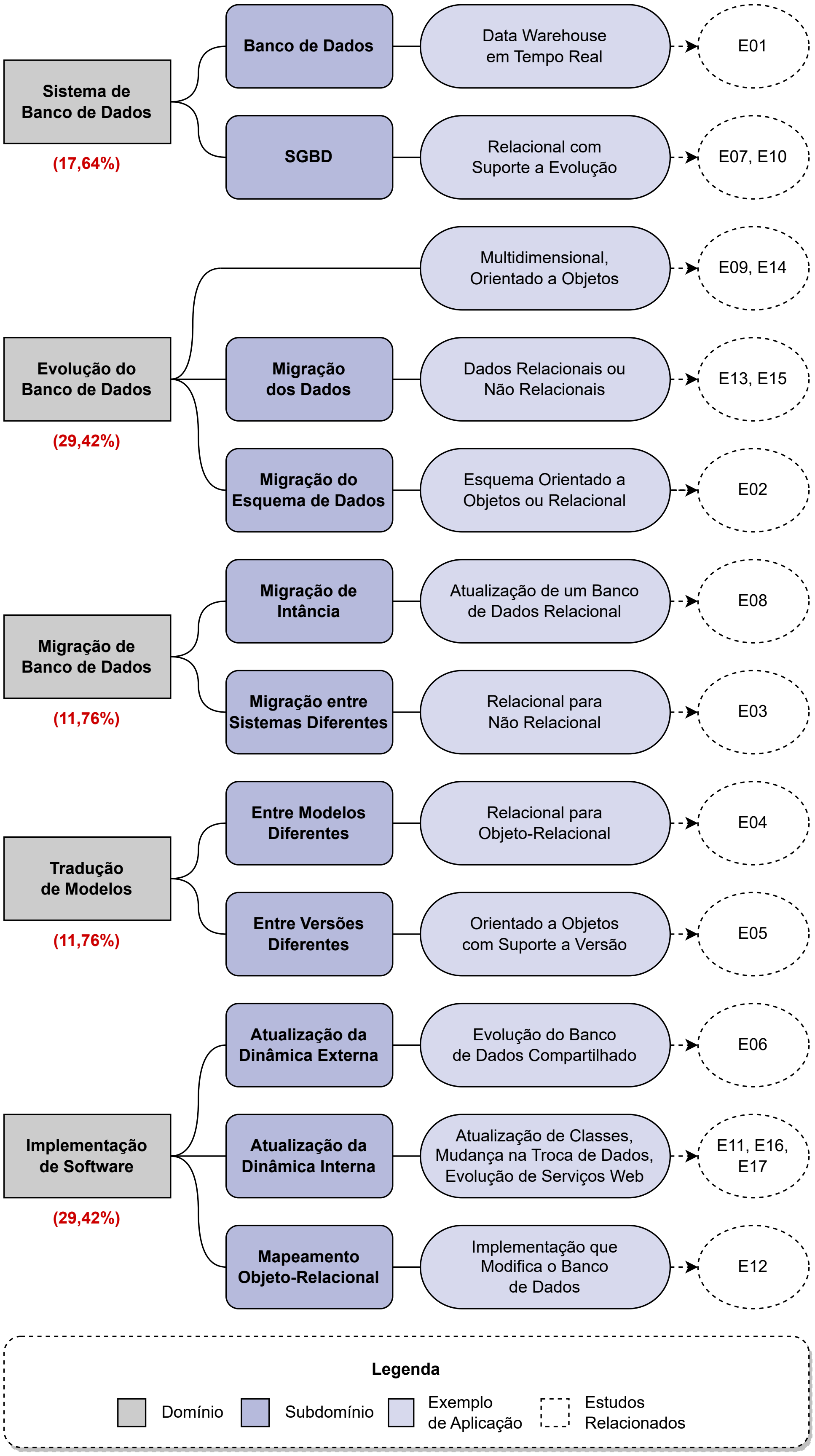
neste projeto de mestrado. Isso indica que esse problema pode ser trabalhado de diferentes maneiras e não existe uma abordagem padrão amplamente adotada para solucioná-lo.

3.1.3 `QP3`: Em quais domínios de aplicação as soluções propostas pelos estudos estão inseridas?

Esta QP teve como objetivo a identificação dos domínios e subdomínios em que estão inseridas as soluções propostas pelos estudos primários levantados por este SMS. Conforme ilustrado na Figura 16, foram identificados cinco tipos principais de domínios, a saber:

1. **Sistema de Banco de Dados:** neste domínio foram incluídos três estudos (17,65%) que apresentaram uma solução para a evolução do esquema de dados em algum aspecto de um sistema de banco de dados. Além disso, as soluções propostas por tais estudos foram classificadas em dois subdomínios diferentes, a saber: (i) *banco de dados*, com um estudo (5,88%) para a evolução do esquema em um tipo específico de banco de dados; e (i) *SGBD*,

Figura 16 – Domínios de aplicação abordados pelos estudos



Fonte: Elaborada pelo autor.

com dois estudos (11,76%) que desenvolveram uma implementação especial de SGBD com suporte nativo para a evolução do esquema de dados;

2. **Evolução do Banco de Dados:** neste domínio foram incluídos cinco estudos (29,42%) que desenvolveram uma solução para lidar com a evolução de um banco de dados, cujo processo deve transformar o esquema e preservar os dados que não foram removidos (SCHULER; KESSELMAN, 2021). Dentre os trabalhos incluídos neste domínio, dois estudos (11,76%) abordaram uma solução geral para a evolução do esquema em tipos específicos de banco de dados. Por outro lado, os estudos restantes apresentaram soluções mais específicas que foram classificadas em dois subdomínios principais, a saber: (i) *migração dos dados*, com dois estudos (11,76%) que trabalharam com o processo de transferência do conteúdo de uma instância de banco de dados para outra; e (ii) *migração do esquema de dados*, com um estudo (5,88%) que apresentou uma maneira de controlar o processo de modificação estrutural do esquema de banco de dados;

3. **Migração de Banco de Dados** neste domínio foram incluídos dois estudos (11,76%) que exploram algum tipo de migração de banco de dados. Tais soluções foram classificadas em dois subdomínios, a saber: (i) *migração de instância*, com um estudo (5,88%) que apresentou uma maneira de transferir o estado do banco de dados para outro do mesmo tipo; e (ii) *migração entre sistemas diferentes*, com um estudo (5,88%) que desenvolveu um método para transferir o estado de um banco de dados para outro de tipo diferente;

4. **Tradução de Modelos:** neste domínio foram incluídos dois estudos (11,76%) que apresentaram uma maneira de transferir dados entre modelos com esquemas distintos. Nesse contexto, foram encontrados dois tipos de tradução de modelos, a saber: (i) *entre modelos diferentes*, com um estudo (5,88%) que abordou uma solução para mapear a estrutura de um tipo de modelo para outro com tipo diferente, ao mesmo tempo que transfere os dados entre os modelos de origem e destino; e (ii) *entre versões diferentes*, com um estudo (5,88%) que desenvolveu uma solução para transferir os dados entre dois modelos do mesmo tipo e com o mesmo propósito semântico, mas com esquemas estruturais diferentes devido a evolução de um dos modelos para a versão mais atualizada; e

5. **Implementação de Software:** neste domínio foram incluídos cinco estudos (29,42%) que abordaram a evolução do esquema de dados em algum aspecto da implementação de um sistema de software. As soluções apresentadas nesses estudos foram classificadas em três

subdomínios, a saber: (i) *atualização da dinâmica externa*, com um estudo (5,88%) que lidou com a diferença de esquemas de dados entre aplicações que compartilham o mesmo banco de dados; (ii) *atualização da dinâmica interna*, com três estudos (17,65%) que apresentaram uma maneira para modificar o esquema de dados de uma aplicação que está em constante evolução; e (iii) *mapeamento objeto-relacional*, com um estudo (5,88%) que abordou uma solução para modificar o banco de dados conforme as classes que mapeiam as entidades e relacionamentos do banco de dados precisem mudar para acomodar novas funcionalidades de uma aplicação.

3.1.4 **QP4:** Quais tipos de modelos de dados são abordados pelos estudos?

Esta QP teve como objetivo a identificação dos tipos de modelos de dados que foram abordados pelos estudos primários levantados neste SMS. As soluções apresentadas em cada um desses estudos abordaram a temática da evolução do esquema de dados em diferentes áreas de aplicação, cujos modelos de dados adotados foram categorizados em quatro tipos, a saber:

1. **Modelo de Banco de Dados:** neste tipo estão incluídas as soluções que abordaram a evolução para um esquema de banco de dados. Essas soluções geralmente apresentam um cenário que envolve o processo de aprimoramento contínuo de um sistema de software ao mesmo tempo que seu esquema do banco de dados precisa ser atualizado para acomodar as novas funcionalidades. Dessa maneira, o banco de dados deve ser modificado para refletir a estrutura do esquema de dados adotado pelo sistema a cada novo ciclo de aprimoramento. Além disso, é importante saber que cada solução pode adotar um tipo diferente de banco de dados que melhor atende às necessidades da sua área de aplicação. Dentre os estudos primários selecionados neste SMS, foram identificados três tipos principais de bancos de dados, a saber: (i) *relacional*; (ii) *não relacional*; e (iii) *orientado a objetos*. Vale destacar que a definição de um esquema de dados explicito não é exigida por um banco de dados não relacional, porém um sistema de software que faz seu uso precisa conhecer esse esquema para manipular os dados de maneira correta. Como exemplos desse tipo de solução, pode-se citar um *middleware* para gerenciar o esquema de dados de um *data warehouse* relacional (SANTOS; BERNARDINO; VIEIRA, 2011), uma ferramenta para automatizar o processo de atualização de um banco de dados relacional (MARKS; STERRITT, 2013), uma técnica para migrar os dados de um banco relacional em tempo real para outro banco de dados não relacional (NAMDEO; SUMAN, 2021), um *middleware* para ajustar

as consultas SQL de uma aplicação de acordo com as versões de esquemas de dados presentes nessa aplicação e em seu banco de dados relacional (SONG *et al.*, 2012), um SGBD relacional especial cuja implementação possui suporte nativo a evolução do esquema de dados (BHATTACHERJEE *et al.*, 2021; NEAMTIU *et al.*, 2013), uma técnica que permite a evolução do esquema de dados em um banco orientado a objetos (RA; RUNDENSTEINER, 1994), entre outros exemplos de soluções;

2. **Modelo Orientado a Objetos:** neste tipo estão incluídas as soluções que propuseram uma maneira de controlar a evolução das classes em um sistema de software orientado a objetos. Tais soluções apresentaram um cenário de aplicação semelhante ao tipo de modelo de banco de dados, porém o processo de evolução é tratado no esquema de dados orientado a objetos do sistema alvo. Além disso, vale salientar que esse tipo de solução se concentra na manutenção das classes do sistema alvo, cujos dados ativos podem existir durante o tempo de execução ou persistidos em um repositório de dados. Como exemplos desse tipo de solução, pode-se citar um *framework* para traduzir diferentes versões de instância de modelos de dados orientados a objetos (BEURER-KELLNER *et al.*, 2022), uma técnica para o versionamento temporal de dados com base em um *data warehouse* relacional e um conjunto de classes (MITRPANONT; FUGKEAW, 2006), uma ferramenta que permite a atualização do código fonte das classes de uma aplicação Java em execução (PUKALL *et al.*, 2013), entre outros exemplos de soluções;

3. **Modelo Objeto-Relacional:** neste tipo estão incluídas as soluções que desenvolveram alguma técnica para a evolução das classes em sistema de software orientado a objetos que estão relacionadas com a persistência de dados. Essas classes modelam as entidades e os relacionamentos presentes em um banco de dados, permitindo que o sistema consiga executar as funcionalidades de leitura, armazenamento e alteração dos seus dados. Como exemplos desse tipo de solução, pode-se citar uma ferramenta que permite a tradução de um modelo objeto-relacional para código fonte em linguagem orientada a objetos (ATZENI *et al.*, 2012), uma implementação de mapeamento objeto-relacional que permite a evolução simultânea do esquema de dados de uma aplicação Java e seu banco de dados (GöTZ; KüHN, 2015), entre outros exemplos de soluções; e

4. **Modelo Genérico:** neste tipo estão incluídas as soluções que não exigiram a utilização de um modelo de dados específico. Tais tipos de solução geralmente apresentam uma

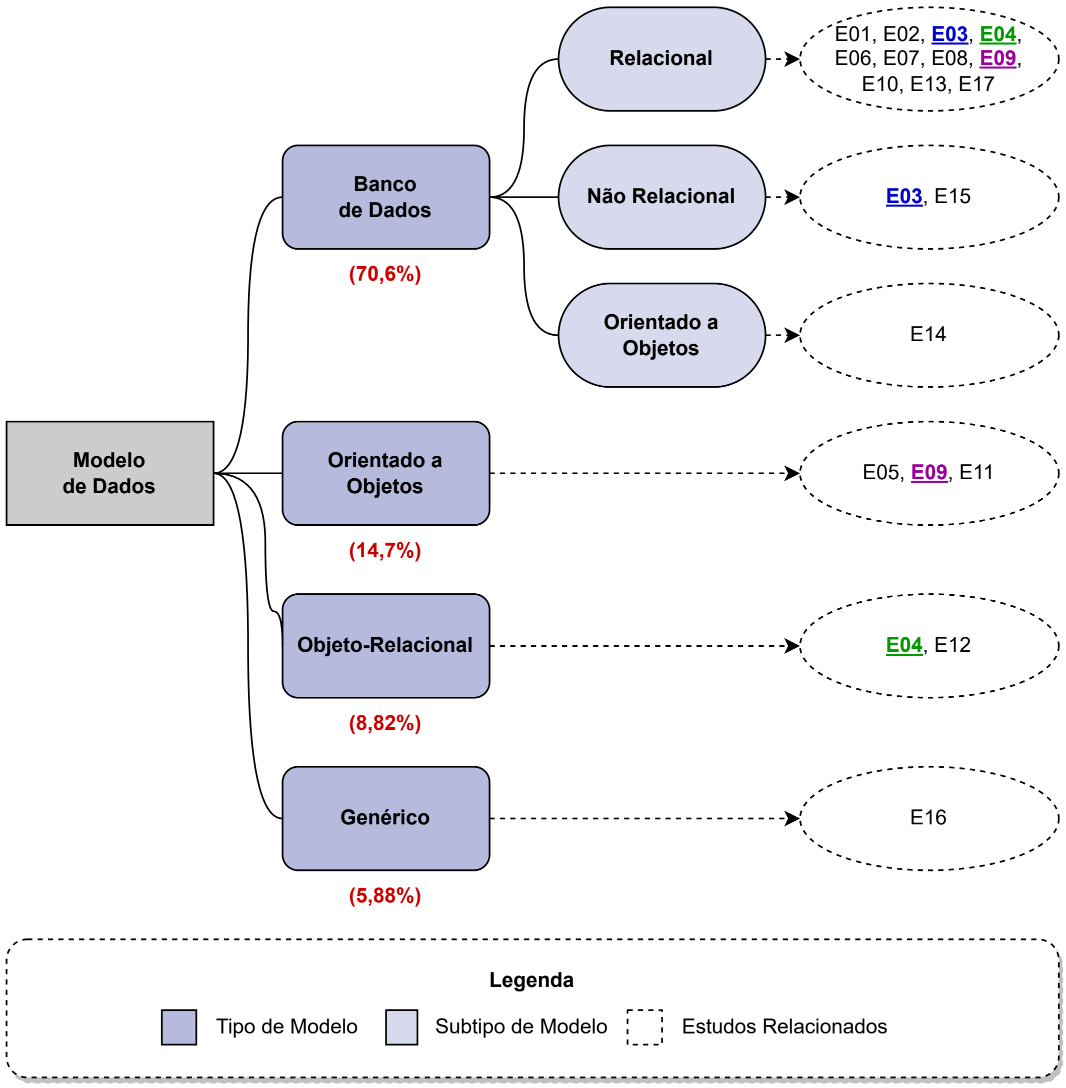
abordagem de evolução de dados em um nível mais alto de abstração, cuja utilização prática pode exigir algum tipo de ajuste dependendo do modelo de dados que foi adotado e do contexto abordado pelo cenário de aplicação. Como exemplo desse tipo de solução, pode-se citar uma técnica autoadaptativa para a troca de dados após a evolução de um esquema genérico de dados (ASSOUDI; LOUNIS, 2011).

Na Figura 17 é ilustrada a distribuição dos tipos de modelos de dados e seus respectivos estudos primários. Dentre os quatro tipos de modelos de dados identificados neste SMS, o modelo de banco de dados se destaca com uma concentração de 13 estudos (76,47%). Tais estudos abordaram a utilização de três tipos de banco de dados, a saber: (i) *relacional*, com 11 estudos (64,71%); (ii) *não relacional*, com dois estudos (11,76%); e (iii) *orientado a objetos*, com um estudo (5,88%). É importante salientar que o estudo `E03` apresentou uma solução que trabalha simultaneamente com os modelos de banco de dados relacional e não relacional. Por outro lado, o modelo orientado a objetos contribuiu com três estudos (17,65%) e o modelo objeto-relacional com dois estudos (11,76%). Além disso, foi encontrado somente um estudo (5,88%) que abordou uma solução envolvendo o modelo de dados genérico. De acordo com o esquema apresentado na Figura 17, também vale destacar a ocorrência de duas intersecções entre modelos de dados diferentes, a saber: (i) *o estudo* `E04`, que abordou uma solução que utiliza os modelos de banco de dados relacional e objeto-relacional; e (ii) *o estudo* `E09`, que desenvolveu uma solução que envolve os modelos de banco de dados relacional e orientado a objetos. Diante desse cenário, essa evidencias destacam a existência de diversas pesquisas alinhadas com a evolução do esquema de dados para diferentes tipos de modelos de dados. Tal constatação indica que o tema de pesquisa abordado neste SMS não está restrito a um tipo específico ou padronizado de modelo de dados.

3.1.5 QP5: Quais tipos de abordagens são utilizadas pelos estudos para tratar o processo de evolução do esquema de dados?

Esta QP teve como objetivos: (i) a identificação de como os estudos primários levantados por este SMS perceberam o processo de evolução do esquema de dados; e (ii) o levantamento de quais abordagens foram adotadas para tratar esse problema. De maneira geral, tais estudos apresentaram diversas soluções para tratar a evolução do esquema de dados em diferentes domínios e cenários de aplicação. Apesar dessa variedade de soluções, foi identificado um

Figura 17 – Tipos de modelos de dados abordados pelos estudos



Fonte: Elaborada pelo autor.

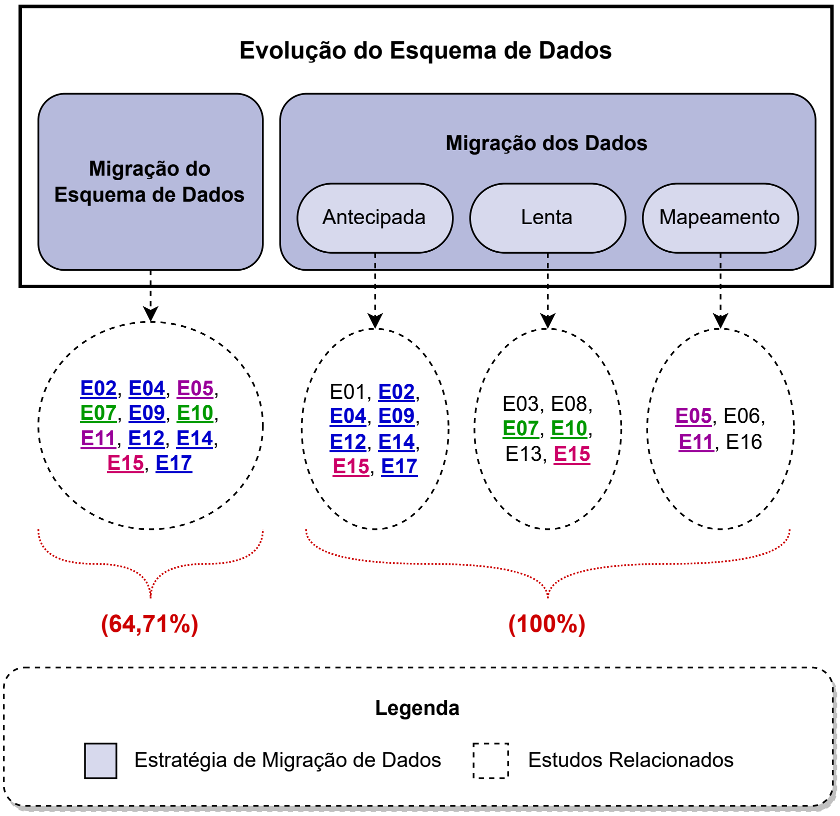
padrão de resolução implícito nas abordagens propostas pelos 17 estudos selecionados neste SMS. Conforme relatado por Stiemer *et al.* (2021), a evolução de esquemas de bancos de dados é uma abordagem que envolve duas etapas principais, a saber: (i) *a evolução/migração do esquema de dados*; e (ii) *a migração dos dados*. Essa abordagem está alinhada com a visão de Schuler e Kesselman (2021), que definem a evolução do esquema de banco de dados como um processo de transformação do esquema que preserva os dados em operações não destrutivas (por exemplo, as que não envolvem a eliminação de uma tabela ou coluna). É importante ressaltar que as duas etapas levantadas por Stiemer *et al.* (2021) foram especificamente identificadas no domínio de banco de dados com esquemas flexíveis, como os bancos de dados não relacionais, por exemplo. Entretanto, essa abordagem é implicitamente seguida pelas soluções de evolução do esquema de dados apresentadas nos estudos levantados no SMS conduzido neste projeto de mestrado. Além disso, vale lembrar que essas soluções podem apresentar uma abordagem que cobre simultaneamente as duas etapas levantadas por Stiemer *et al.* (2021) ou somente uma delas. Por outro lado, Schuler e Kesselman (2021) revelaram a existência de outras abordagens complementares no contexto da evolução do banco de dados, como a limpeza de dados e técnicas ETL (do inglês, *Extraction, Transformation and Loading*), por exemplo. Nesse sentido, as ideias contidas em tais abordagens podem ser reaproveitadas no contexto da evolução do esquema de dados para aprimorar as etapas de migração do esquema de dados e de migração dos dados. A seguir, essas duas etapas são abordadas com mais detalhes:

Migração do Esquema de Dados. Esta etapa consiste na aplicação de operações de evolução para transformar o esquema de dados antigo na sua versão mais nova (HILLENBRAND *et al.*, 2020). Tais operações modificam a estrutura de dados antiga para a nova versão respeitando o tipo e formato de dado de cada campo/valor (por exemplo, como um texto de vinte caracteres) (AZEROUAL; JHA, 2021). É importante salientar que a migração do esquema de dados é uma atividade difícil e pode exigir a execução de uma sequência bem coordenada de operações complexas ou não usuais (SCHULER; KESSELMAN, 2021). Além disso, uma mudança de esquema aparentemente simples pode exigir um processo de migração computacionalmente custoso que envolve a execução de uma grande sequência de operações de transformação. Como exemplos de abordagens para esta etapa da evolução do esquema de dados, pode-se citar: (i) a utilização de um arquivo de metadados para especificar a sequência de operações de modificação que transformam o esquema de dados atual na sua versão estrutural mais nova (MARKS; STERRITT, 2013); (ii) uma técnica de tradução que identifica a sequência de operações necessária para transformar o esquema de dados antigo na sua versão mais nova (ATZENI *et al.*, 2012); (iii) uma solução de transformação automática onde o desenvolvedor escreve funções para traduzir pares de versões de esquema de dados diferentes (BEURER-KELLNER *et al.*, 2022); (iv) um método baseado em grafo dirigido que estabelece uma hierarquia de versões de esquemas de dados (MITRPANONT; FUGKEAW, 2006); (v) uma abordagem que fornece visões diferentes para cada versão estrutural de um esquema de dados global (RA; RUNDENSTEINER, 1994); entre outros exemplos de soluções.

Migração dos Dados. Esta etapa consiste na transferência dos dados contidos no esquema antigo para o novo. Vale salientar que essa transferência não envolve necessariamente o esvaziamento dos dados do esquema antigo, pois existem abordagens para a evolução do esquema de dados que mantém os dados antigos para alguma finalidade. Como exemplo, pode-se citar a técnica proposta por Mitrpanont e Fugkeaw (2006), que utiliza um tipo de versionamento para manter o estado do esquema e seus respectivos dados em diferentes intervalos de tempo. Além disso, a migração dos dados pode adotar três estratégias principais (HILLENBRAND *et al.*, 2020; STIEMER *et al.*, 2021), a saber: (i) *antecipada*, que migra todos os dados instantaneamente; (ii) *lenta*, que migra os dados à medida que são requisitados ou através de uma fila de migração; e (iii) *mapeamento*, que não migra os dados propriamente, mas utiliza algum método em tempo execução para mapear os dados entre esquemas diferentes. Com relação a primeira estratégia de migração, os dados ficam disponíveis instantaneamente para a aplicação, porém é possível que haja um tempo de espera quando muitos dados precisem ser transferidos. Para a segunda estratégia de migração, o tempo de espera é reduzido com a transferência progressiva dos dados, porém a aplicação pode recuperar consultas inconsistentes até que todos os dados sejam migrados. Por fim, a terceira estratégia de migração não envolve a transferência de dados, porém é possível que a aplicação tenha que lidar com valores nulos e com o aumento do tempo de tradução causada pela defasagem progressiva. Como exemplos de abordagens para esta etapa, pode-se citar: (i) a utilização de uma fila para migar os dados (NAMDEO; SUMAN, 2021); (ii) uma técnica baseada em mapa de *bits* para gerenciar a migração dos dados (BHATTACHERJEE *et al.*, 2021); (iii) um método que utiliza um vetor de versões para controlar a migração dos dados (SOUNDARARAJAN; AMZA, 2005); (iv) uma abordagem que utiliza um *middleware* especializado para migrar os dados após um ciclo de evolução do esquema de dados (HILLENBRAND *et al.*, 2021); (v) a adoção de um mecanismo de gatilhos (em inglês, *triggers*) para realizar a replicação automática dos dados (JONG; DEURSEN; CLEVE, 2017); entre outros exemplos de soluções.

Na Figura 18 é ilustrada a abordagem de evolução do esquema de dados composta pelas etapas de migração do esquema e migração dos dados. Dentre os estudos selecionados neste SMS, a etapa de migração do esquema de dados foi abordada por 11 estudos (64,71%). Por outro lado, todos os 17 estudos (100%) apresentaram algum tipo de estratégia de migração dos dados, sendo observada a seguinte distribuição de estratégias: (i) *antecipada*, com oito estudos (47,06%); (ii) *lenta*, com seis estudos (35,29%); e (iii) *mapeamento*, com quatro estudos (23,53%). Vale salientar que o estudo `E15` apresentou uma abordagem de migração dos dados que pode ser ajustada para trabalhar com uma estratégia antecipada ou lenta. De acordo com

Figura 18 – Abordagem para a evolução do esquema de dados



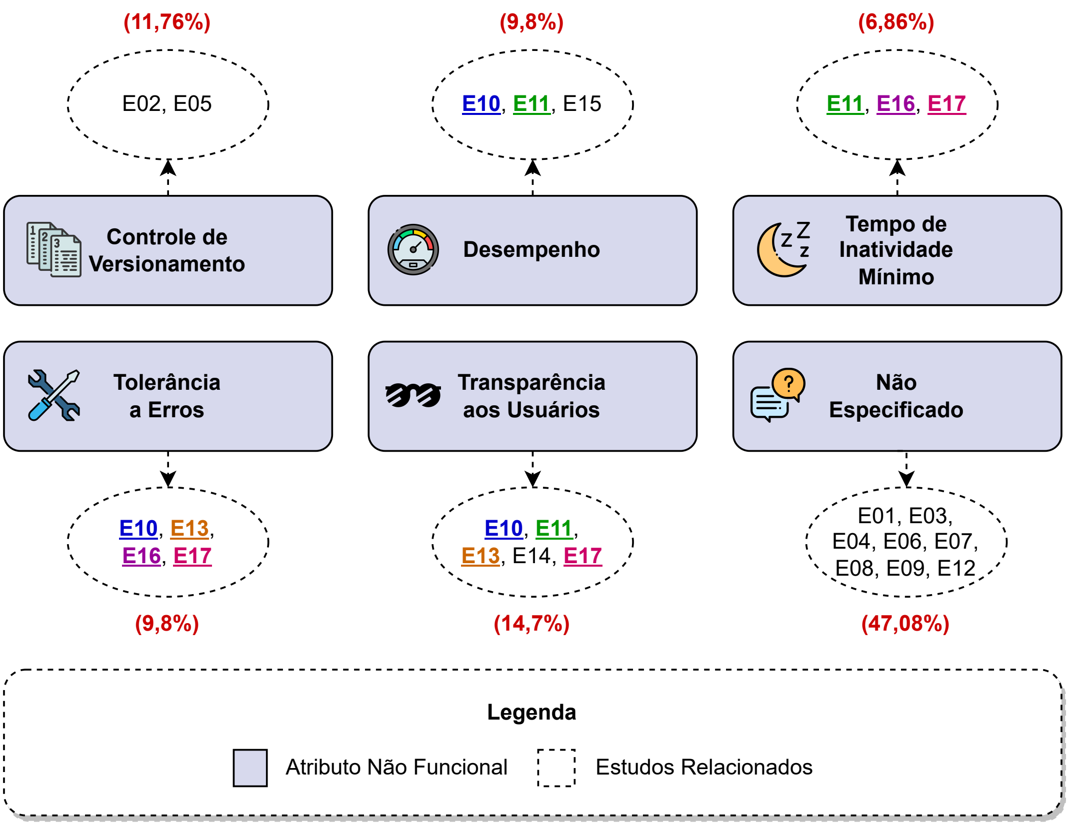
Fonte: Elaborada pelo autor.

essas informações, pode-se notar uma diferença de cobertura entre as duas etapas envolvidas na abordagem de evolução do esquema de dados apresentada na Figura 18. A etapa de migração dos dados foi identificada em todos os estudos, aparentando ser uma atividade mais fácil de lidar devido à grande oferta de soluções para essa finalidade. Em contrapartida, a etapa da migração do esquema de dados foi abordada em pouco mais da metade dos estudos levantados pelo SMS conduzido neste projeto de mestrado. Conforme reportado por Schuler e Kesselman (2021), a migração do esquema de dados é um problema difícil de lidar e exige a adoção de uma solução mais sofisticada. De modo geral, essa evidencias indicam que as soluções para a evolução do esquema de dados geralmente seguem uma abordagem de resolução dividida nas etapas de migração do esquema e migração dos dados. Nesse sentido, pode-se dizer que as soluções alinhadas com o tema de pesquisa abordado neste SMS apresentam um padrão de solução implícito que é composto pelas duas etapas supracitadas.

3.1.6 QP6: Quais atributos não funcionais foram considerados no projeto das solução propostas pelos estudos?

Esta QP teve como objetivo a identificação dos atributos não funcionais que foram considerados durante a fase de projeto das soluções propostas nos estudos primários levantados por este SMS. De acordo com Sommerville (2019), os atributos não funcionais influenciam diretamente no funcionamento de um sistema de software, cuja escolha do seu projeto arquitetural deve atender às necessidades desses requisitos não funcionais. Conforme ilustrado na Figura 19, foram identificados seis atributos não funcionais que apareceram com maior frequência nos 17 estudos selecionados por este SMS, a saber:

Figura 19 – Atributos não funcionais abordados pelos estudos



Fonte: Elaborada pelo autor.

1. **Controle de Versionamento:** este atributo foi abordado por dois estudos (11,76%) e estabelece um requisito de gerenciamento das versões de esquema de dados. Nesse sentido, tais estudos apresentaram uma solução para a evolução do esquema de dados que se baseia no gerenciamento das versões estruturais do esquema que surgem com o passar do tempo;

2. **Desempenho:** este atributo foi abordado por três estudos (17,65%) e estipula um requisito para a performance da solução para a evolução do esquema de dados. Dessa maneira, esses estudos buscaram propor soluções que não afetem o desempenho do sistema de software alvo ou pelo menos apresentem algum tipo de degradação tolerável;

3. **Tempo de Inatividade Mínimo:** este atributo foi abordado por três estudos (17,65%) e determina um requisito para o tempo de inatividade que a solução para a evolução do esquema de dados causa no sistema de software alvo. Dependendo dos cenários de aplicação aos quais esses estudos estão submetidos, as soluções propostas podem admitir um tempo de inatividade mínimo ou inexistente;

4. **Tolerância a Erros:** este atributo foi abordado por quatro estudos (23,53%) e estabelece um requisito para lidar com a ocorrência de erros durante o processo de evolução do banco de dados. Nesse sentido, tais estudos apresentam soluções que incorporam funcionalidades de recuperação de desastres para tratar o aparecimento de erros inesperados;

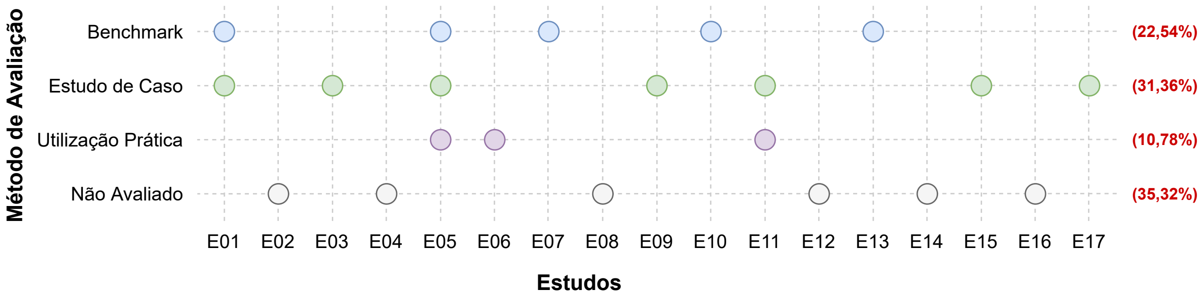
5. **Transparência aos Usuários:** este atributo foi abordado por cinco estudos (29,42%) e determina um requisito de como a solução para a evolução do esquema de dados é percebida pelo sistema alvo ou seus usuários. Dessa maneira, esses estudos buscaram desenvolver soluções que escondem sua presença do sistema alvo ou seus usuários, evitando que tais agentes precisem mudar sua estrutura interna ou alterar seu comportamento de funcionamento para adotar a solução proposta; e

6. **Não Especificado:** foram encontrados oito estudos (47,06%) que não estabeleceram um ou mais requisitos para a construção da sua solução para a evolução do esquema de dados. Tal condição indica que esses estudos não apresentaram ou mencionaram nenhum atributo não funcional no texto da sua respectiva publicação.

3.1.7 **QP7:** Como as soluções propostas pelos estudos foram avaliadas?

Esta QP tem como objetivo a identificação de como os estudos primários levantados pelo SMS realizaram a avaliação de sua solução para a evolução do esquema de dados. Vale salientar que esses estudos abordaram diferentes cenários e domínios de aplicação, cujas soluções propostas atendem a conjunto de requisitos específico. Nesse sentido, é esperado que os métodos de avaliação utilizados por tais estudos verifiquem se suas soluções satisfazem esses requisitos de interesse. De acordo com a Figura 20, foram identificadas quatro abordagens de avaliação nos 17 estudos selecionados por este SMS, a saber:

Figura 20 – Métodos de avaliação abordados pelos estudos



Fonte: Elaborada pelo autor.

1. ***Benchmark*****:** esta abordagem foi adotada em cinco estudos (29,42%) e estabelece uma metodologia para avaliar algum aspecto da solução para a evolução do esquema de dados. Existem diferentes tipos de *benchmark* para uma variedade de finalidades, como testar o desempenho de suporte a decisão de um *data warehouse*, medir o tempo de execução de uma solução para a migração de dados; avaliar o consumo de recursos de hardware, verificar a flexibilidade de uma abordagem para a migração de esquemas de dados, entre outros objetivos. Além disso, vale lembrar que é possível criar um *benchmark* personalizado para atender às necessidades de um projeto específico, porém costuma-se adotar um ou mais *benchmarks* amplamente testados e adotados pela comunidade científica;

2. **Estudo de Caso:** esta abordagem foi adotada em sete estudos (41,18%) e utiliza um ou mais casos de uso para avaliar a solução para a evolução do esquema de dados. Os autores dos estudos elaboram seus casos de uso visando apresentar um cenário de aplicação que contenha todos os requisitos que foram considerados no projeto da sua solução. Diante do exposto, espera-se que os casos de uso sirvam como instrumento para verificar se a solução proposta atende aos seus requisitos de projeto;

3. **Utilização Prática:** esta abordagem foi adotada em três estudos (17,65%) e avalia a solução para a evolução do esquema de dados através de sua aplicação em um sistema de software existente. Essa abordagem pode ser considerada um tipo especial de estudo de caso que contém um cenário de aplicação "não controlado" pelos autores dos estudos. Nesse sentido, espera-se que a utilização prática da solução sirva como meio de verificar

sua aplicabilidade em um sistema de software real. Vale lembrar que esse tipo de teste prático pode revelar a existência de outros requisitos que não foram considerados pelos autores na etapa de projeto da sua solução proposta; e

4. **Não Avaliado:** foram encontrados seis estudos (35,29%) que não estabeleceram uma maneira de avaliar sua solução proposta para a evolução do esquema de dados. Isso significa que esses estudos não apresentaram ou mencionaram nenhum tipo de avaliação no texto da sua respectiva publicação.

3.2 Discussão de Resultados

Com base nos resultados levantados pelas QPs da Seção 3.1, pode-se estabelecer uma análise comparativa entre os 17 estudos levantados por este SMS. É importante salientar que o SMS conduzido neste trabalho de mestrado acadêmico buscou ser fiel aos textos originais dos estudos, interpretando os dados em casos estritamente necessários. Inicialmente, a **`QP1`** buscou presentar um panorama sobre a relevância da temática de evolução do esquema de dados na literatura científica. Conforme reportado na Seção 3.1.1, as publicações sobre esse assunto cobrem um intervalo de 29 anos de pesquisa que contém períodos de produção que variam com o tempo (**`QP1.1`**), sendo que o último período produtivo entre os anos de 2021 e 2022 destaca uma possível tendência na investigação das soluções para a evolução do esquema de dados. Os estudos selecionados neste SMS foram publicados em conferências, periódicos, simpósios e *workshops* (**`QP1.2`**), cujas referências estão indexadas nas bases de pesquisa ACM, IEEE, SC, SD e SL (**`QP1.3`**). Além disso, esses estudos foram produzidos por diversos pesquisadores filiados a 27 universidades e três empresas espalhados entre os continentes americano, asiático e europeu (**`QP1.4`**). Em seguida, as **`QP2`** a **`QP7`** buscaram apresentar um panorama sobre as características das soluções para a evolução do esquema de dados, cujo resumo comparativo pode ser observado na Figura 21. A seguir, uma síntese do levantamento de cada QP é apresentada:

`QP2.` Esta QP se preocupou com a identificação da essência das soluções, cujas ideias centrais abordaram o desenvolvimento de ferramentas, *frameworks*, *middlewares*, processos, SGBDs e técnicas para apoiar a evolução do esquema de dados. A distribuição desses tipos de soluções pode ser vista com mais detalhes na Figura 15.

`QP3.` Nesta QP foram levantados os domínios de aplicação que adotaram as soluções propos-

Figura 21 – Resumo comparativo entre os estudos

Estudos	QP2 Solução	QP3 Domínio	QP4 Modelo de Dados	QP5 Apresenta Etapa da Abordagem de Evolução		QP6 Apresenta Atributo Não Funcional	QP7 Apresenta Método de Avaliação
				Migração do Esquema	Migração dos Dados		
E01	Middleware	Sistema de Banco de Dados	Banco de Dados	✘	✔	✘	✔
E02	Ferramenta	Evolução do Banco de Dados	Banco de Dados	✔	✔	✔	✘
E03	Técnica	Migração de Banco de Dados	Banco de Dados	✘	✔	✘	✔
E04	Ferramenta	Tradução de Modelos	Banco de Dados e Objeto-Relacional	✔	✔	✘	✘
E05	Framework	Tradução de Modelos	Orientado a Objetos	✔	✔	✔	✔
E06	Middleware	Implementação de Software	Banco de Dados	✘	✔	✘	✔
E07	SGBD	Sistema de Banco de Dados	Banco de Dados	✔	✔	✘	✔
E08	Processo	Migração de Banco de Dados	Banco de Dados	✘	✔	✘	✘
E09	Técnica	Evolução do Banco de Dados	Banco de Dados e Orientado a Objetos	✔	✔	✘	✔
E10	SGBD	Sistema de Banco de Dados	Banco de Dados	✔	✔	✔	✔
E11	Ferramenta	Implementação de Software	Orientado a Objetos	✔	✔	✔	✔
E12	Framework	Implementação de Software	Objeto-Relacional	✔	✔	✘	✘
E13	Técnica	Evolução do Banco de Dados	Banco de Dados	✘	✔	✔	✔
E14	Técnica	Evolução do Banco de Dados	Banco de Dados	✔	✔	✔	✘
E15	Middleware	Evolução do Banco de Dados	Banco de Dados	✔	✔	✔	✔
E16	Técnica	Implementação de Software	Genérico	✘	✔	✔	✘
E17	Middleware	Implementação de Software	Banco de Dados	✔	✔	✔	✔

Fonte: Elaborada pelo autor.

tas, sendo importantes para compreender as particularidades envolvidas nos contextos e cenários de utilização das soluções para a evolução do esquema de dados. Esses domínios podem ser vistos com mais detalhes na Figura 16.

QP4. Esta QP ficou responsável por identificar os modelos de dados abordados pelas soluções de evolução do esquema de dados, cujas especificidades estão relacionadas com a maneira como essas soluções gerenciam e armazenam seus dados. Esses modelos de dados podem ser vistos com mais detalhes na Figura 17.

QP5. Conforme descrito na Seção 3.1.5, nesta QP foi identificada uma abordagem para a evolução do esquema de dados que foi implicitamente seguida pelos estudos selecionados neste SMS, sendo composta pelas etapas de migração do esquema e migração dos dados. A cobertura dessas etapas pode ser visualizada na Figura 21, onde um círculo verde com um símbolo de visto indica que a solução do estudo correspondente aborda a etapa marcada. Caso contrário, um círculo vermelho com um símbolo de cruz indica que a etapa marcada não é abordada pela solução do estudo correspondente. É importante salientar que essa mesma simbologia de cobertura é utilizada tanto na QP6 quanto na QP7.

QP6. Esta QP foi planejada para verificar a presença de requisitos de projeto nas soluções para a evolução do esquema de dados, cujos atributos não funcionais mais frequentes giram em torno cinco aspectos, a saber: (i) *controle de versionamento*; (ii) *desempenho*; (iii) *tempo de inatividade mínimo*; (iv) *tolerância a erros*; e (v) *transparência aos usuários*. Caso uma solução apresente um requisito de projeto, seu respectivo estudo é marcado na Figura 21 através de um círculo verde com um símbolo de visto. Por outro lado, as soluções que não mencionaram nenhum dos requisitos supracitados no texto da sua publicação são marcadas através de um círculo vermelho com um símbolo de cruz.

QP7. Esta QP foi responsável por verificar se as soluções para a evolução do esquema de dados foram avaliadas pelos seus autores. Esse tipo de teste é importante para averiguar a validade de aplicação das soluções propostas, podendo ser feita através de métodos baseados em *benchmarks*, estudos de caso e utilizações práticas, por exemplo.

Durante a condução deste SMS, especificamente na fase de extração de informações, foram identificadas quatro limitações nos estudos selecionados, a saber:

Poucas soluções no domínio de SaS. Apesar dos cuidados tomados no protocolo de pesquisa descrito no Apêndice B, não foi encontrado nenhum estudo que se declarou pertencente ao domínio de SaS. Isso significa que os estudos selecionados por este SMS estão inseridos em outros domínios correlacionados com a área de SaS. O estudo `E16` destacou-se nesse contexto, pois foi o único trabalho que abordou a utilização de algumas propriedades autoadaptativas. Além disso, esse estudo apresentou uma solução para a evolução do esquema de dados que é muito semelhante ao laço de adaptação MAPE.

Imprecisão do termo autoadaptativo. Dentre os estudos selecionados neste SMS, o estudo `E15` foi o único que se considerou autoadaptativo. Entretanto, esse estudo apresentou uma solução de ajuste automático para a evolução do esquema de dados que não pode ser considerada autoadaptativa. De acordo com a definição apresentada na Seção 2.2, um sistema autoadaptativo é capaz de modificar sua dinâmica interna para responder as mudanças no seu contexto operacional (SALEHIE; TAHVILDARI, 2009). Nesse sentido, o entendimento do termo autoadaptativo neste projeto de mestrado acadêmico se baseia em duas ideias, a saber: (i) a ocorrência de alguma mudança no contexto operacional do SaS que interfere em seu funcionamento; e (ii) a necessidade do SaS em modificar sua estrutura interna para superar a degradação do contexto operacional. Portanto, um software considerado autoadaptativo deve estar correlacionado simultaneamente com essas ideias.

Falta de clareza e concisão nas explicações. Durante a fase de leitura dos estudos, foi percebido que muitos trabalhos apresentam suas soluções través de textos longos com muito formalismo matemático. Esse tipo de redação possui grande importância na construção de conhecimentos acadêmicos formais, porém a maioria dos estudos está preocupada com os aspectos práticos, deixando de lado a exploração de provas matemáticas e outros assuntos teóricos. Nesse sentido, existem maneiras mais elegantes e concisas para apresentar o funcionamento dessas soluções para a evolução do esquema de dados. Como exemplo, os cenários de aplicação da solução podem ser abordados através de casos de uso, o funcionamento da solução pode ser ilustrado através de um diagrama de sequência, a implementação da solução pode ser apresentada através de um diagrama de classes, entre outros tipos técnicos de descrição.

Engenharia de software pouco explorada. Os estudos selecionados neste SMS geralmente reportam suas atividades através de textos estruturados. Nesse sentido, os autores desses estudos buscam um equilíbrio entre a apresentação da fundamentação teórica e o funcionamento da sua solução proposta. Durante a fase de leitura, foi possível notar uma carência no desenvolvimento de ideias através de modelos e a exploração de questões de projeto. Os atributos não funcionais foram a única informação sobre o projeto das soluções que foi abordado por pouco mais da metade dos estudos. Além disso, vale destacar que existem vários aspectos de um projeto de software que podem ser explorados, como a definição dos seus requisitos funcionais e não funcionais, a escolha de uma arquitetura baseada em critérios, a exploração de padrões de *design*, entre outras questões de projeto.

De modo geral, pode-se notar que a evolução do esquema de dados é uma área de pesquisa que possui grande potencial de exploração em diferentes cenários de aplicação. A partir das informações reportadas na Seção 3.1, foi possível obter uma visão inicial de como esse assunto é abordado na literatura científica. Além disso, durante o processo de extração e análise dos dados, foram percebidas algumas limitações nos estudos selecionados neste SMS. Portanto, pode-se dizer que a evolução do esquema de dados é um tema de pesquisa pouco investigado, principalmente com relação a comunidade de SaS e as soluções para esse domínio.

3.3 Considerações Finais

Este capítulo apresentou uma visão geral sobre a evolução do esquema de dados para o domínio de SaS. Inicialmente, na Seção 3.1 foram apresentadas as respostas para cada QP, conforme descrito no protocolo de pesquisa elaborado no Apêndice B. Em seguida, na Seção 3.2 foi realizada uma análise comparativa entre os 17 estudos selecionados pelo SMS conduzido neste projeto de mestrado. De acordo com os resultados dessa análise, foi percebido que a evolução do esquema de dados é um tema de pesquisa pouco explorado, principalmente com relação a comunidade de SaS e as soluções propostas para lidar com esse domínio de aplicação. Além disso, tais resultados também foram levados em consideração na elaboração da biblioteca DynaSchema desenvolvida neste projeto de mestrado acadêmico.

4 Biblioteca DynaSchema

Neste capítulo são apresentados os detalhes de projeto e *design* da biblioteca DynaSchema desenvolvida neste projeto de mestrado. Inicialmente, na Seção 4.1 é abordado um exemplo de execução que contextualiza e delimita o escopo do problema enfrentado por essa biblioteca, além de descrever os detalhes de projeto sobre seus componentes funcionais, conjunto de funcionalidades e arquitetura interna. Em seguida, na Seção 4.2 são reportados os detalhes de *design* dos seis módulos internos da biblioteca DynaSchema e da sua interface com seus usuários. Por fim, na Seção 4.3 são apresentadas as considerações finais deste capítulo.

4.1 Projeto da Biblioteca

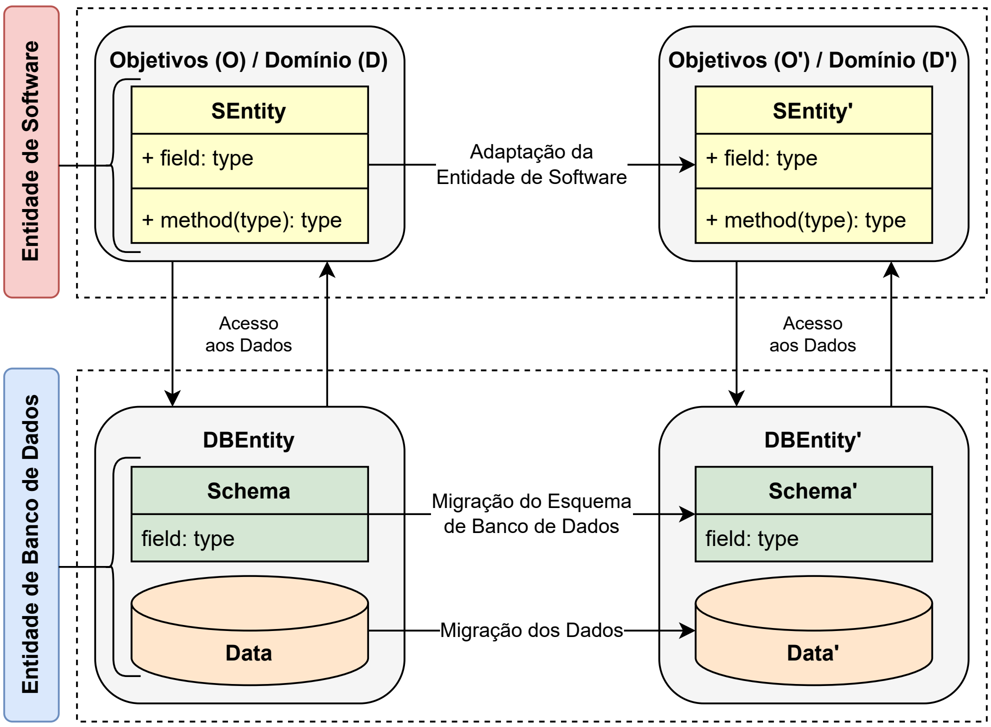
A biblioteca DynaSchema é uma solução de software que permite lidar com o problema da evolução do esquema de dados no domínio de SaS. Para isso, essa biblioteca atua como um *middleware* (Seção 3.1.2) entre um SaS e seus dados armazenados em um banco de dados relacional, permitindo que o SaS possa se autoadaptar ao mesmo tempo que a biblioteca é responsável por evoluir o esquema de banco de dados. É importante salientar que a evolução do esquema de dados é um processo amplo que envolve a migração tanto do esquema (ou seja, as tabelas do banco de dados relacional) quanto dos dados (ou seja, os registros armazenados) (Seção 3.1.5). Por razões de escopo de projeto, a biblioteca DynaSchema considerou as necessidades da evolução de um SaS orientado a objetos, baseado na arquitetura de referência RA4SaS e que utiliza um modelo objeto-relacional (Seção 3.1.4) para persistir seus dados em um banco de dados relacional. Essas restrições nortearam o escopo da biblioteca, porém seu projeto se preocupou em atender as necessidades gerais de um SaS genérico com persistência de dados.

Com relação a organização dessa seção, a apresentação do projeto da biblioteca DynaSchema foi estruturado em quatro seções. Inicialmente, na Seção 4.1.1 é abordado um exemplo de execução para esclarecer a problemática enfrentada por essa biblioteca. Em seguida, na Seção 4.1.2 são descritos os componentes funcionais do seu contexto de funcionamento. Na Seção 4.1.3 são apresentados os requisitos funcionais que serviram de base para projetar a biblioteca. Por fim, na Seção 4.1.4 é descrita sua arquitetura interna, destacando uma organização modular e as funcionalidades de cada um dos módulos da biblioteca DynaSchema.

4.1.1 Exemplo de Execução

Na Figura 22 é ilustrado um exemplo de execução que delimita o contexto e o escopo do problema enfrentado pela biblioteca DynaSchema. Tal exemplo de execução aborda o cenário de autoadaptação de um SaS organizado em duas camadas principais. Na primeira camada estão contidas as entidades de software, que é um termo genérico para se referir a um SaS. Essas entidades estão relacionadas com o modelo de dados da aplicação, cuja estrutura está sujeita às mudanças de cada novo ciclo de autoadaptação do SaS. Por outro lado, na segunda camada estão contidas as entidades de banco de dados que armazenam o esquema e os dados da aplicação. Vale lembrar que esse esquema armazena a estrutura das tabelas do banco de dados e está diretamente relacionado com o modelo de dados do SaS. Isso quer dizer que existe uma dependência entre a estrutura das entidades de software e o esquema de banco de dados.

Figura 22 – Exemplo de execução da autoadaptação de um SaS



Fonte: Elaborada pelo autor.

Inicialmente, pode-se observar na Figura 22 que uma entidade de software `SEntity` acessa seus dados através de uma entidade de banco de dados `DBEntity`. Nesse contexto, a entidade de software `SEntity` é projetada para atender um certo conjunto de objetivos `O` e para operar em um domínio `D`. Por outro lado, a entidade de banco de dados `DBEntity` contém o esquema de baco de dados `Schema` e os registros armazenados `Data` utilizados pelo SaS. Através de um gatilho externo que dispara a adaptação, é possível que o SaS detecte uma nova necessidade e passe por um ciclo de autoadaptação, onde a entidade de software `SEntity` é adaptada para `SEntity'`. Assim, a nova entidade `SEntity'` possui uma estrutura para lidar com os objetivos da autoadaptação `O'` e para operar no domínio de melhoria `D'`. Consequentemente, a entidade de banco de dados `DBEntity` também deve ser modificada para `DBEntity'`, acompanhando as mudanças promovidas pelo ciclo de autoadaptação do SaS. Para isso, devem ocorrer dois tipos de migração, a saber: (i) *uma migração do esquema de banco de dados*, transformando a antiga estrutura do esquema `Schema` na nova versão `Schema'`; e (ii) *uma migração de dados*, movendo os registros armazenados `Data` do esquema antigo (ou seja, `Schema`) para `Data'`, cujo conteúdo está de acordo com a nova versão estrutural do esquema de banco de dados (ou seja, `Schema'`).

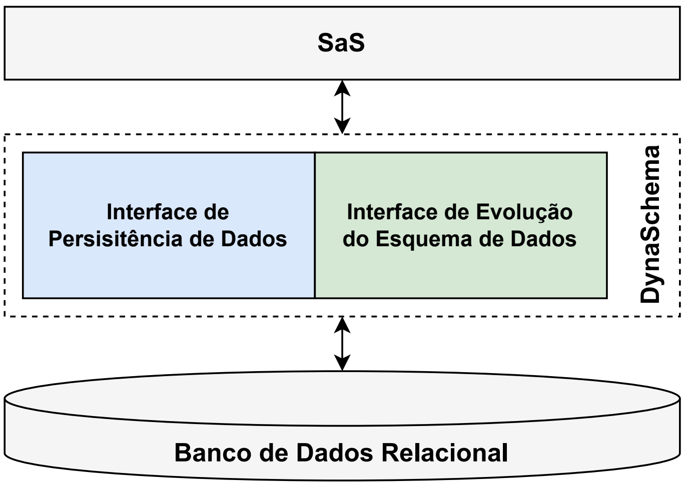
De acordo com o exemplo de execução ilustrado na Figura 22, pode-se dizer que o processo de autoadaptação de um SaS com persistência de dados envolve três atividades principais, a saber: (1) *a adaptação da entidade de software*; (2) *a migração do esquema de banco de dados*; e (3) *a migração dos dados*. Além disso, tal processo de autoadaptação pode ser explorado a partir de um ponto de vista operacional através de dois cenários de execução. No primeiro cenário **(A)**, o processo de adaptação das entidades está em andamento e pode ser existir a necessidade de manter a execução de duas versões da mesma entidade de software (ou seja, `SEntity` e `SEntity'`). Isso exige a preservação de duas versões do esquema de banco de dados (ou seja, `Schema` e `Schema'`) que podem compartilhar dados de maneira simultânea. Nesse contexto são executadas as atividades 1 e 2 do processo de autoadaptação do SaS. No segundo cenário de execução **(B)**, não existem chamadas para a entidade de software antiga (ou seja, `SEntity`), permitindo somente a manutenção da execução da nova entidade (ou seja, `SEntity'`). Nesse contexto são executadas as três atividades do processo de autoadaptação do SaS. Adicionalmente, a antiga entidade de banco de dados (ou seja, `DBEntity`) pode ser apagada para liberar espaço de armazenamento no banco de dados relacional utilizado pelo SaS.

A partir do exemplo ilustrado na Figura 22 podem ser levantadas várias preocupações com relação à evolução do esquema de dados no domínio de SaS. Nesta seção foram apresentados alguns dos cenários enfrentados por um SaS que precisa armazenar seus dados em um banco de dados relacional. Tais cenários evidenciam a importância da biblioteca DynaSchema como uma nova solução para gerenciar o processo de evolução do esquema de banco de dados. Desse modo, o SaS pode se concentrar na adaptação das suas entidades (Atividade 1), enquanto a biblioteca gerencia o processo de migração do esquema e dos dados em si (Atividades 2 e 3).

4.1.2 Componentes Funcionais

Na Seção 4.1.1 foram levantados alguns dos desafios enfrentados em um cenário de evolução do esquema de dados no domínio de SaS. Para lidar com esses problemas, a biblioteca DynaSchema pode estabelecer uma camada intermediária (ou seja, um *middleware*) entre o SaS e seu banco de dados, conforme ilustrado na Figura 23. Nesse sentido, o SaS utiliza essa biblioteca como um mecanismo de persistência, permitindo que suas informações possam ser armazenadas e consultadas em um banco de dados relacional. Quando um gatilho externo de adaptação é acionado, o SaS passa por um ciclo de autoadaptação estrutural e utiliza a biblioteca DynaSchema para realizar a evolução do seu esquema de banco de dados. Assim, o SaS deve informar à biblioteca quais alterações foram feitas no modelo de dados da aplicação e quando as ações de evolução podem ser executadas. É importante salientar que o tempo de execução do processo de evolução do esquema pode aumentar conforme a quantidade de dados armazenados cresce. Por esse motivo, tal processo deve ser dividido em etapas, permitindo que o SaS possa coordenar as chamadas de cada uma dessas etapas e escolher o melhor momento para executá-las. Assim, o SaS pode decidir quando um novo esquema de banco de dados deve ser criado, qual o melhor momento para migrar os dados da versão antiga do esquema de dados para a nova versão e quando pode ser eliminada a versão antiga do esquema de dados.

Figura 23 – Contexto de funcionamento da biblioteca DynaSchema



Fonte: Elaborada pelo autor.

O contexto ilustrado na Figura 23 salienta um agrupamento lógico das funcionalidades fornecidas pela biblioteca DynaSchema. O primeiro grupo está relacionado com a persistência de dados (por exemplo, inserção e consulta) executada pelo SaS, já o segundo grupo está associado com o processo de evolução do esquema de banco de dados em si. Por esse motivo, a biblioteca DynaSchema pode ser dividida em dois componentes funcionais bem definidos, a saber:

Interface de Persistência de Dados. Este componente está relacionado com as funcionalidades de persistência de dados fornecidas pela biblioteca DynaSchema, modelando a maneira como as operações de inserção, alteração, exclusão e consulta de dados devem ser disponibilizadas. De acordo com as evidências do SMS apresentado no Capítulo 3, grande parte dos estudos apresenta uma implementação própria para lidar com a evolução do esquema de dados no domínio de SaS. Essa característica dificulta a ampla adoção de alguma dessas soluções propostas, pois os desenvolvedores de software precisam aprender como suas implementações podem ser utilizadas na construção de um SaS. Diante desse cenário, a biblioteca DynaSchema foi desenvolvida na linguagem Java[1] e seu projeto de implementação se preocupou em reaproveitar os conhecimentos adotados pela comunidade que utiliza essa linguagem. A JPA (do inglês, *Java Persistence API*) é uma especificação que permite lidar com o mapeamento objeto-relacional, onde as classes devem ser anotadas para que os dados dos seus objetos sejam persistidos no banco de dados. Sua primeira especificação foi lançada em 2006[2], estabelecendo uma tecnologia que foi testada e aprimorada ao longo dos anos[3], além de ser amplamente adotada pelos desenvolvedores Java. Vale lembrar que a JPA somente especifica as interfaces do mecanismo de persistência, assim é necessário adotar uma implementação específica, com o Hibernate[4] ou EclipseLink[5], para utilizar suas funcionalidades de fato. Motivado por esse contexto, a biblioteca DynaSchema contém uma implementação especial da JPA, permitindo que um desenvolvedor Java reaproveite seus conhecimentos na criação de um SaS que utiliza essa biblioteca. Dessa maneira, só é necessário aprender como a biblioteca lida com a evolução do esquema de dados, uma vez que a interface do mecanismo de persistência segue uma especificação que já é extensamente aceita e validada pela comunidade Java.

Interface de Evolução do Esquema de Dados. Este componente está relacionado com as funcionalidades de evolução do esquema de dados fornecidas pela biblioteca DynaSchema,

[1] A biblioteca DynaSchema foi desenvolvida utilizando a versão 11 da linguagem Java fornecida pela Eclipse Temurin JDK (do inglês, *Java Development Kit*), disponível em: <https://adoptium.net>
[2] A primeira especificação da JPA pode ser encontrada em: <https://www.jcp.org/en/jsr/detail?id=220>
[3] A especificação mais recente da JPA, ou seja a versão 3.1 de 2022, pode ser encontrada em: <https://jakarta.ee/specifications/persistence/3.1>
[4] O projeto Hibernate pode ser encontrado em: <https://hibernate.org>
[5] O projeto EclipseLink pode ser encontrado em: <https://eclipse.dev/eclipselink>

modelando a sequência de etapas que devem ser executadas para realizar o processo de evolução desse esquema. Conforme levantado no SMS apresentado no Capítulo 3, grande parte das soluções para a evolução de esquema de banco de dados já utilizam uma abordagem implícita composta de duas etapas (Seção 3.1.5). A primeira está relacionada com as operações para transformar o antigo esquema de banco de dados na sua versão mais nova, já a segunda se preocupa com a transferência dos dados da versão antiga do esquema para a nova versão. Diante desse cenário, a biblioteca DynaSchema apresenta um processo de evolução do esquema de dados alinhado com a execução dessas etapas, permitindo que um SaS possa escolher o melhor momento para executá-las.

4.1.3 Conjunto de Funcionalidades

Conforme reportado no Capítulo 3, foram elaboradas sete questões de pesquisa para compreender como a literatura científica tem lidado com a temática da persistência de dados para o domínio de SaS e os problemas relacionados à evolução do esquema de dados para acompanhar os ciclos de autoadaptação desse tipo de sistema de software. Em seguida, foi realizada uma análise comparativa de 17 estudos primários (listados na Tabela 1 do Apêndice A) com base nas sete questões supracitadas, sendo que a discussão sobre os resultados dessa análise foi apresentada na Seção 3.2. Além disso, na Seção 4.1.1 foram abordadas algumas das preocupações relacionadas com esse tema de pesquisa, cujas soluções para esse problema podem apresentar algum dos componentes funcionais reportados na Seção 4.1.2. De acordo com essas descobertas, foi elaborado um conjunto de seis funcionalidades principais (**F1** a **F6**) que devem ser oferecidas pela biblioteca DynaSchema, a saber:

[F1] Um SaS com persistência de dados deve ser capaz de alterar sua estrutura interna ao mesmo tempo que mantém o acesso aos dados que estavam armazenados em um banco de dados relacional. Nesse sentido, a evolução do esquema de dados envolve a manipulação de dois tipos esquemas, a saber: (i) *o esquema orientado a objetos*, que reflete a estrutura interna do SaS (ou seja, as entidades de software); e (ii) *o esquema de banco de dados*, que reflete a estrutura das tabelas que armazenam os dados do SaS. Para ilustrar a complexidade da manutenção do sincronismo entre esses dois esquemas, pode-se citar os cenários de decomposição de classes (especialização) e a quebra de relacionamentos de herança (generalização). O primeiro cenário exige minimamente a criação de uma nova tabela para cada nova subclasse, além da definição de chaves estrangeiras para relacionar essas tabelas (ou seja, estabelecendo uma maneira de agrupar os dados que estavam originalmente em uma única tabela). Em relação ao segundo cenário, devem ser eliminadas as tabelas que representam as subclasses e suas colunas inseridas na tabela que representa a superclasse. Para melhorar a separação de responsabilidades, a biblioteca DynaSchema deve agir como uma camada intermediária entre o SaS e o banco de dados (ou seja, um *middleware*), podendo gerenciar a evolução do esquema de banco de dados ao mesmo tempo que permite que o SaS acesse seus dados de maneira mais transparente. Assim, o SaS pode se concentrar seus esforços no processo de autoadaptação das suas entidades de software.

[F2] Os SaS que armazenam seus dados em um banco de dados relacional possuem características comuns em relação à evolução do esquema de dados, pois a estrutura interna desses tipos de sistemas deve ser projetada de modo que sua alteração não interrompa o funcionamento da aplicação. De acordo com esse cenário, a biblioteca DynaSchema deve permitir que a migração do esquema de dados do SaS seja feita em tempo de execução. Além disso, a investigação conduzida no SMS do Capítulo 3 revelou que à medida que o volume dos dados cresce, o processo de migração de dados se torna mais caro e pode impor algum tempo de inatividade para acessar esses dados. Portanto, a biblioteca DynaSchema deve adotar um mecanismo de migração de dados assíncrono com disparo de mensagens de notificação, permitindo que um SaS solicite uma migração de esquema e seja notificado pela biblioteca sobre o andamento desse processo. Vale salientar que a operação de consulta dos dados pode ser afetada durante o processo de migração, por esse motivo a biblioteca deve notificar o SaS quando a migração dos dados estiver finalizada para evitar que a aplicação recupere dados parciais.

[F3] Quando o SaS realiza um ciclo de autoadaptação, a estrutura de um componente interno pode ser alterada e reinserida no ambiente operacional. Durante o processo de autoadaptação pode existir um estado transitório onde as partes ainda não adaptadas do SaS utilizam a versão antiga do esquema de dados, enquanto as partes que já foram adaptadas utilizam a versão mais recente desse esquema. Portanto, a biblioteca DynaSchema deve permitir que o SaS acesse seus dados em uma condição de estado misto, onde existe a necessidade de interagir com as versões antiga e nova do esquema de dados simultaneamente. Além disso, deve existir um mecanismo de sincronização de dados para que as alterações feitas na versão antiga do esquema de dados sejam refletidas na nova versão e vice-versa.

[F4] O processo de autoadaptação de um SaS pode exigir uma série de modificações no esquema de dados para refletir as alterações realizadas nas suas entidades de software. Por sua vez, essas alterações também podem exigir algum tipo de mapeamento de dados entre as versões antiga e nova do esquema de dados da aplicação. Para ilustrar esse cenário, pode-se citar um caso em que uma tabela contém uma coluna numérica para categorização dos dados, onde cada categoria é representada por um número inteiro positivo. Após um ciclo de autoadaptação, o SaS pode otimizar a semântica dessa informação através de um rótulo alfanumérico para cada categoria numérica. Nesse caso, a nova versão do esquema de dados exige que a antiga categoria numérica seja convertida para o seu rótulo alfanumérico correspondente. De acordo com esse cenário, a biblioteca DynaSchema deve permitir que o SaS especifique um mapeamento de dados, cuja informação será utilizada para migrar os dados da versão antiga do esquema para a nova versão.

[F5] O desenvolvimento de um SaS não é uma tarefa trivial, pois a necessidade de realizar as modificações em tempo de execução exige o levantamento de várias questões de projeto para que a aplicação funcione de maneira correta. Além disso, novos desafios são encontrados quando esse tipo de sistema é combinado com o requisito de persistência de dados. Analisando os SGBDs relacionais disponíveis na literatura, pode-se observar que cada SGBD adota um dileto SQL distinto com possíveis diferenças sintáticas na declaração das operações que modificam o banco de dados. Por exemplo, a instrução para criar uma coluna de incremento automático pode ser diferente dependendo do produto de banco de dados utilizado. Para obter esse efeito, deve ser utilizada a palavra chave `IDENTITY` no Microsoft SQL Server, já no MariaDB a coluna deve ser declarada como `AUTO_INCREMENT`. Portanto, a biblioteca DynaSchema deve estar preparada para lidar com diferentes tipos de bancos de dados relacionais, permitindo que novos dialetos SQL sejam inseridos sem que alterações significativas precisem ser feitas no seu código fonte.

[F6] Um ciclo de autoadaptação permite que a estrutura das entidades de software do SaS evoluam, consequentemente o esquema de banco de dados e seus respectivos dados também devem acompanhar essa evolução. Durante esse ciclo, pode existir um estado misto onde as entidades não adaptadas e adaptadas estão funcionando ao mesmo tempo, exigindo a interação simultânea com as versões antiga e nova do esquema de dados da aplicação. Ao final do processo de adaptação, somente a nova versão do esquema de dados do SaS deve ser mantida, pois todas as suas entidades de software já sofreram as adaptações necessárias. Nesse sentido, a biblioteca DynaSchema deve permitir que o SaS troque definitivamente para a nova versão do esquema de dados e apague a antiga, assim liberando espaço de armazenamento no banco de dados. Esse processo deve ser feito de maneira assíncrona para evitar que o SaS fique esperando a eliminação de uma versão antiga de esquema com muitos dados. Além disso, a biblioteca DynaSchema deve notificar quando a troca da versão do esquema de dados da aplicação for concluída, permitindo que o SaS finalize seu ciclo de autoadaptação.

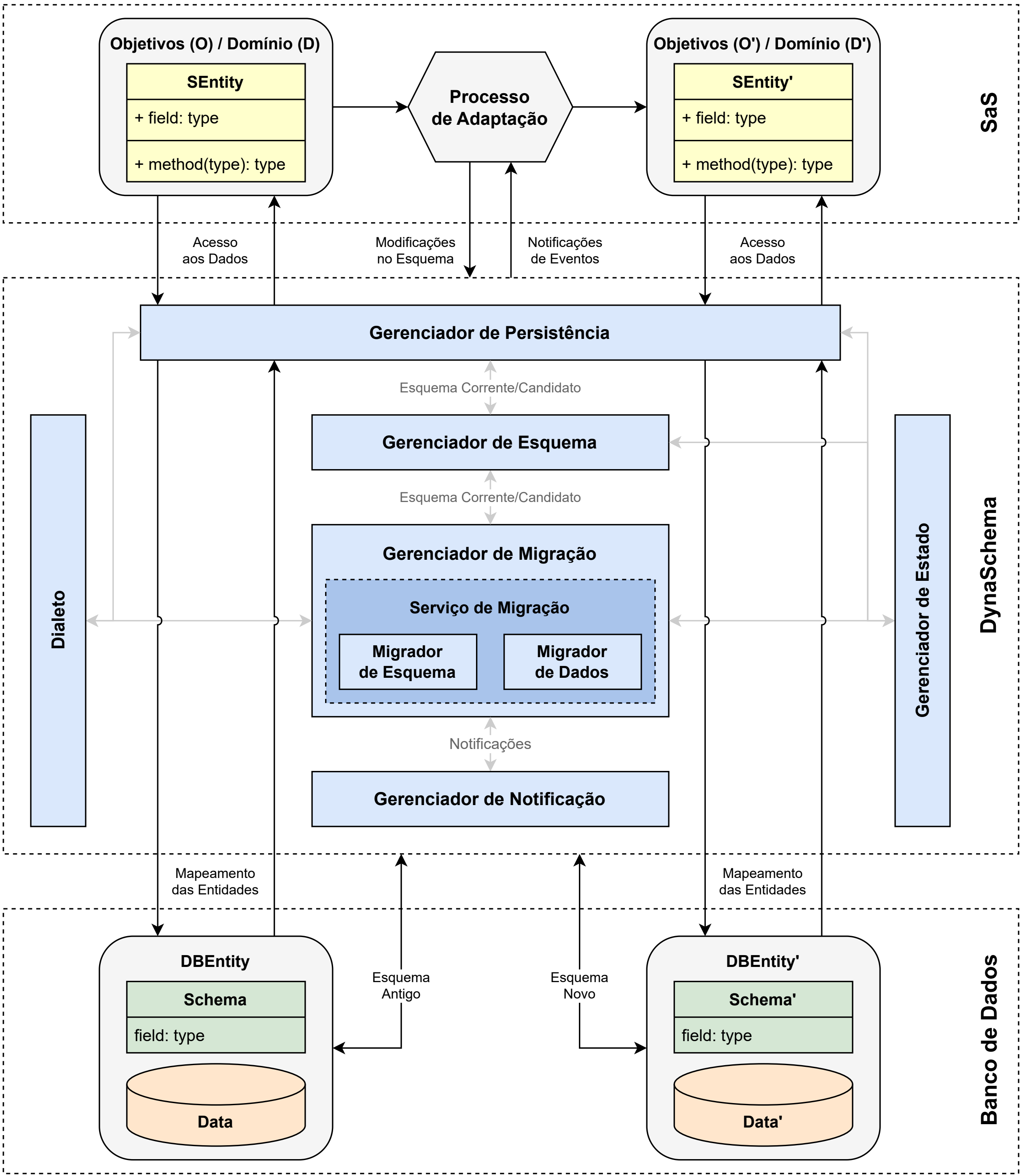
4.1.4 Arquitetura Interna

De acordo com os requisitos apresentados nas Seções 4.1.2 e 4.1.3, foi adotado um estilo de arquitetura modular (Seção 2.1) para estruturar a biblioteca DynaSchema. Esse estilo arquitetural permite lidar com a complexidade de implementação da biblioteca, pois cada módulo é responsável por conter um conjunto bem definido e coeso de funcionalidades, facilitando a compreensão e manutenibilidade do código fonte. A arquitetura da biblioteca DynaSchema pode ser visualizada na Figura 24, sendo composta por seis módulos principais, a saber:

Módulo Gerenciador de Estado. Este módulo é responsável por armazenar os metadados utilizados por outros módulos da biblioteca DynaSchema. Tais metadados são necessários para refletir o estado de cada módulo e coordenar suas funcionalidades individuais. Como exemplo, esses metadados podem conter informações sobre o esquema de dados atual, o *status* da migração de esquema em andamento, o *status* da migração de dados em andamento, o histórico de migrações de esquema, as informações de conexão com o banco de dados, o dialeto SQL que deve ser utilizado, entre outras informações. Para que isso seja possível, este módulo deve fornecer um repositório de dados local (por exemplo, um banco de dados, arquivo XML, binário ou JSON), permitindo que os metadados sejam atualizados durante o processo de evolução do esquema de dados do SaS. Além disso, o módulo Gerenciador de Estado deve permitir que os outros módulos da biblioteca DynaSchema possam acessar seus metadados sem acesso conflitante.

Módulo Gerenciador de Notificação. Este módulo é responsável por notificar o SaS em relação às ações que estão sendo executadas pela biblioteca DynaSchema. O crescimento do volume de dados armazenados no banco pode provocar a demora do processo de migração do esquema de dados. Por esse motivo, essa migração deve ser realizada de maneira

Figura 24 – Arquitetura da biblioteca DynaSchema



Fonte: Elaborada pelo autor.

assíncrona para evitar que a execução do SaS seja comprometida e/ou interrompida. Assim, o SaS deve ser notificado quando essa tarefa começa e termina, evitando que o mesmo evolua para um estado de execução defeituosa. A notificação de término de migração é importante na leitura dos registros contidos no banco de dados, pois informações parciais podem ser recuperadas caso a migração do esquema ainda esteja em andamento. Além disso, o SaS também pode ser notificado em relação ao processo de remoção da versão antiga do esquema de dados, permitindo que um ciclo de autoadaptação seja concluído.

Módulo de Dialeto. Este módulo é responsável por definir as instruções SQL executadas pela biblioteca DynaSchema. O projeto de um SaS é livre para escolher um SGBD relacional que melhor atenda às necessidades da sua aplicação. Nesse contexto, vale destacar que a sintaxe de uma mesma instrução SQL pode ser declarada de maneira diferente entre SGBDs distintos. Diante desse cenário, este módulo deve facilitar a extensão da biblioteca DynaSchema, permitindo a implementação de um novo dialeto SQL para lidar com cada SGBD suportado. Além disso, o SaS também deve configurar a biblioteca DynaSchema para utilizar um dialeto específico que atende às suas necessidades. Tal configuração deve ser feita nos metadados armazenados pelo módulo Gerenciador de Estado.

Módulo Gerenciador de Esquema. Este módulo é responsável por gerenciar as versões do esquema de dados (ou seja, corrente e candidato) utilizadas pelo SaS. Assim, este módulo implementa uma estratégia que é capaz de lidar simultaneamente com ambos os esquemas, evitando um mapeamento incorreto entre as classes das entidades de software e as tabelas do banco de dados. Para isso, as ações de modificação do esquema de dados devem ser realizadas de maneira independente em um esquema candidato, evitando que esquema de dados corrente (que está sendo utilizado pelo SaS) seja perturbado. Essas duas versões de esquema devem ser armazenadas e acessadas através do módulo Gerenciador de Estado.

Módulo Gerenciador de Persistência. Este módulo é responsável por gerenciar a persistência de dados do SaS em um banco de dados relacional. Tal mecanismo de persistência se baseia na abordagem de mapeamento objeto-relacional com a utilização de anotações para marcar os elementos de uma classe. Para estabelecer a estratégia de mapeamento, este módulo deve consultar o módulo Gerenciador de Esquema para escolher o plano contido no esquema de dados corrente ou no esquema de dados candidato (estado misto). Além disso, o módulo Gerenciador de Persistência utiliza a reflexão computacional para extrair e inserir as informações em tempo de execução, permitindo a persistência dos objetos no banco de dados relacional e vice-versa. Vale ressaltar que a estratégia de mapeamento dinâmico utilizada neste módulo é um diferencial relevante em relação às soluções ORM disponíveis na literatura. O mapeamento entre classes e tabelas geralmente é configurado estaticamente durante a fase de desenvolvimento do software, porém este módulo permite que o mapeamento ocorra de maneira dinâmica na fase de execução de acordo com uma das versões do esquema apontada pelo módulo Gerenciador de Esquema. Além disso, os metadados fornecidos por esse último módulo também podem ser utilizados para armazenar as informações de conexão com o banco de dados.

Módulo Gerenciador de Migração. Este módulo é responsável pela da evolução do esquema de dados do SaS. Vale ressaltar que esse processo envolve a alteração simultânea dos esquemas do SaS (orientado a objetos) e do banco de dados (relacional). Nesse sentido, o SaS é responsável por adaptar a estrutura das suas entidades de software enquanto o módulo Gerenciador de Migração realiza as alterações no esquema de banco de dados. Para isso, este módulo coordena o processo de migração do esquema através de três etapas, a saber: (i) *a migração do esquema de dados*, cuja tarefa é delegada ao módulo Migrador de Esquema; (ii) *a migração dos dados*, cuja tarefa é delegada ao módulo Migrador de Dados; e (iii) *a alteração do contexto para o esquema de dados mais atual*, que requer coordenação entre o módulo Gerenciador de Esquema para alterar o esquema atual e o módulo Gerenciador de Estado para salvar as alterações. Além disso, esse último módulo pode ser utilizado para armazenar algumas informações sobre o processo de migração em andamento, a saber: (1) se a migração do esquema foi concluída com sucesso ou com erros; (2) se a migração dos dados foi concluída com sucesso ou com erros; e (3) se as migrações de esquema e dados foram concluídas com sucesso para que o contexto do SaS possa mudar para a versão mais atual do esquema de dados.

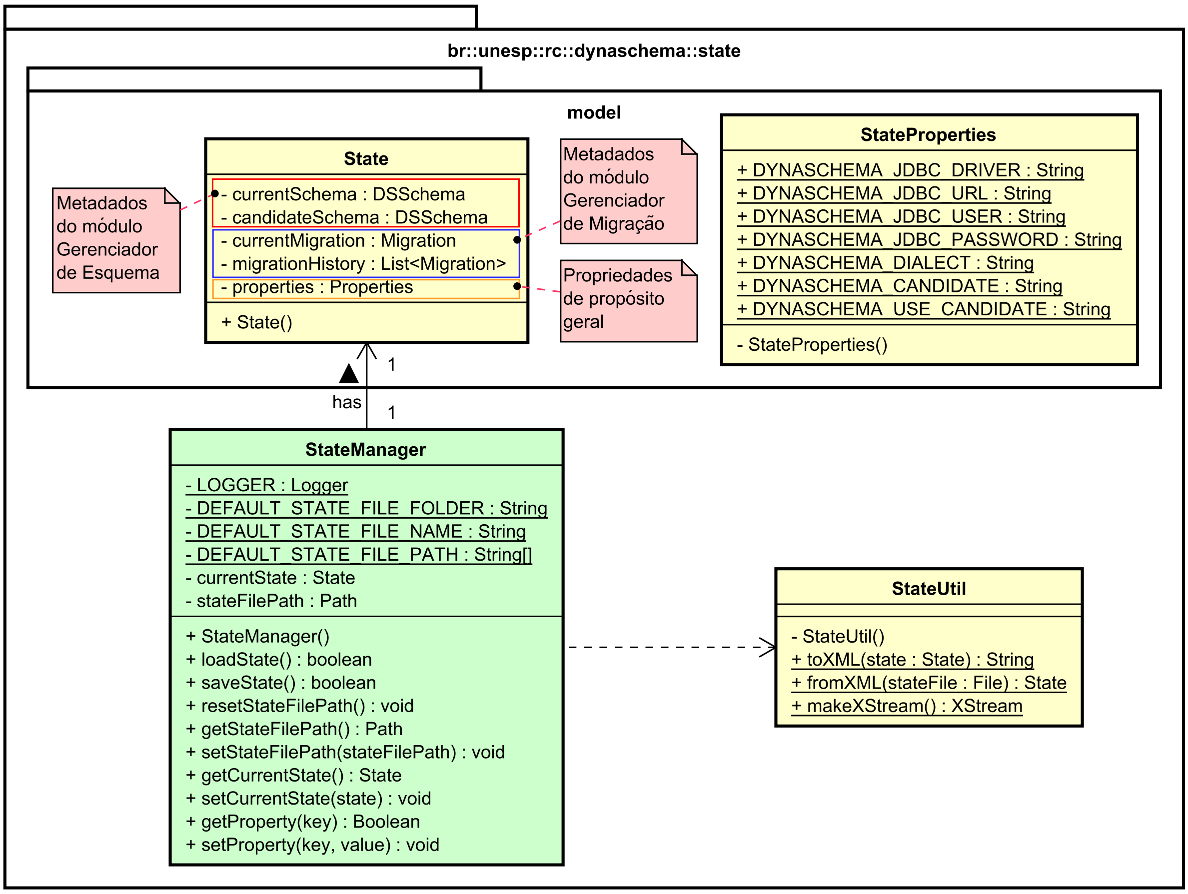
4.2 *Design* dos Módulos e da Interface da Biblioteca

Uma visão geral da arquitetura da biblioteca DynaSchema foi apresentada na Seção 4.1.4, onde foram discutidas as funções dos seus módulos internos, além de ilustrar suas interações através da Figura 24. Nessa seção são apresentados os detalhes de *design* dessa biblioteca, cujos seis módulos internos são apresentados nas Seções 4.2.1 a 4.2.6. Além disso, a interface de interação com seus usuários é detalhada na Seção 4.2.7.

4.2.1 Módulo Gerenciador de Estado

O módulo Gerenciador de Estado tem a função de controlar a maneira como os metadados são armazenados e acessados pelos outros módulos da biblioteca DynaSchema. Para isso, foi utilizada a biblioteca XStream[6] para serializar as informações do objeto de metadados em um arquivo XML e desserializar esse arquivo posteriormente em um objeto de metadados. A linguagem XML foi escolhida por ser um formato de texto legível pelos seres humanos que permite a estruturação hierárquica das informações dos metadados através de *tags*. A estrutura interna do módulo Gerenciador de Estado pode ser visualizada na Figura 25.

Figura 25 – Diagrama de classes do módulo Gerenciador de Estado



Fonte: Elaborada pelo autor.

A classe `StateManager` constitui a interface de utilização módulo Gerenciador de Estado e contém dois atributos principais, a saber: (i) `currentState`, que armazena o estado atual dos metadados (ou seja, o objeto de metadados); e (ii) `stateFilePath`, que aponta para o local de armazenamento do arquivo XML. Além disso, também é possível observar outros atributos acessórios, contendo o utilitário para a geração de *logs*[7] (`LOGGER`) e as constantes para definir a localização do diretório (`DEFAULT_STATE_FILE_FOLDER`), o nome

[6] O projeto XStream pode ser encontrado em: <https://x-stream.github.io>
[7] Os *logs* são gerados através da biblioteca Apache Log4j, disponível em: <https://logging.apache.org/log4j>.

(`DEFAULT_STATE_FILE_NAME`) e o caminho completo (`DEFAULT_STATE_FILE_PATH`) do arquivo XML de metadados.

Além dos atributos, a classe `StateManager` contém alguns métodos para recuperar as informações dos metadados do arquivo XML (`loadState()`), salvar o estado do objeto de metadados no arquivo XML (`saveState()`), restaurar o apontamento para o caminho padrão do arquivo de metadados (`resetStateFilePath()`), alterar o caminho do arquivo de metadados (`setStateFilePath()`), recuperar o objeto de metadados (`getCurrentState()`), modificar o conteúdo das propriedades de propósito geral (`setProperty()`), entre outras funcionalidades. Vale salientar que a classe `StateUtil` contém os métodos utilitários para configurar e interagir com a biblioteca XStream, sendo utilizados posteriormente pela classe `StateManager` para serializar e desserializar os dados do objeto de metadados.

A classe `State` modela a estrutura do objeto de metadados, cujo conteúdo armazena as informações utilizadas por outros módulos da biblioteca DynaSchema. Conforme ilustrado na Figura 25, essa classe armazena os dados dos modulos Gerenciador de Esquema (ou seja, os atributos `currentSchema` e `candidateSchema` destacados na caixa de linha vermelha) e Gerenciador de Migração (ou seja, os atributos `currentMigration` e `migrationHistory` contidos na caixa de linha azul), além das propriedades de propósito geral disponibilizadas pelo módulo Gerenciador de Estado (ou seja, o atributo `properties` marcado na caixa de linha laranja). Por razões de espaço, os métodos de acesso aos atributos da classe `State` (ou seja, os métodos *getters* e *setters*) foram omitidos no diagrama de classes ilustrado na Figura 25.

A classe `StateProperties` armazena o nome das propriedades de propósito geral na forma de constantes, podendo ser utilizadas posteriormente para definir ou acessar as configurações de conexão com o banco de dados (constantes `DYNASCHEMA_JDBC_DRIVER`, `DYNASCHEMA_JDBC_URL`, `DYNASCHEMA_JDBC_USER` e `DYNASCHEMA_JDBC_PASSWORD`), informar a estratégia de dialeto SQL que deve ser utilizada pela biblioteca DynaSchema (`DYNASCHEMA_DIALECT`) e indicar se deve ser utilizado o mapeamento entre classes e tabelas condidos no esquema candidato ao invés do esquema corrente (`DYNASCHEMA_USE_CANDIDATE`). Para isso, devem ser utilizados os métodos `getProperty()` e `setProperty()` da classe `StateManager` para recuperar e alterar o valor dessas propriedades, respectivamente. Um exemplo prático da utilização das propriedades de propósito geral pode ser visto na Figura 29 da Seção 4.2.3.

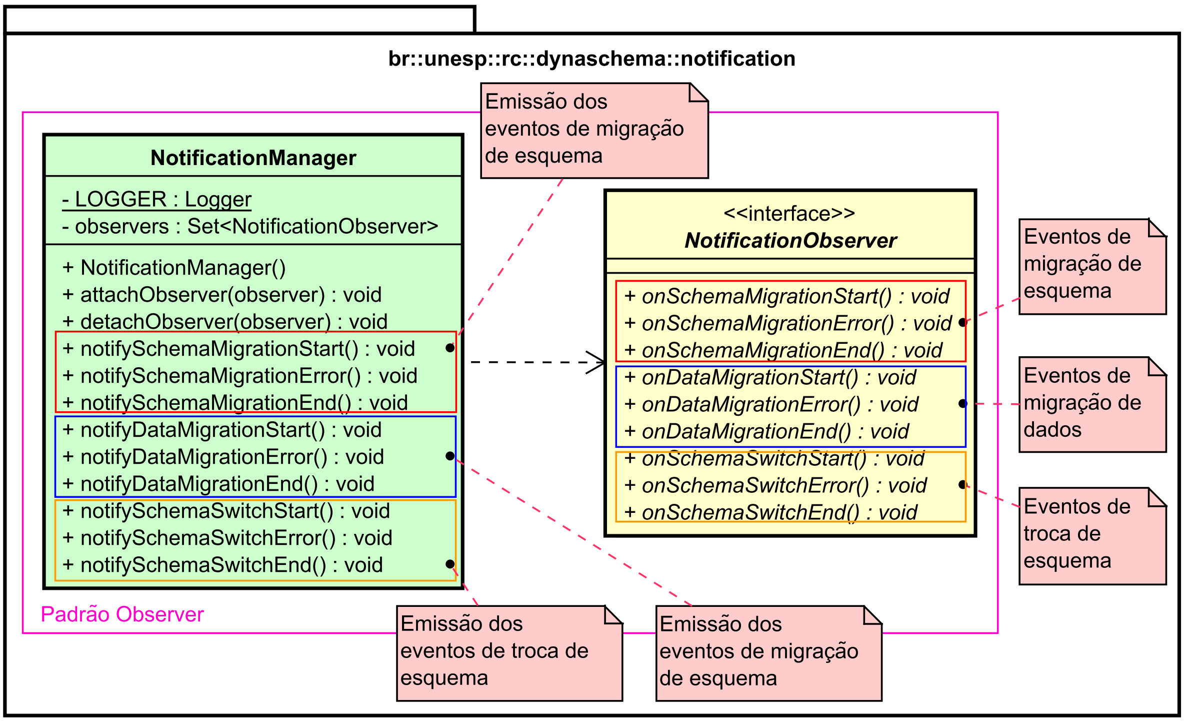
Com relação ao funcionamento do módulo Gerenciador de Estado, podem ser destacas três características importantes. A **primeira** está relacionada com a volatilidade de armazenamento do estado do objeto de metadados, assim o método `saveState()` da classe `StateManager` deve ser invocado para que suas informações sejam salvas no arquivo XML de metadados. Esse comportamento é interessante para lidar com os casos de falhas catastróficas do SaS, evitando a inserção de metadados parciais no arquivo XML. A **segunda** estabelece um diferencial com relação a outras implementações da JPA, onde é possível especificar o diretório de armazenamento e o nome do arquivo de metadados através do método `setStateFilePath()` da classe `StateManager`. Essa característica permite que o SaS possa definir um repositório central de arquivos (como um diretório compartilhado) ou utilizar algum padrão de nomenclatura para os arquivos de metadados, por exemplo. Finalmente, a **terceira** característica estabelece que o módulo Gerenciador de Estado não manipula as informações dos metadados, sendo somente responsável pelo gerenciamento do seu armazenamento. Nesse sentido, os outros módulos da biblioteca DynaSchema devem acessar e alterar o conteúdo dos seus respectivos metadados através do método `getCurrentState()` da classe `StateManager`.

4.2.2 Módulo Gerenciador de Notificação

O módulo Gerenciador de Notificação implementa o mecanismo de notificações da biblioteca DynaSchema, permitindo que seus usuários possam reagir aos eventos lançados durante o processo de evolução do esquema de dados da aplicação. Vale lembrar que a evolução do esquema deve ser executada de maneira assíncrona para evitar que a biblioteca interfira no funcionamento do SaS, assim este módulo mantém o SaS atualizado em relação ao estado de execução das atividades para a evolução do seu esquema de dados. Em relação aos requisitos estabelecidos na Seção 4.1.3, este módulo complementa o funcionamento das funcionalidades **F2** e **F6** oferecidas pela biblioteca DynaSchema. Na Figura 26 são ilustradas as classes que compõem o módulo Gerenciador de Notificação.

A interface `NotificationObserver` modela a maneira de "escutar" os eventos da biblioteca DynaSchema. Nesse sentido, vale destacar que um SaS pode criar uma ou mais extensões dessa interface para implementar o modo de reagir a esses eventos. De acordo com o diagrama de classes ilustrado na Figura 26, essa interface apresenta alguns métodos abstratos que estão relacionados com três tipos de eventos, a saber: (i) na caixa de linha vermelha estão contidos os métodos relacionados com os eventos de migração de esquema (métodos `onSchemaMigrationStart()`, `onSchemaMigrationError()` e `onSchemaMigrationEnd()`); (ii) na caixa de linha azul

Figura 26 – Diagrama de classes do módulo Gerenciador de Notificação



Fonte: Elaborada pelo autor.

estão contidos os métodos relacionados com os eventos de migração de dados (métodos `onDataMigrationStart()`, `onDataMigrationError()` e `onDataMigrationEnd()`); e (iii) na caixa de linha laranja estão contidos os métodos relacionados com os eventos de troca da versão do esquema de dados (métodos `onSchemaSwitchStart()`, `onSchemaSwitchError()` e `onSchemaSwitchEnd()`). Vale salientar que foi adotado um padrão de sufixos na nomenclatura desses métodos, onde "`Start`" indica que a ação foi iniciada, "`Error`" sinaliza a ocorrência de algum erro que interrompe a execução da ação, e "`End`" aponta que a ação foi concluída.

O "ponto de entrada" do módulo Gerenciador de Notificação é modelado pela classe `NotificationManager`, cujo atributo `observers` armazena a lista de observadores que devem ser notificados sobre aos eventos emitidos pela biblioteca DynaSchema. Tais observadores podem se registrar nessa lista através do método `attachObserver()`, ou podem utilizar o método `detachObserver()` para abandonar a lista e deixarem de receber as notificações enviadas pela biblioteca. Além disso, a classe `NotificationManager` contém outros métodos com o prefixo "`notify`" para que a biblioteca DynaSchema possa lançar eventos relacionados com três tipos de notificação, a saber: (i) na caixa de linha vermelha estão contidos os métodos para emitir os eventos relacionados com a migração de esquema (métodos `notifySchemaMigrationSt-`

`art()`, `notifySchemaMigrationError()` e `notifySchemaMigrationEnd()`); (ii) na caixa de linha azul estão contidos os métodos para emitir os eventos relacionados com a migração de dados (métodos `notifyDataMigrationStart()`, `notifyDataMigrationError()` e `notifyDataMigrationEnd()`); e (iii) na caixa de linha laranja estão contidos os métodos para emitir os eventos relacionados com a troca da versão do esquema de dados (métodos `notifySchemaSwitchStart()`, `notifySchemaSwitchError()` e `notifySchemaSwitchEnd()`). Vale salientar que a interface `NotificationObserver` e a classe `NotificationManager` estão destacadas com uma caixa de linha rosa na Figura 26, indicando a relação dessas estruturas com o padrão *Observer*[8].

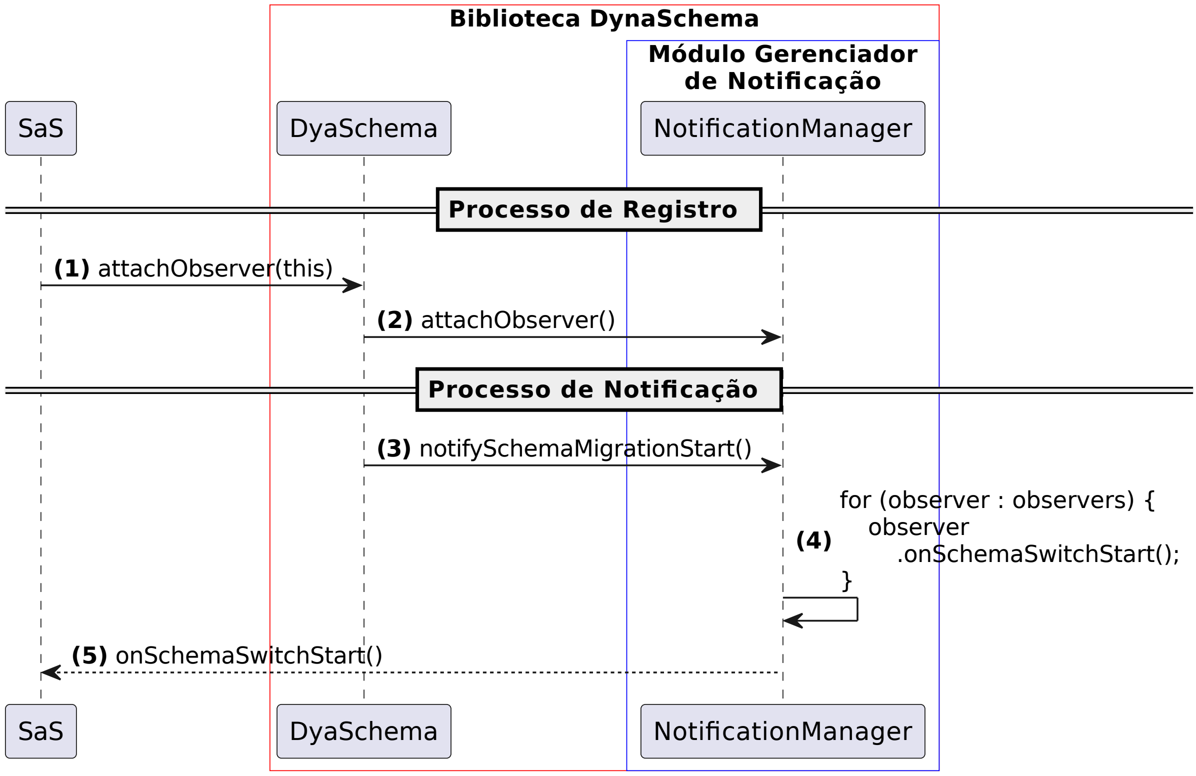
Na Figura 27 é ilustrado um exemplo do funcionamento do mecanismo de notificação da biblioteca DynaSchema. Inicialmente, o SaS se registra na biblioteca para ser notificado sobre seus eventos internos **(Passo 1)**. Em seguida, a biblioteca DynaSchema registra esse SaS na lista de observadores do módulo de Gerenciador de Notificação **(Passo 2)**. A execução desses passos conclui o processo de registro dos observadores, estabelecendo a lista de usuários que devem ser notificados pela biblioteca. Durante um ciclo de autoadaptação, o SaS pode solicitar a criação de uma nova versão de esquema de dados. Por sua vez, a biblioteca DynaSchema recebe essa solicitação e emite um evento quando o processo de migração do esquema for iniciado **(Passo 3)**, resultando na notificação de todos os observadores registrados no módulo de Gerenciador de Notificação **(Passo 4)**. Por fim, o SaS recebe uma mensagem indicando que a solicitação de uma nova versão de esquema está sendo processada pela biblioteca **(Passo 5)**. Vale lembrar que o passo a passo ilustrado na Figura 27 pode ser seguido para emitir os outros tipos de eventos que ocorrem dentro dessa biblioteca (por exemplo, a conclusão da migração de dados).

4.2.3 Módulo de Dialeto

O módulo de Dialeto armazena e gerencia as instruções SQL utilizadas pela biblioteca DynaSchema. Nesse sentido, o SaS deve escolher uma das estratégias de dialeto SQL disponibilizadas por este módulo, permitindo que a biblioteca possa utilizar o conjunto adequado de instruções para manipular um determinado produto de banco de dados. Tais instruções são utilizadas por dois outros módulos da biblioteca DynaSchema, a saber: (i) *o Gerenciador de Persistência*, que manipula os dados armazenados no baco de dados; e (ii) *o Gerenciador de Migração*, que gerencias as mudanças no esquema do banco de dados. Nesse sentido, o módulo

[8] Mais detalhes sobre o padrão *Observer* podem ser consultados na Seção 2.1.

Figura 27 – Funcionamento de mecanismo de notificação

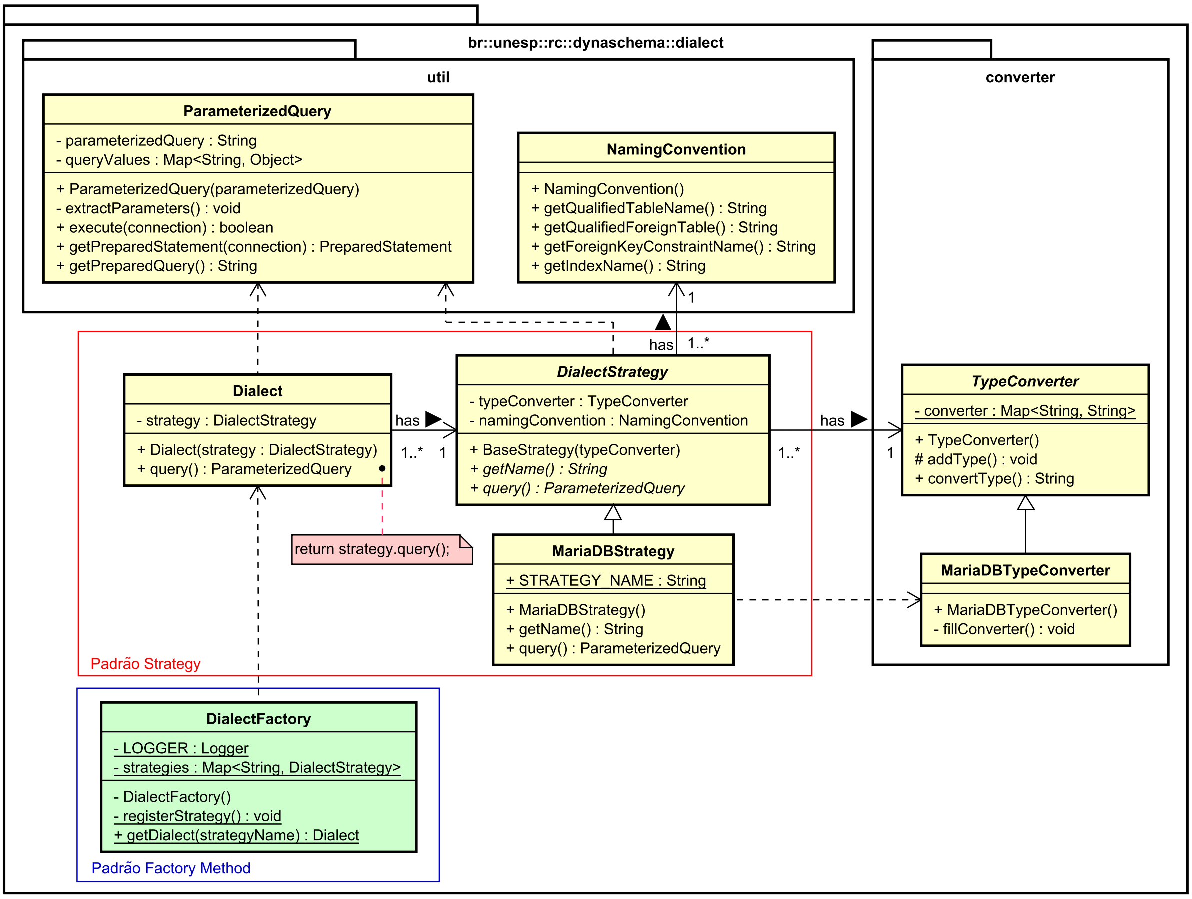


Fonte: Elaborada pelo autor.

de Dialeto complementa o funcionamento dos dois componentes funcionais da biblioteca DynaSchema (Seção 4.1.2), além implementar o recurso exigido pela funcionalidade **F5** que foi apresentada na Seção 4.1.3. Na Figura 28 é ilustrado o diagrama de classes deste módulo.

A interface funcional de um dialeto SQL é modelada pela classe `Dialect`, cujo conjunto de instruções SQL é representado de maneira resumida pelo método `query()`. Essa classe não disponibiliza diretamente as instruções SQL, pois as chamadas são repassadas para o objeto apontado pelo atributo `strategy` (ou seja, `strategy.query()`). Esse objeto pertence à classe `DialectStrategy`, que modela a interface de uma estratégia de dialeto SQL. Dessa maneira, as novas estratégias concretas podem especializar a classe `DialectStrategy` e implementar de fato os métodos que armazenam as instruções SQL (representados resumidamente pelo método `query()`). Além disso, cada estratégia concreta de dialeto SQL deve conter um nome exclusivo que é acessado pelo método `getName()` da classe `DialectStrategy`. Um exemplo de estratégia de dialeto SQL é a classe `MariaDBStrategy`, que implementa as instruções SQL para manipular o produto de banco de dados MariaDB. Conforme ilustrado na Figura 28, as classes `Dialect`, `DialectStrategy` e `MariaDBStrategy`, contidas em uma caixa de linha

Figura 28 – Diagrama de classes do módulo de Dialeto



Fonte: Elaborada pelo autor.

vermelha, estão relacionadas com padrão arquitetural *Strategy*[9], cujo esquema de organização facilita a adição de outras estratégias de dialeto SQL na biblioteca DynaSchema através da criação de classes concretas que estendem a classe `DialectStrategy`.

A classe `DialectFactory` constitui um utilitário para a montagem de um objeto de dialeto SQL (ou seja, uma instância de `Dialect`). Internamente, essa classe contém o atributo `strategies`, que armazena um mapa com o nome de todas as estratégias concretas de dialeto SQL implementadas na biblioteca DynaSchema. Quando necessário, o método fábrica `getDialect()` pode ser invocado para obter a instância de um objeto de dialeto SQL. Para isso, o nome da estratégia concreta deve ser passado pelo parâmetro `strategyName`, permitindo que esse método consulte seu mapa interno, procure pela instância da estratégia e injete sua referência no objeto de dialeto SQL que será instanciado. De acordo com o diagrama de classes exibido na Figura 28, a classe `DialectFactory` e seu método `getDialect()`, englobados

[9] Mais detalhes sobre o padrão *Strategy* podem ser consultados na Seção 2.1.

por uma caixa de linha azul, estão relacionados com o padrão arquitetural *Factory Method*[10], cujo mecanismo de instanciação permite a configuração dinâmica do processo de criação dos objetos de dialeto SQL. Além disso, também é possível notar a presença de outras quatro classes acessórias nesse diagrama, a saber:

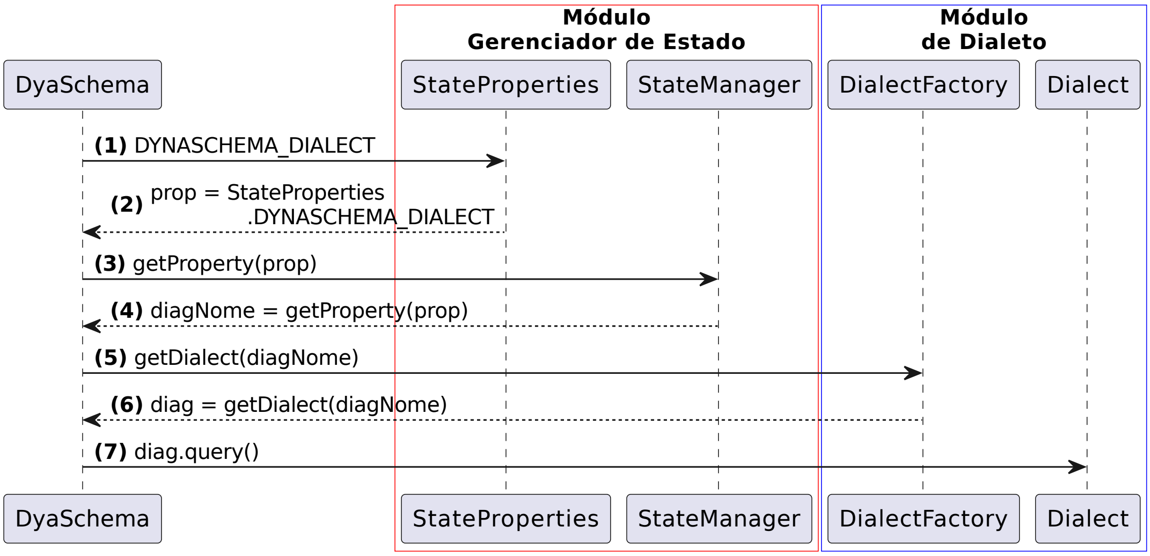
- `ParameterizedQuery`: esta classe é utilizada para formatar uma instrução SQL, permitindo que seus parâmetros sejam preenchidos na ordem em que foram declarados. Essa característica é especialmente importante para a funcionalidade de persistência de dados, pois a ordem de declaração dos atributos na classe de uma entidade de software pode ser diferente da ordem dos parâmetros da instrução SQL;
- `NamingConvention`: esta classe é utilizada para formatar o nome das tabelas, restrições e índices de acordo com a versão do esquema de dados que está sendo utilizada pelo SaS;
- `TypeConverter`: esta classe modela a interface de um conversor de tipos, mapeando os tipos dos atributos das entidades de software (por exemplo, escritas na linguagem Java) para os tipos correspondente no banco de dados; e
- `MariaDBTypeConverter`: esta classe implementa um conversor de tipos para a estratégia de dialeto SQL do MariaDB[11]. Nesse sentido, cada nova estratégia de dialeto SQL suportada pela biblioteca DynaSchema deve implementar seu próprio conversor de tipos.

De acordo com o diagrama de sequência ilustrado na Figura 29, os metadados contidos no módulo Gerenciador de Estado podem ser utilizados para configurar o processo de instanciação do objeto de dialeto SQL. Inicialmente, a biblioteca DynaSchema recupera o nome da propriedade que configura a estratégia de dialeto SQL **(Passo 1)**, sendo armazenada em seguida na variável `prop` **(Passo 2)**. O valor da propriedade correspondente é obtido através do método `getProperty()` da classe `StateManager` **(Passo 3)**, cujo retorno é armazenado na variável `diagNome` **(Passo 4)**. Em seguida, a instância do objeto de dialeto SQL pode ser obtida através do método fábrica `getDialect()` **(Passo 5)** fornecido pela classe `DialectFactory`. A variável `diagNome`, que contém o nome da estratégia concreta de dialeto SQL, é passada como parâmetro para esse método, permitindo que seja retornado um objeto de dialeto SQL que está configurado para lidar com um SGBD específico. A instância desse objeto é referenciada pela variável `diag`

[10] Mais detalhes sobre o padrão *Factory Method* podem ser consultados na Seção 2.1.
[11] No estudo de caso conduzido no Capítulo 5 foi utilizada a versão 10.6 do MariaDB em um *container* Docker.

(Passo 6), estabelecendo um meio para que a biblioteca DynaSchema execute as instruções SQL (resumidas pelo método `query()`) de maneira adequada **(Passo 7)**. Esse passo a passo pode ser seguido tanto pelo módulo Gerenciador de Persistência quanto pelo módulo Gerenciador de Migração, possibilitando a configuração dinâmica do dialeto SQL que será utilizado pela biblioteca DynaSchema em tempo de execução.

Figura 29 – Processo de instanciação de um objeto de dialeto SQL



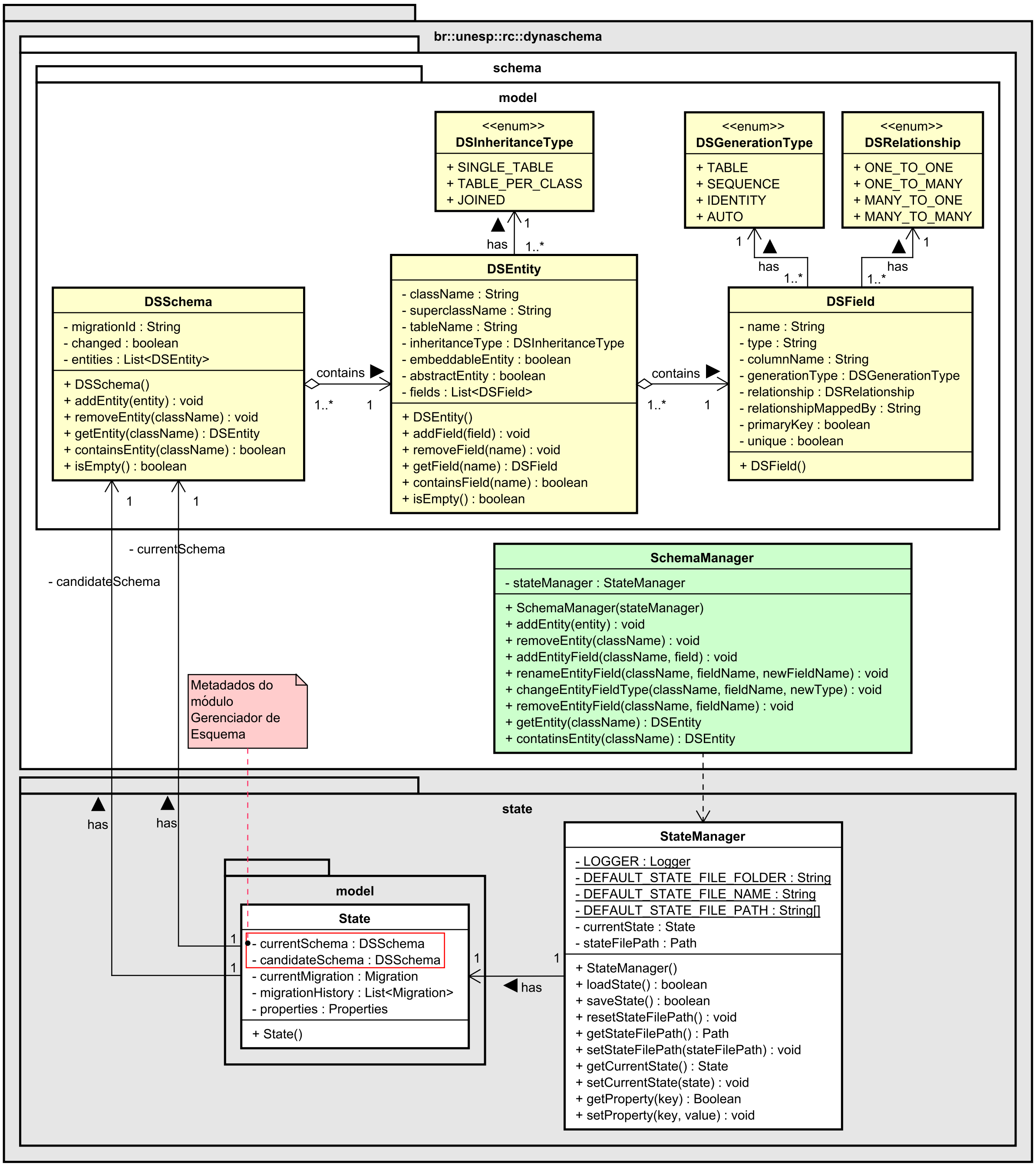
Fonte: Elaborada pelo autor.

4.2.4 Módulo Gerenciador de Esquema

O módulo Gerenciador de Esquema controla o mapeamento entre as classes das entidades de software do SaS que precisam ser persistidas e as tabelas contidas no banco de dados. Tal mapeamento é intensamente consultado por outros dois módulos da biblioteca DynaSchema, a saber: (i) *o Gerenciador de Persistência*, que realiza o processo de "montagem" e "desmontagem" das instâncias das entidades de software de acordo com o esquema de dados utilizado pelo SaS; e (ii) *o Gerenciador de Migração*, que processa a criação de uma nova versão de esquema de dados com base nas modificações solicitadas pelo SaS. Nesse sentido, este módulo apoia o funcionamento das interfaces de persistência de dados e evolução do esquema de dados abordadas na Seção 4.1.2, além de estar relacionado com as funcionalidades **F1** e **F3** (Seção 4.1.3). O digrama de classes do módulo Gerenciador de Esquema é apresentado na Figura 30.

No pacote `schema::model` estão contidas as classes para criar um modelo conceitual do esquema de dados de um SaS. Nesse sentido, esse conjunto de classes é utilizado pela biblioteca

Figura 30 – Diagrama de classes do módulo Gerenciador de Esquema



Fonte: Elaborada pelo autor.

DynaSchema para representar as informações estruturais das entidades de software do SaS. De acordo com a descrição da interface de persistência de dados abordada na Seção 4.1.2, a biblioteca DynaSchema utiliza um mecanismo de persistência baseado na abordagem de mapeamento objeto-relacional, onde as classes e seus respectivos atributos devem ser marcados com anotações de persistência de dados. Dessa maneira, as classes das entidades de software que estão relacionadas com a persistência de dados armazenam dois tipos principais de informação.

O primeiro tipo está relacionado com a estrutura física da classe que deve ser representada em uma ou mais tabelas no baco de dados. Já o segundo tipo está relacionado com as anotações de persistência, que podem estabelecer o mapeamento entre uma entidade de software e uma tabela específica do banco de dados, marcar um atributo de valor único ou de incremento automático, determinar o tipo do relacionamento de cardinalidade entre as entidades de software do SaS, entre outras funcionalidades. Diante desse cenário, as classes do pacote `schema::model` foram projetadas para armazenar esses dois tipos de informações das entidade de software do SaS, possuindo um papel semelhante ao metamodelo da arquitetura de referência RA4SaS, cuja estrutura foi ilustrada na Figura 7 da Seção 2.3.

Na Figura 30 pode-se notar que os nomes das classes do pacote `schema::model` possuem o prefixo "`DS`" que significa DynaSchema. Tal convenção de nomenclatura foi utilizada para evitar o conflito do espaço de nomes com a linguagem Java e a interface da JPA, cujos conceitos estão alinhados aos da biblioteca DynaSchema e tendem a possuir os mesmos nomes. A adoção dessa convenção facilita a manutenção do código fonte da biblioteca, evitando a exigência da utilização do caminho completo para referenciar as classes e/ou interfaces com o mesmo nome. Vale salientar que os métodos de acesso aos atributos dessas classes foram omitidos na Figura 30 por razões de espaço. Uma descrição de cada uma das classes contidas no pacote `schema::model` do módulo Gerenciador de Esquema é apresentada a seguir:

- **`DSSchema`:** esta classe modela a estrutura geral de um esquema de dados utilizado pelo SaS. Isso significa que a classe `DSSchema` armazena o estado estrutural das entidades de software do SaS em um instante de tempo. Nesse sentido, tal classe pode ser utilizada para representar tanto o esquema corrente, que está sendo utilizado atualmente pelo SaS, quanto o esquema candidato, que contém as mudanças estruturais exigidas pelo ciclo de autoadaptação do SaS e estabelece a nova versão do esquema de dados;

- **`DSEntity`:** esta classe carrega as informações de uma entidade de software que precisa ser persistida no banco de dados. Nesse sentido, a classe `DSEntity` armazena dois tipos principais de conhecimentos, a saber: (i) *a estrutura da classe da entidade de software* e (ii) *as informações das anotações de persistência contidas nessa classe*. Com relação ao primeiro tipo, podem ser recolhidas as informações sobre o nome da classe, o nome da superclasse, se a classe é abstrata e não pode ser instanciada, entre outras informações. Por outro lado, os conhecimentos do segundo tipo estão relacionados com a estratégia de mapeamento de herança, se a classe deve ser considerada como um tipo "embutido" que não deve possuir uma tabela própria, entre outras informações;

- **DSInheritanceType:** esta enumeração armazena a lista das estratégias de mapeamento para a herança de classes que foram especificadas pela JPA. Assim, pode ser definido se todas as classes da herança serão mapeadas em uma única tabela (`SINGLE_TABLE`), em uma tabela para cada classe (`TABLE_PER_CLASS`) ou em uma hierarquia de tabelas que agrupam os campos comuns por classe (`JOINED`);

- **DSField:** esta classe representa os campos/atributos das entidades de software. Assim como a classe `DSEntity`, a classe `DSField` também armazena dois tipos de conhecimentos. Com relação à estrutura da classe da entidade de software, podem ser coletadas as informações sobre o nome dos campos/atributos e seu tipo, por exemplo. Em contrapartida, com relação às anotações de persistência, podem ser rastreadas as informações sobre o nome da coluna na tabela do banco de dados, seu tipo de relacionamento de cardinalidade com outra entidade de software, se seu valor deve ser gerado automaticamente, se possui um valor único ou é uma chave primária, entre outros dados;

- **DSGenerationType:** esta enumeração armazena a lista das estratégias de geração dos campos de uma tabela que foram especificadas pela JPA. Dessa maneira, o valor de um campo pode ser gerado com base em um contador armazenado em uma tabela (`TABLE`) ou em uma sequência (`SEQUENCE`), incrementado unitariamente a cada novo registro (`IDENTITY`) ou com a escolha automática da estratégia de geração (`AUTO`); e

- **DSRelationship:** esta enumeração armazena os tipos de relacionamentos de cardinalidade entre as entidades e os tipos dos seus atributos que foram especificadas pela JPA. Nesse sentido, pode ser definido se a entidade atual e o tipo do seu atributo, que também deve ser outra entidade, estabelecem uma relação do gênero "uma para um" (`ONE_TO_ONE`), "um para muitos" (`ONE_TO_MANY`), "muitos para um" (`MANY_TO_ONE`) ou "muitos para muitos" (`MANY_TO_MANY`). No contexto da biblioteca DynaSchema, essa configuração é importante para estabelecer as chaves estrangeiras entre as classes do banco de dados.

A classe `SchemaManager` do pacote `schema` modela a interface de interação com o módulo Gerenciador de Esquema, sendo responsável pelo gerenciamento de duas versões do esquema de dados do SaS. A **primeira versão** está relacionada com o esquema de dados corrente, que contém o mapeamento entre classes e tabelas que estão sendo utilizadas atualmente pelo SaS. Tal esquema é consultado pelo módulo Gerenciador de Persistência para estabelecer a maneira como os dados do SaS serão recuperados e armazenados em um banco de dados relacional. A **segunda versão** está relacionada com o esquema candidato, que acumula as alterações estruturais do esquema corrente solicitadas pelo SaS durante um ciclo de autoadaptação. Esse esquema será utilizado posteriormente pelo módulo Gerenciador de Migração para modificar o banco de dados segundo a nova versão estrutural do esquema do SaS. Além disso, caso a migração do esquema (ou seja, a primeira etapa da abordagem de evolução do esquema de dados apresentada na Seção 3.1.5) tenha sido concluída, o módulo Gerenciador de Persistência pode utilizar o esquema candidato para acessar os dados do SaS em uma condição de estado misto, fornecendo o comportamento que é exigido pela funcionalidade **F3** da biblioteca DynaSchema que foi descrita na Seção 4.1.3. Vale destacar que os esquemas corrente e candidato, destacados na caixa de linha vermelha na Figura 30, são armazenados pelo módulo Gerenciador de Estado na forma de metadados, cuja estrutura é modelada pela classe `State` do pacote `state::model`.

Do ponto de vista comportamental, a classe `SchemaManager` fornece alguns métodos para alterar o esquema corrente do SaS em dois níveis estruturais diferentes, cujo resultado das manipulações será armazenado no esquema candidato para evitar a perturbação das informações contidas no esquema corrente. O **primeiro nível** aborda as mudanças na estrutura da entidade, que inclui as funcionalidades para adicionar, remover, recuperar e verificar se uma entidade existe no esquema de dados do SaS (essas funções são executadas respectivamente pelos métodos addEntity(), `removeEntity()`, `getEntity()` e `containsEntity()`). Por outro lado, o **segundo nível** está relacionado com as mudanças nos campos/atributos, que inclui as atividades para adicionar, renomear, alterar o tipo e remover um campo/atributo de uma entidade existente no esquema do SaS (tais funcionalidades são executadas através dos métodos `addEntityField()`, `renameEntityField()`, `changeEntityFieldType()` e `removeEntityField()`, respectivamente). Vale salientar que no contexto do módulo Gerenciador de Esquema, o termo entidade se refere a uma classe relacionada com a abordagem de mapeamento objeto-relacional baseada em anotações (Seção 4.1.4), cujas alterações na estrutura devem ser rastreadas pela biblioteca DynaSchema para apoiar a evolução do esquema de dados do SaS.

4.2.5 Módulo Gerenciador de Persistência

O módulo Gerenciador de Persistência permite que as informações de um SaS possam ser armazenadas e recuperadas de um banco de dados relacional. Para isso, este módulo foi projetado com base na JPA[12], uma API amplamente conhecida pela comunidade Java. Nesse sentido, a adoção dessa API promove o reaproveitamento de conhecimentos por parte dos desenvolvedores, diminuindo o impacto inicial da utilização da biblioteca DynaSchema em um projeto de SaS. Além disso, este módulo implementa o componente da interface de persistência de dados da biblioteca DynaSchema que foi abordada na Seção 4.1.2 e está relacionada com as funcionalidades **F1** e **F3** que foram descritas na Seção 4.1.3. Na Figura 31 são ilustradas as classes do módulo Gerenciador de Persistência.

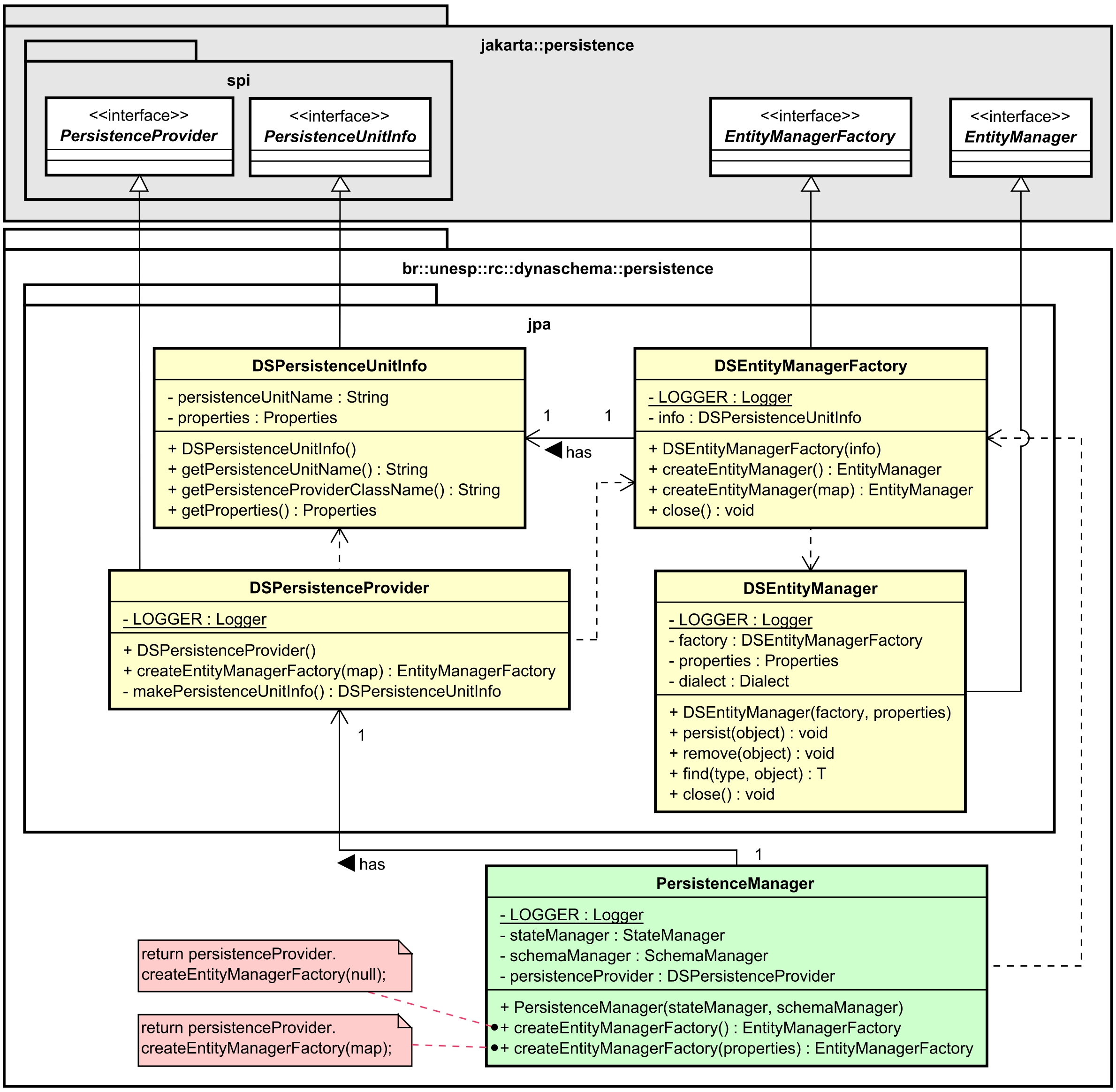
A classe `PersistenceManager` estabelece o "ponto de entrada" para acessar as funcionalidades de persistência da biblioteca DynaSchema. Internamente, essa classe referencia os módulos Gerenciador de Estado e Gerenciador de Esquema (atributos `stateManager` e `schemaManager`, respectivamente), além do utilitário para acessar as funcionalidades da JPA (atributo `persisitenceProvider`), cuja implementação está contida no pacote `jpa`. Além disso, a classe `PersistenceManager` contém o método `createManagerFactory()` para obter um objeto do tipo `EntityManagerFactory`, cuja instância é retornada pela classe `DSPersistenceProvider` do pacote `jpa` via atributo `persisitenceProvider`, conforme destacado nas anotações de cor salmão da Figura 31.

No pacote `jpa` estão contidas as classes que implementam as interfaces essenciais da JPA. Tais classes possuem o mesmo padrão de nomenclatura (ou seja, o prefixo "`DS`") utilizado no módulo Gerenciador de Esquema. Vale salientar que o diagrama ilustrado na Figura 31 apresenta as classes desse pacote de maneira resumida, pois as interfaces da JPA são extensas e somente as funcionalidades relativas à biblioteca DynaSchema foram implementadas. Uma síntese das classes que compõem do pacote `jpa` é apresentada a seguir:

- **`DSPersistenceUnitInfo`**: esta classe é uma extensão da interface `PersistenceUnit-Info` e armazena as informações de uma unidade de persistência. No contexto da JPA, essa unidade representa um conjunto de configurações para definir o comportamento dessa API, estabelecer as informações de conexão com o banco de dados, manipular o mapeamento entre classes e tabelas, entre outras funcionalidades. Esta classe é utilizada

[12] Foi utilizada a interface da JPA mantida pelo projeto Jakarta, disponível em: <https://jakarta.ee/specifications/persistence>. Essa interface também é utilizada no projeto da biblioteca Hibernate (veja Seção 4.1.2).

Figura 31 – Diagrama de classes do Módulo Gerenciador de Persistência



Fonte: Elaborada pelo autor.

pela biblioteca DynaSchema de maneira "virtual", pois as informações contidas em uma unidade de persistência são armazenadas no módulo Gerenciador de Estado;

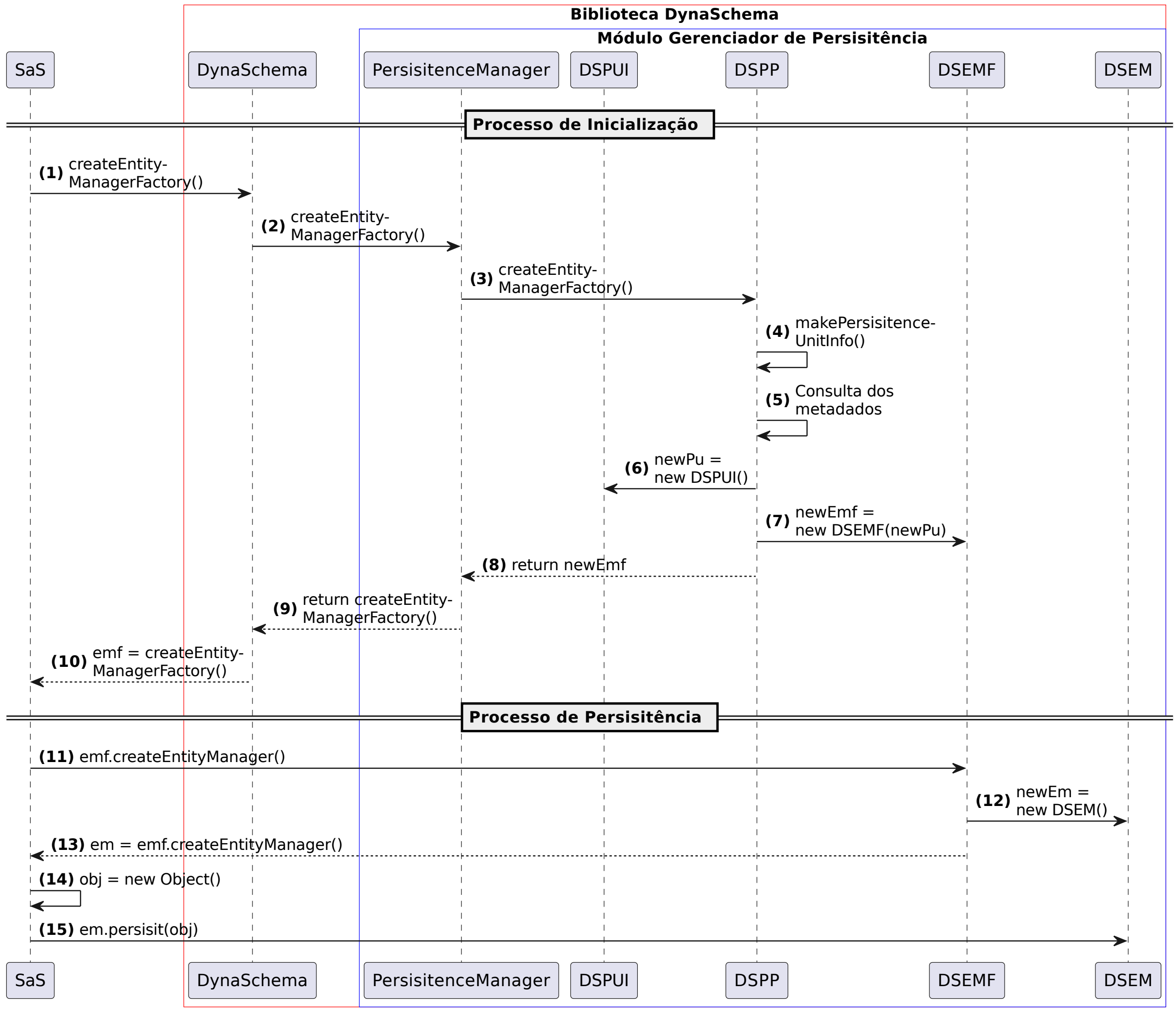
- **`DSPersisitenceProvider`**: esta classe é uma extensão da interface `PersisitenceProvider`, sendo responsável por encontrar e extrair os dados de uma unidade de persistência. No contexto da biblioteca DynaSchema, o método `makePersisitenceUnitInfo()` da classe `DSPersisitenceProvider` consulta os metadados fornecidos pelo módulo Gerenciador de Estado para montar uma unidade de persistência que será utilizada pela classe `DSEntityManagerFactory`. Além disso, o método `createEntityManagerFactory()` permite a criação de instâncias dessa última classe já configuradas com uma unidade de persistência montada pelo método `makePersisitenceUnitInfo()`;

- `DSEntityManagerFactory`: esta classe é uma extensão da interface `EntityManager-Factory` e fornece o método `createEntityManager()` para criar objetos do tipo `EntityManager`. Para isso, são utilizadas as informações contidas em uma unidade de persistência provida pela classe `DSPersisitenceProvider` e referenciada pelo atributo `info` da classe `DSEntityManagerFactory`, cujas configurações definirão a maneira como as instâncias do tipo `EntityManager` serão produzidas. Nesse sentido, esta classe permite a criação de instâncias padronizadas com os mesmos parâmetros de conexão com o banco de dados e estratégia de dialeto SQL, por exemplo; e

- `DSEntityManager`: esta classe é uma extensão da interface `EntityManager`, sendo responsável pela implementação dos métodos que oferecem as funcionalidades de persistência de dados, como a inserção ou atualização (`persisit()`), remoção (`remove()`) e busca (`find()`) das informações dos objetos no banco de dados. No contexto da biblioteca DynaSchema, a classe `DSEntityManager` utiliza o módulo Gerenciador de Esquema para obter o plano de mapeamento entre as classes dos objetos persistidos e suas respectivas tabelas. Para isso, a API de reflexão da linguagem Java é utilizada para "montar" e "desmontar" esses objetos. Em outras palavras, essa API permite criar um objeto com base nos seus dados armazenados e salvar as informações desse objeto no banco de dados. Além disso, o atributo `properties` dessa classe provê um meio de sobrescrever os parâmetros da unidade de persistência, possibilitando a escolha do mapeamento apontado pelo esquema atual ou candidato (Seção 4.1.4). Esse tipo de configuração está relacionado com a funcionalidade **F3** da biblioteca DynaSchema que foi apresentada na Seção 4.1.3.

Na Figura 32 é ilustrado um exemplo do funcionamento do mecanismo de persistência da biblioteca DynaSchema em duas etapas principais, a saber: (i) *o Processo de Inicialização*, que instancia a fábrica de objetos do tipo `EntityManager`; e (ii) *o Processo de Persistência*, que utiliza um desses últimos objetos para armazenar e recuperar as informações do SaS de um banco de dados relacional. Além disso, vale salientar que os nomes das classes contidas no pacote `jpa` do módulo Gerenciador de Persistência foram abreviados na Figura 32 por questões de espaço, sendo que a convenção de siglas está presente na legenda da referida figura. Uma descrição das duas etapas supracitadas é apresentada a seguir:

Figura 32 – Funcionamento do mecanismo de persistência



Legenda

Sigla	Nome da Classe
DSPUI	DSPersistenceUnitInfo
DSPP	DSPersisitenceProvider
DSEMF	DSEntityManagerFactory
DSEM	DSEntityManager

Fonte: Elaborada pelo autor.

Processo de Inicialização. Nesta etapa o SaS deve solicitar a criação de um objeto do tipo `EntityManagerFactory` para a biblioteca DynaSchema **(Passo 1)**, que encaminha a solicitação de instanciação para a classe `PersisitenceManager` **(Passo 2)**. Por sua vez, essa classe utiliza o método `createEntityFactory()` fornecido por `DSPersisitenceProvider` para obter a instância do tipo `EntityManagerFactory` **(Passo 3)**. Internamente, a classe `DSPersisitenceProvider` produz uma unidade de persistência através do método `makePersistenceUnitInfo()` **(Passo 4)** baseada nos metadados fornecidos pelo módulo Gerenciador de Estado **(Passo 5)**. A referência para essa unidade de persistência é armazenada na variável `newPu` **(Passo 6)**, sendo posteriormente utilizada na instanciação do objeto do tipo `EntityManagerFactory` **(Passo 7)**. No final desta etapa de inicialização, esse objeto retorna pela cadeia de chamadas até o SaS **(Passos 8 e 9)**, cuja referência é guardada na variável `emf` **(Passo 10)**.

Processo de Persistência. Nesta etapa o SaS utiliza o objeto do tipo `EntityManagerFactory` (referenciado pela variável `emf`) para obter uma instância da classe `DSEntityManager` **(Passos 11 e 12)**, cuja referência é armazenada na variável `em` **(Passo 13)**. A partir desse ponto, pode-se dizer que o SaS está preparado para armazenar ou recuperar suas informações do banco de dados. Dado uma determinada instância **(Passo 14)**, o SaS pode utilizar o método `persist()` da classe `DSEntityManager` (referenciada pela variável `em`) para persistir os dados desse objeto de interesse **(Passo 15)**.

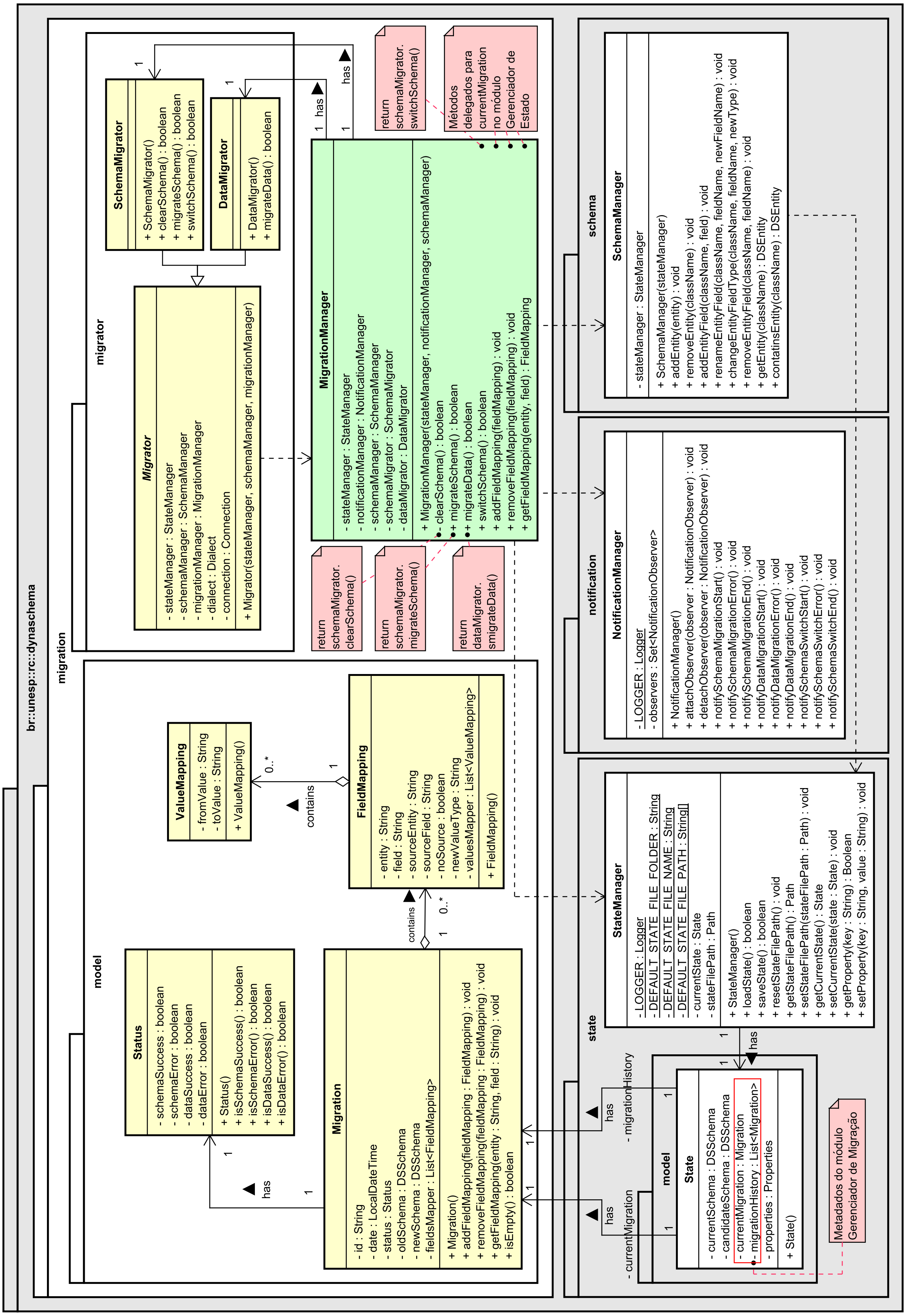
4.2.6 Módulo Gerenciador de Migração

O módulo Gerenciador de Migração fornece o mecanismo para lidar com a evolução do esquema de dados de um SaS. Em síntese, este módulo permite que o SaS possa fazer as alterações no seu esquema de dados em tempo de execução com base em três atividades principais, a saber: (i) migração do esquema de dados; (ii) migração dos dados entre a versão antiga do esquema e a nova versão; e (iii) troca de contexto para a nova versão do esquema de dados. Sobre essa última atividade, vale destacar que no processo de troca do esquema de dados também está incluída a eliminação da versão antiga do esquema de banco de dados que deixou de ser referenciada pelo SaS. Além disso, o módulo Gerenciador de Migração implementa o componente da interface de evolução do esquema de dados da biblioteca DynaSchema (Seção 4.1.2) e está relacionada com as funcionalidades **F1**, **F2**, **F4** e **F6** que foram apresentadas na Seção 4.1.3. A organização interna das classes deste módulo é ilustrada na Figura 33.

As classes do pacote `migration::model` são utilizadas pela biblioteca DynaSchema para estabelecer um modelo para o plano de migração do esquema de dados. Nesse sentido, tais classes armazenam todas as informações necessárias para processar a migração do esquema de dados do SaS. Internamente, o pacote `migration::model` possui quatro classes, a saber:

- **`Migration`:** esta classe modela o corpo de um plano de migração e seus atributos carregam as informações sobre o identificador da migração, a data da sua criação, as referências para os esquemas corrente e candidato da aplicação, além da lista de mapeamento de valores,

Figura 33 – Diagrama de classes do Módulo Gerenciador de Migração



Fonte: Elaborada pelo autor.

caso seja especificada pelo SaS. Vale salientar que o esquema candidato armazena as alterações que foram feitas no esquema corrente. Isso significa que o esquema corrente contém as informações da versão antiga do esquema de dados e o esquema candidato carrega as informações da nova versão de esquema do SaS;

- **`Status`:** esta classe é responsável por armazenar o status do processamento da migração do esquema de dados do SaS. Nesse sentido, pode-se saber se a migração do esquema e/ou dos dados foram executadas com sucesso ou com erros;

- **`FieldMapping`:** esta classe tem a função de indicar como o mapeamento de dados deve ser realizado. Assim é possível especificar se os valores do campo/atributo de uma entidade devem ser preenchidos com base em outro campo/atributo da mesma ou de outra entidade, se é um novo campo/atributo e não possui uma correspondência na versão antiga do esquema de dados, ou ainda se o campo/atributo mudou de tipo e é necessário realizar um mapeamento de valores com relação à versão antiga do esquema de dados. Vale ressaltar que a classe `FieldMapping` está relacionada com a funcionalidade **F4** da biblioteca DynaSchema que foi apresentada na Seção 4.1.3; e

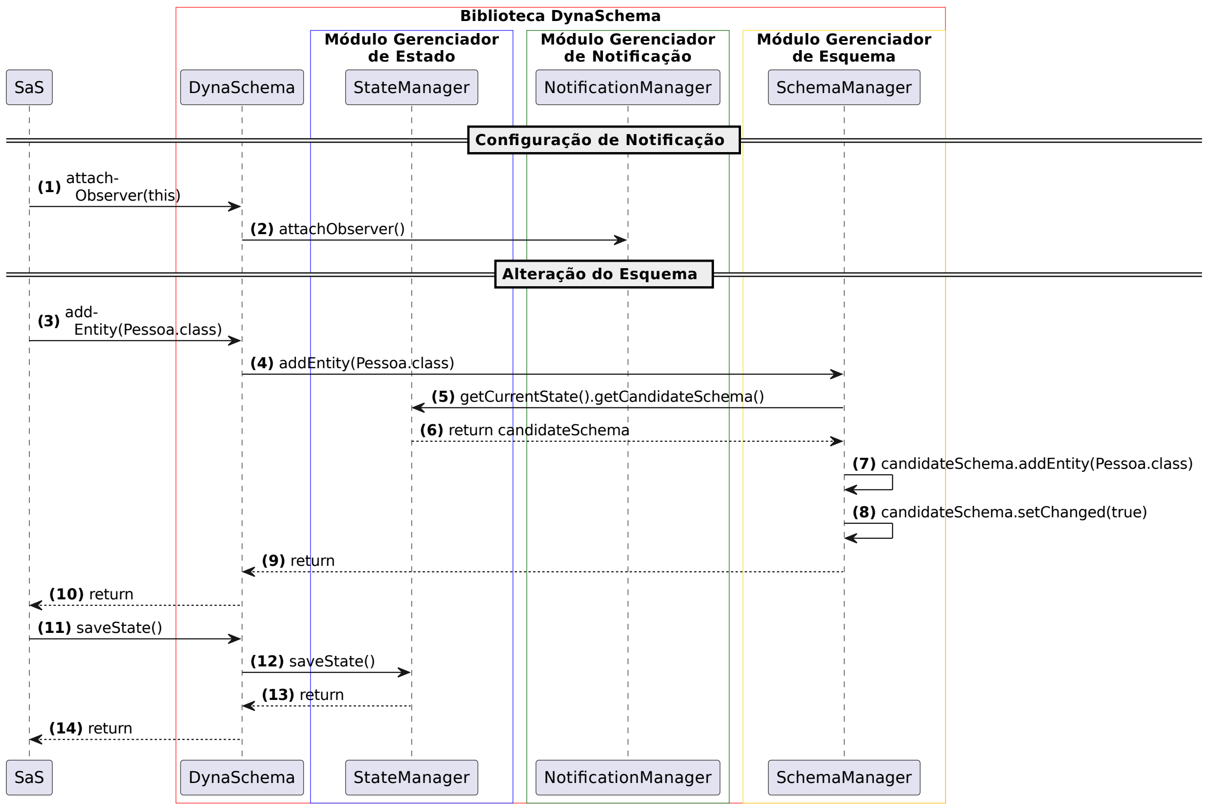
- **`ValueMapping`:** esta classe é responsável pelo mapeamento entre os valores dos campos/atributos da versão antiga do esquema de dados e a nova versão. Nesse sentido, essa classe estabelece um mapeamento do tipo "um para um" entre o valor antigo e o novo valor correspondente (ou seja, uma substituição simples).

No pacote `migration::migrator` estão contidas as classes utilitárias para processar a migração do esquema e dos dados. A classe abstrata `Migrator` contém a estrutura comum de um "migrador genérico", cujos atributos referenciam as instâncias dos módulos Gerenciador de Estado, Gerenciador de Esquema e Gerenciador de Migração (atributos `stateManager`, `schemaManager` e `migrationMnager`, respectivamente), o objeto de dialeto SQL (atributo `dialect`) e a conexão com o banco de dados (atributo `connection`). Além disso, essa classe é especializada em outras duas subclasses para separar as duas etapas levantadas pela abordagem de evolução do esquema de dados descrita na Seção 3.1.5. A subclasse `SchemaMigrator` é responsável pelo processamento da etapa de migração do esquema de dados da aplicação. Nesse sentido, essa subclasse fornece os métodos para "destruir" o esquema de dados, realizar a migração do esquema e trocar para a versão mais atual do esquema (ou seja, os métodos `clearSchema()`, `migrateSchema()` e `switchSchema()`, respectivamente). Vale salientar que o método `clearSchema()` é utilizado para limpar uma migração de esquema de dados mal sucedida. Por outro lado, a subclasse `DataMigrator` é responsável pela etapa de migração dos dados, cuja tarefa pode ser executada através da chamada do seu método `migrateData()`.

A classe `MigrationManager` estabelece o "ponto de entrada" para acessar as funcionalidades de evolução do esquema de dados da biblioteca DynaSchema. Seus atributos referenciam as instâncias dos módulos Gerenciador de Estado (atributo `stateManager`), Gerenciador de Notificação (atributo `notificationManager`) e Gerenciador de Esquema (atributo `schemaManager`), além dos migradores de esquema e dados (atributos `schemaMigrator` e `dataMigrator`, respectivamente). Em relação a coordenação entre essa classe e os outros módulos da biblioteca DynaSchema, o módulo Gerenciador de Estado é utilizado para salvar as alterações que foram feitas no plano atual de migração, o módulo Gerenciador de Notificação fornece o meio para notificar o SaS sobre a execução das etapas de evolução do esquema de dados e o módulo Gerenciador de Esquema é utilizado para acessar os esquemas corrente e candidato da aplicação. Vale salientar que o plano de migração atual e o histórico de migrações são armazenados nos metadados do módulo Gerenciador de Estado, sendo destacados na Figura 33 através de uma caixa de linha vermelha. A classe `MigrationManager` também é responsável pela configuração do objeto de dialeto SQL (conforme ilustrado na Figura 29) e da conexão com o banco de dados utilizados pelos migradores de esquema e de dados. Além disso, de acordo com as anotações de cor salmão presentes no pacote `migration` da Figura 33, essa classe repassa as chamadas das etapas de evolução para os migradores de esquema e de dados.

Para ilustrar o funcionamento do módulo Gerenciador de Migração, pode-se citar a adição de uma nova entidade `Pessoa` no esquema de dados de um SaS. Para facilitar a compreensão desse processo, o exemplo de funcionamento foi dividido em cinco etapas, sendo que as duas primeiras podem ser vistas na Figura 34. Na **primeira etapa** o SaS utiliza a biblioteca DynaSchema para se registrar no módulo Gerenciador de Notificação, fazendo com que o SaS possa ser informado sobre os eventos lançados durante o processo de evolução do seu esquema de dados **(Passos 1 e 2)**. Em seguida, na **segunda etapa** o SaS realiza o registro da entidade `Pessoa` no seu esquema. Para isso, o método `addEntity()` da biblioteca DynaSchema deve ser utilizado para adicionar a nova entidade no esquema de dados da aplicação **(Passo 3)**. Essa chamada é transferida para o módulo Gerenciador de Esquema **(Passo 4)**, que recupera o esquema candidato do SaS **(Passos 5 e 6)**, adiciona a entidade `Pessoa` na sua lista de entidades **(Passo 7)** e altera a *flag* que indica que esse esquema foi modificado **(Passo 8)**. Ao final desse processo de adição da nova entidade `Pessoa`, seu estado de sucesso de execução é retornado pela cadeia de chamadas até chegar no SaS **(Passos 9 e 10)**. Por sua vez, o SaS pode chamar o método `saveState()` da biblioteca DynaSchema para salvar as alterações no seu esquema de dados candidato **(Passos 11 e 12)**, cujo status de sucesso de execução será retornado pela cadeia de chamadas até o SaS **(Passos 13 e 14)**.

Figura 34 – Passos para as duas primeiras etapas do exemplo de funcionamento para migração de esquema



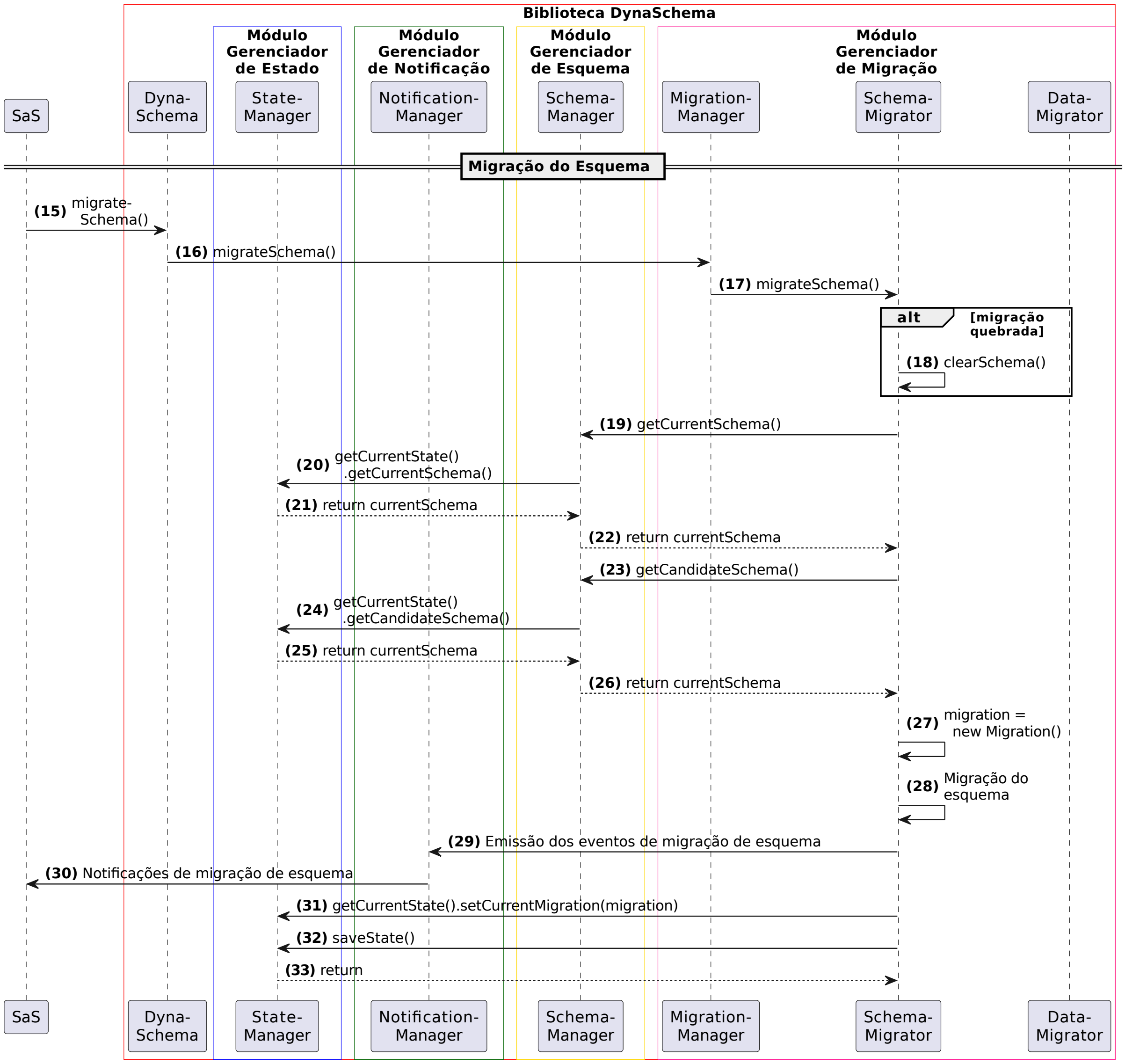
Fonte: Elaborada pelo autor.

Na Figura 35 é ilustrada a **terceira etapa** do exemplo de funcionamento do módulo Gerenciador de Migração, onde o SaS solicita a migração do seu esquema de dados. Para isso, o SaS deve executar o método `migrateSchema()` **(Passo 15)**, cuja chamada será transferida para o módulo Gerenciador de Migração **(Passo 16)**, que será responsável por encaminhar a solicitação de uma nova migração do esquema de dados da aplicação para o migrador de esquema **(Passo 17)**. Caso seja encontrada uma migração corrente mal executada, o migrador de esquema executa a remoção das tabelas do banco de dados antes de tentar uma nova migração do esquema de dados **(Passo 18)**. Antes de montar um novo plano de migração, esse migrador recupera as referências para os esquemas corrente **(Passos 19 a 22)** e candidato **(Passos 23 a 26)** do módulo Gerenciador de Esquema. Vale lembrar que esses esquemas estão armazenados nos metadados gerenciados por esse último módulo. Em seguida, o migrador de esquema cria um novo plano de migração **(Passo 27)**, onde seus atributos `oldSchema` e `newSchema` (veja a classe `Migration` do pacote `migration::model` ilustrada na Figura 33) armazenam as referências para os esquemas corrente e candidato, respectivamente. Esse plano é utilizado para executar a migração do esquema do SaS **(Passo 28)**, cujo processo utiliza o módulo Gerenciador de Notificação **(Passo 29)** para enviar os eventos sobre o andamento da migração do esquema para o SaS **(Passo 30)**. Por motivos de espaço, as emissões desses eventos foram representadas somente após o processo de migração dos dados **(Passo 29)**. Entretanto, conforme descrito na Seção 4.2.2, existem notificações específicas para avisar quando o processo de migração do esquema é iniciado, finalizado ou executado com erros. Além disso, o plano de migração é armazenado posteriormente nos metadados do módulo Gerenciado de Estado **(Passos 31 a 33)**.

Após a execução de uma migração de esquema com sucesso, pode-se prosseguir para a **quarta etapa** do exemplo de funcionamento do módulo Gerenciador de Migração, como ilustra a Figura 36. Nessa etapa, o SaS solicita a execução da migração dos dados da versão antiga do esquema de dados para a nova versão. Para isso, o SaS deve chamar o método `migrateData()` da biblioteca DynaSchema **(Passo 34)**, que encaminha a solicitação de uma nova migração dos dados através do módulo Gerenciador de Migração **(Passo 35)** até chegar no migrador de dados **(Passo 36)**. Em seguida, o plano de migração é recuperado dos metadados do módulo Gerenciado de Estado **(Passos 37 e 38)**, sendo posteriormente utilizado para executar o processo de migração dos dados **(Passo 39)**. Por sua vez, esse processo utiliza o módulo Gerenciador de Notificação para emitir os eventos de migração dos dados **(Passo 40)** que são "escutados" pelo SaS **(Passo 41)**. Assim como na terceira etapa deste exemplo de funcionamento, a emissão desses eventos foi ilustrada de maneira simplificada nesta etapa. Ao final do processo de migração dos dados, o status de execução do plano de migração é atualizado **(Passo 42)** e suas informações são salvas os metadados armazenados pelo módulo Gerenciador de Estado **(Passos 43 a 45)**.

Por fim, na Figura 37 é ilustrada a **quinta etapa** do exemplo de funcionamento do módulo Gerenciador de Migração, onde os SaS realiza a troca definitiva para o contexto da nova versão do esquema de dados da aplicação. Vale ressaltar que essa troca de contexto só pode ser feita após a execução bem sucedida das etapas de migração do esquema e dos dados. Para realizar a troca do esquema, o SaS deve invocar o método `switchSchema()` da biblioteca DynaSchema **(Passo 46)**, cuja chamada é propagada entre o módulo Gerenciador de Esquema **(Passo 47)** e o migrador de esquema **(Passo 48)**. A execução da troca de esquema começa com

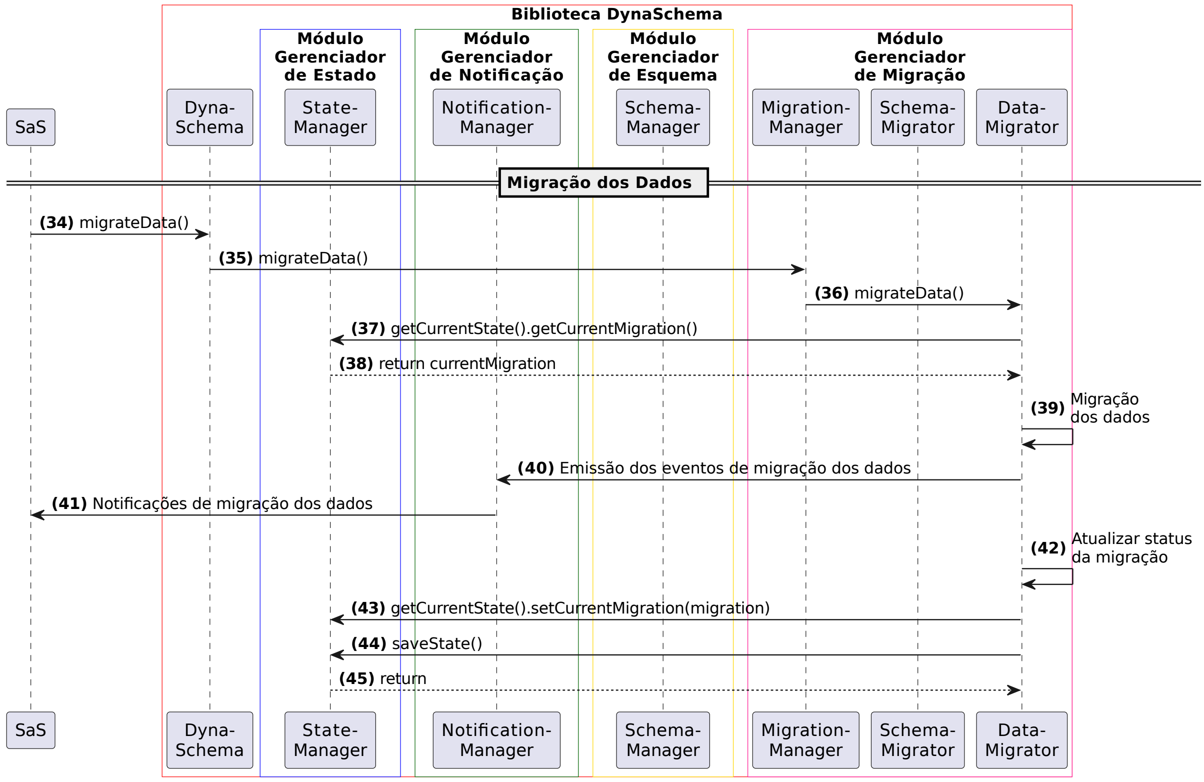
Figura 35 – Passos para terceira etapa do exemplo de funcionamento para migração de esquema



Fonte: Elaborada pelo autor.

a recuperação do esquema candidato através do módulo Gerenciador de Esquema **(Passos 49 a 52)**. Em seguida, é realizada uma alteração de apontamentos, onde o esquema corrente é substituído pelo esquema candidato **(Passos 53 e 54)** e o esquema candidato é substituído por uma cópia de sí próprio **(Passos 55 e 56)**. Dessa maneira, o novo esquema candidato pode ser modificado pelo SaS sem afetar o esquema corrente da aplicação. Após a conclusão da troca de apontamentos, a versão antiga do esquema de dados do SaS é eliminada **(Passo 57)** para liberar espaço de armazenamento no banco de dados. Durante o processo de atualização dos apontamentos, o módulo Gerenciador de Notificação é utilizado para enviar os eventos sobre a troca de esquema **(Passos 58 e 59)**. Assim como ocorreu nas etapas anteriores, a representação do processo de notificação também foi simplificada nesta etapa. Ao final da troca de esquema,

Figura 36 – Passos para quarta etapa do exemplo de funcionamento para migração de esquema



Fonte: Elaborada pelo autor.

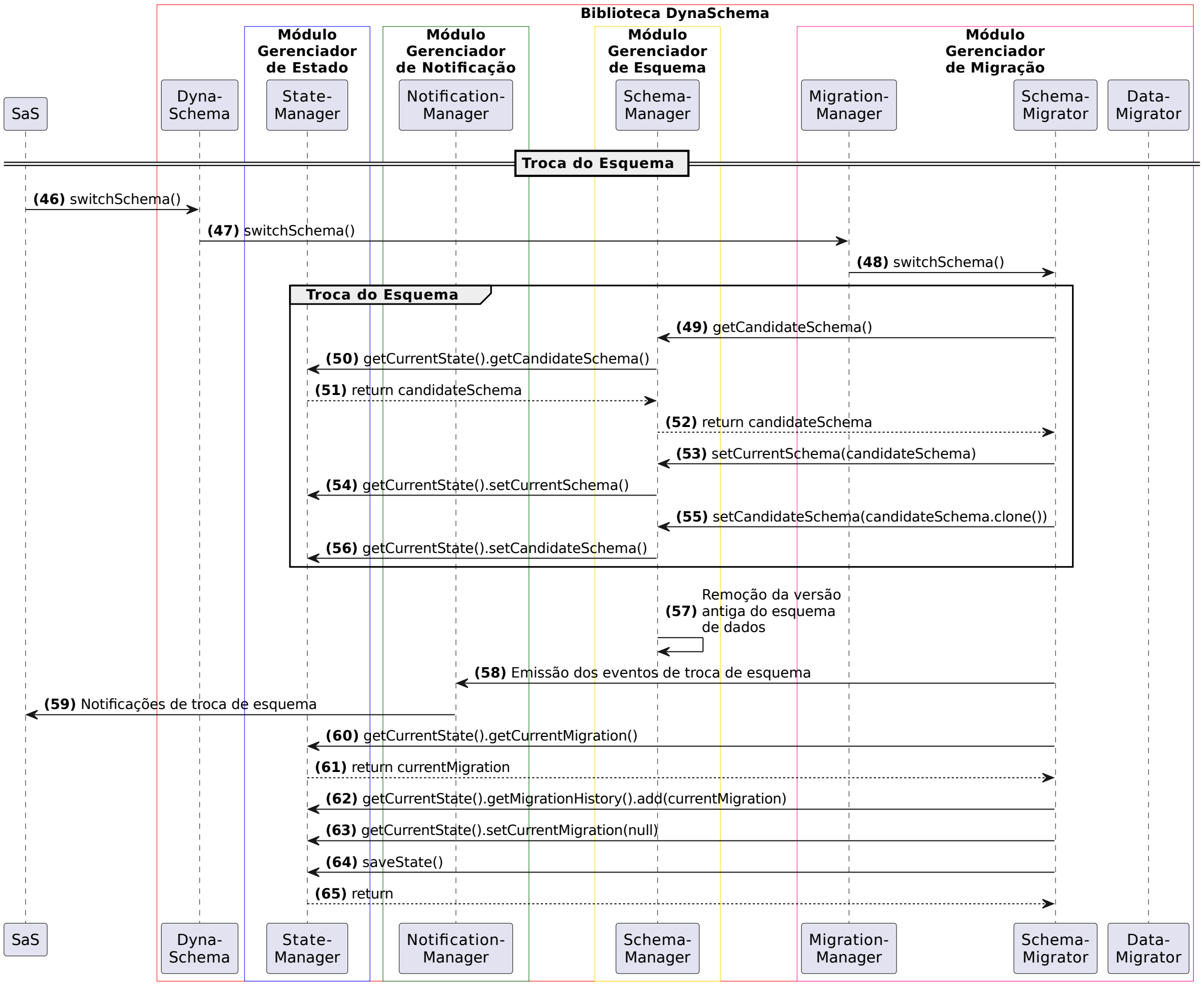
o plano de migração é recuperado **(Passos 60 e 61)** para que possa ser adicionado ao histórico de migrações da aplicação **(Passo 62)**. Além disso, é removido o apontamento para o objeto do plano de migração corrente **(Passo 63)** e todas as alterações que foram feitas no processo de troca de esquema é salvo nos metadados do módulo Gerenciador de Estado **(Passos 64 e 65)**.

4.2.7 Interface da Biblioteca

Conforme discutido na Seção 4.1.4, o estilo arquitetural modular foi adotado para lidar com a complexidade interna da biblioteca DynaSchema, cujos detalhes de *design* foram reportados nas Seções 4.2.1 a 4.2.6. Além disso, a visão geral dos seis módulos que constituem a arquitetura dessa biblioteca e seus relacionamentos podem ser consultados na Figura 24. Por um lado, esses módulos facilitam a organização e a manutenção das funcionalidades da biblioteca DynaSchema. Por outro lado, o SaS deve lidar com processo de instanciação desses módulos e a injeção de suas dependências. Para resolver esse problema, foi criada uma interface para facilitar a utilização dessa biblioteca por seus usuários, cujas classes podem ser vistas na Figura 38.

A classe `DynaSchema` modela a interface de utilização da biblioteca DynaSchema,

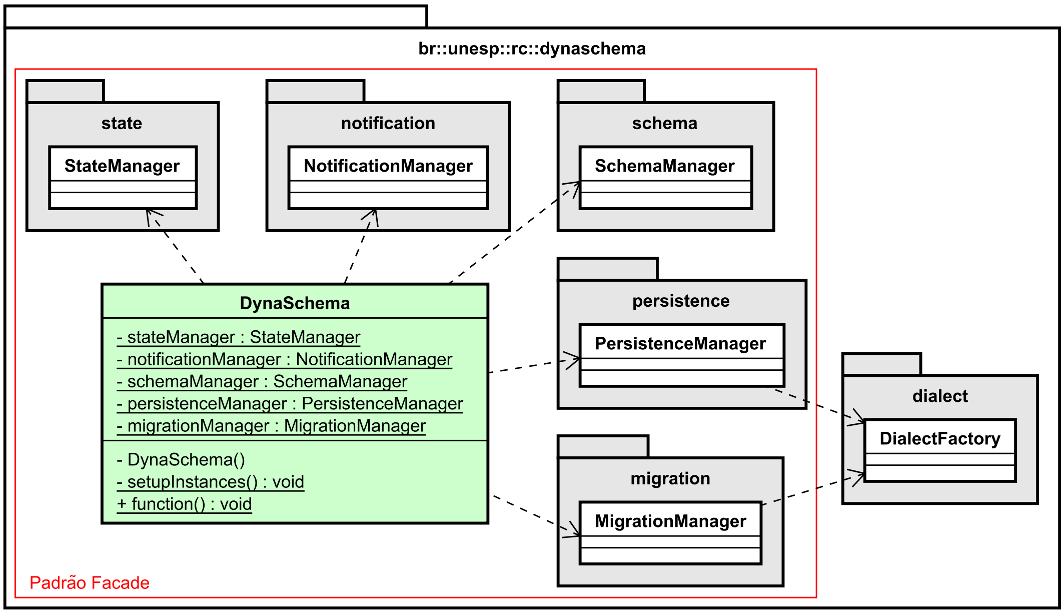
Figura 37 – Passos para quinta etapa do exemplo de funcionamento para migração de esquema



Fonte: Elaborada pelo autor.

atuando como um utilitário para simplificar o acesso às suas funcionalidades. Internamente, essa classe utiliza o método `setupInstances()` para instanciar os cinco módulos da biblioteca, a saber: (i) o atributo `stateManager` referencia o Gerenciador de Estado; (ii) o atributo `notificationManager` contém a instância do Gerenciador de Notificação; (iii) o atributo `schemaManager` guarda a referência para o Gerenciador de Esquema; (vi) o atributo `persistenceManager` aponta para a instância do Gerenciador de Persistência; e (vi) o atributo `migrationManager` guarda o apontamento para a instância do Gerenciador de Migração. Vale lembrar que o módulo de Dialeto é utilizado indiretamente pela interface da biblioteca DynaSchema através dos módulos Gerenciador de Persistência e Gerenciador de Migração, cujo processo de instanciação do objeto de dialeto SQL foi ilustrado na Figura 29. Além disso, a classe `DynaSchema` contém vários métodos estáticos (representados de maneira resumida pelo método `function()`) que delegam as chamadas das funcionalidades dessa biblioteca para seus

Figura 38 – Diagrama de classes da interface da biblioteca DynaSchema



Fonte: Elaborada pelo autor.

respectivos módulos. A abstração da biblioteca DynaSchema fornecida por sua interface com os usuários finais (ou seja, a classe `DynaSchema`) está relacionada com o padrão *Facade*[13], cuja representação é marcada na Figura 29 através de uma caixa de linha vermelha.

Na Listagem 1 é apresentado um exemplo de código para entender como a biblioteca DynaSchema é utilizada pelos seus usuários. Tal código é escrito em linguagem Java e apresenta de maneira prática o cenário abordado nas Figuras 34, 35, 36 e 37 da Seção 4.2.6, onde uma nova entidade `Pessoa` é adicionada no esquema de dados do SaS. Inicialmente, é declarada a classe principal do programa (**linha 1**), nomeada como `Main`, que implementa a interface `NotifiactionObserver` (veja o diagrama de classes apresentado na Figura 26 da Seção 4.2.2) para "escutar" os eventos lançados pela biblioteca DynaSchema. Em seguida, é declarado o corpo do método `main()` (**linhas 3 a 15**) que é chamado automaticamente quando a aplicação é executada. Dentro desse método, a classe `Main` se registra na biblioteca para ser notificado sobre os eventos de evolução do esquema de dado (**linha 5**), adiciona a nova entidade `Pessoa` no esquema de dados da aplicação (**linha 7**), salva o estado desse esquema (**linha 8**) e solicita a atividade de migração do esquema (**linha 10**). Quando essa atividade for finalizada, será chamado o método `onSchemaMigrationEnd()` da classe `Main` (**linhas 13 a 16**), cuja implementação dispara a migração dos dados da aplicação (**linha 15**). Por sua vez, quando essa segunda atividade

[13] Mais detalhes sobre o padrão *Facade* podem ser consultados na Seção 2.1.

for finalizada, será chamado o método `onDataMigrationEnd()` da classe `Main` (**linhas 18 a 21**). Dentro desse método, é invocada a atividade de troca do esquema (**linha 20**) para concluir a evolução do esquema de dados da aplicação. Finalmente, quando essa última atividade termina, será chamado o método `onSchemaSwitchEnd()` da classe `Main` (**linhas 23 a 25**), que exibe uma mensagem no *console* sobre o fim da evolução do esquema de dados da aplicação (**linha 24**). De acordo com o exemplo de código contido na Listagem 1, pode-se observar que todas as funcionalidades da biblioteca DynaSchema são acessadas através da classe `DynaSchema`, cuja interface foi projetada para facilitar a interação entre a biblioteca e seus usuários.

Listagem 1 – Exemplo de utilização da biblioteca DynaSchema

```
1  public class Main implements NotificationObserver {
2
3      public static void main(String[] args) {
4          // Configuração de notificação
5          DynaSchema.attachObserver(this);
6          // Alteração do esquema
7          DynaSchema.addEntity(Pessoa.class);
8          DynaSchema.saveState();
9          // Migração do esquema
10         DynaSchema.migrateSchema();
11     }
12
13     public void onSchemaMigrationEnd() {
14         // Migração dos dados
15         DynaSchema.migrateData();
16     }
17
18     public void onDataMigrationEnd() {
19         // Troca do esquema
20         DynaSchema.switchSchema();
21     }
22
23     public void onSchemaSwitchEnd() {
24         System.out.println("Evolução Finalizada!");
25     }
26
27     // Os outros métodos de interceptação das notificações foram omitidos ...
28
29 }
```

Fonte: Elaborada pelo autor

4.3 Considerações Finais

Neste capítulo foram relatados os principais pontos envolvidos na elaboração da biblioteca DynaSchema desenvolvida neste projeto de mestrado. Inicialmente, na Seção 4.1 foram apresentadas os detalhes de projeto da biblioteca, onde foram levantadas as principais preocupações que estão envolvidas no processo de autoadaptação de um SaS com persistência de dados (Seção 4.1.1), as questões de projeto relacionadas com seus componentes funcionais (Seção 4.1.2), seu conjunto de funcionalidades (Seção 4.1.3) e a estrutura da sua arquitetura interna e os relacionamentos dos seus componentes internos (Seção 4.1.4). Por fim, foram abordadas as questões de *design* referentes aos seis módulos internos da biblioteca DynaSchema (Seções 4.2.1 a 4.2.6) e sua interface de utilização (Seção 4.2.7).

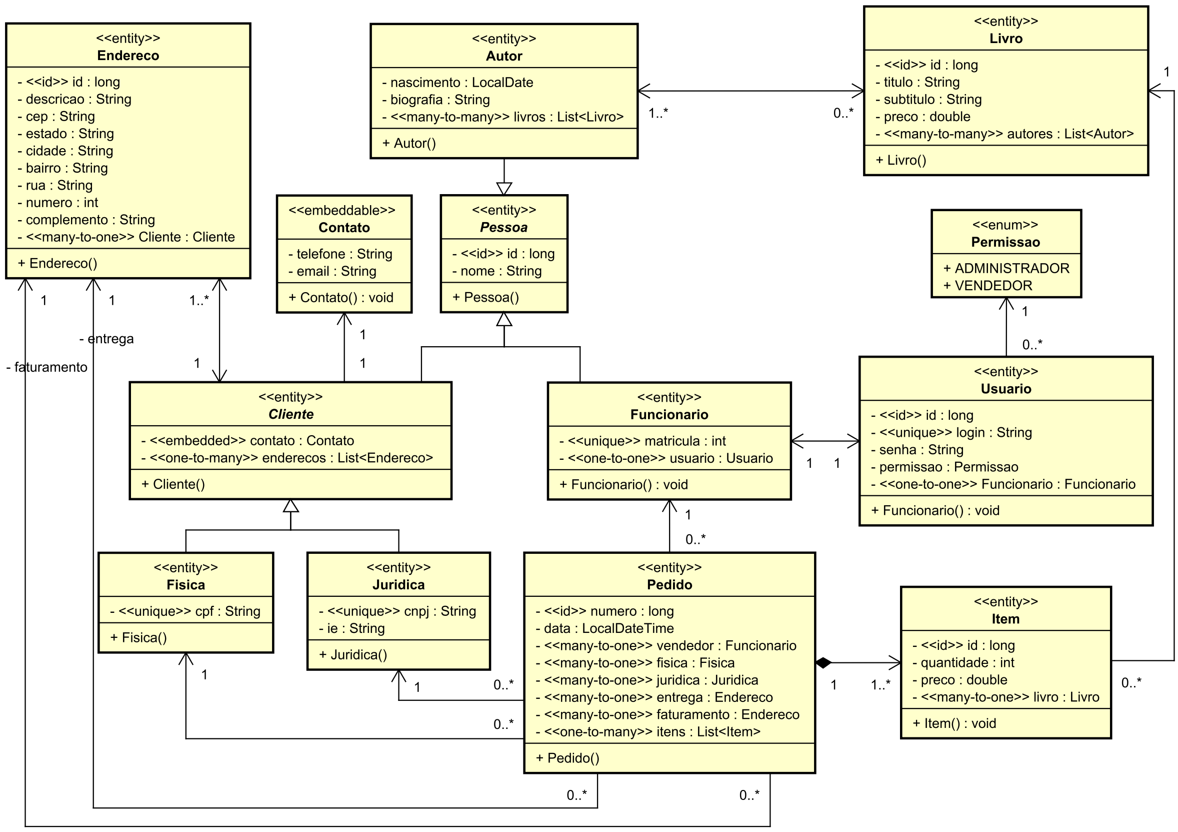
5 Estudo de Caso

Neste capítulo é apresentado o estudo de caso que foi desenvolvido para avaliar a biblioteca DynaSchema em relação aos seus requisitos de projeto, suas funcionalidades e seu comportamento. Inicialmente, na Seção 5.1 é apresentado o sistema alvo que foi utilizado no estudo de caso para avaliar essa biblioteca. Em seguida, na Seção 5.2 é discutido como a arquitetura de referência RA4SaS e a ferramenta de modelagem DSLModeler4SaS são utilizadas para implementar esse sistema alvo. Nas Seções 5.3 e 5.4 é reportado como a biblioteca DynaSchema pode ser utilizada para apoiar o processo da evolução do esquema de dados do sistema alvo. Na Seção 5.5 são discutidos os resultados obtidos na condução deste estudo de caso. Por fim, na Seção 5.6 são apresentadas as considerações finais deste capítulo.

5.1 Sistema Alvo

No Capítulo 4 foram apresentados os detalhes de projeto e *design* da biblioteca DynaSchema que revelam o contexto e o escopo da sua problemática, os componentes funcionais do seu contexto de aplicação, seu conjunto de mínimo de funcionalidades e a descrição dos módulos de sua arquitetura interna. Diante desse contexto, neste capítulo foi elaborado um estudo de caso para avaliar o funcionamento da biblioteca DynaSchema em relação a seus requisitos de projeto. Para isso, tal estudo de caso utiliza essa biblioteca para gerenciar a evolução do esquema de dados em um sistema fictício que administra as vendas de uma distribuidora de livros, referenciado deste ponto em diante como **SisVenda**. De acordo com as regras de negócio dessa distribuidora, seus livros podem ser adquiridos por clientes, que são caracterizados como pessoas física e jurídicas. Em resumo, os clientes devem contatar um dos vendedores da distribuidora de livros para realizar um pedido de compra. Durante o atendimento, o vendedor deve cadastrar o cliente no sistema, caso seja sua primeira compra. Assim, devem ser coletadas suas informações pessoais, dados de contato e endereços para entrega e/ou faturamento. Em seguida, o vendedor deve criar um novo pedido com os livros solicitados pelo cliente. Ao final do atendimento, o vendedor deve conferir as informações do pedido com o cliente, concluir a inclusão desse pedido de venda no sistema e transferir o cliente para o departamento de finanças. Além disso, a distribuidora de livros mantém um catálogo *online* para que seus clientes possam consultar os livros em estoque. O modelo UML (do inglês, *Unified Modeling Language*) do sistema SisVenda pode ser visualizado na Figura 39, cujas classes são descritas a seguir:

Figura 39 – Modelo UML do sistema SisVenda



Fonte: Elaborada pelo autor.

- **Pessoa:** esta classe abstrata serve de base para modelar os usuários que interagem direta ou indiretamente com o sistema SisVenda. Internamente, seus atributos armazenam o identificador do usuário e seu nome completo;

- **Autor:** esta classe contém as informações dos autores dos livros cadastrados no sistema SisVenda. Tais informações podem ser utilizadas para encontrar um determinado livro tanto no catálogo *online* quanto no ato da venda. Nesse sentido, o cliente pode informar o nome do autor para depois encontrar a obra que deseja comprar;

- **Livro:** esta classe modela as informações dos livros que são vendidos pela distribuidora. Essa classe representa o produto do sistema SisVenda, que deve conter tanto seus dados descritivos quanto seu preço de venda;

- `Funcionario`: esta classe representa os funcionários que trabalham na distribuidora de livros e precisam interagir com seu sistema SisVenda. Essa classe herda os atributos de `Pessoa`, além de adicionar as informações de matrícula do funcionário e seu usuário para acessar o sistema de vendas;

- `Usuario`: esta classe modela a conta de um usuário do sistema SisVenda. Nesse contexto, cada funcionário que precisa interagir com o sistema possui seu próprio nome de *login*, senha e nível de permissão;

- `Permissao`: esta enumeração armazena os níveis de permissão das contas de usuário do sistema SisVenda. Uma conta com a permissão `VENDEDOR` só habilita a inclusão dos pedidos de compra no sistema. Por outro lado, a permissão `ADMINISTRADOR` libera o acesso à todos os recursos do sistema;

- `Cliente`: esta classe abstrata armazena as informações de um cliente genérico da distribuidora de livros. Internamente, tal classe contém os dados de contato e endereços do cliente, além de herdar os atributos da classe `Pessoa`;

- `Contato`: esta classe armazena os dados de contato de um cliente da distribuidora de livros, como o número de telefone e endereço eletrônico de *e-mail*;

- `Endereco`: esta classe contém os dados do endereço de um cliente da distribuidora de livros. Para isso, seis atributos armazenam informações como o CEP (ou seja, Código de Endereçamento Postal), sigla do estado, nome da cidade, do bairro e da rua, número do imóvel, informações complementares de referência, entre outros dados;

- `Fisica`: esta classe representa um cliente do tipo pessoa física do sistema SisVenda. Isso significa que sua estrutura herda as informações de um cliente genérico (ou seja, os atributos da classe `Cliente`), além de incluir o do documento de uma pessoa física (ou seja, o número de Cadastro de Pessoa Física – CPF);

- `Juridica`: esta classe representa um cliente do tipo pessoa jurídica do sistema SisVenda. Assim como no caso da pessoa física, esta classe herda os atributos da classe `Cliente` e incluí os documentos de uma pessoa jurídica (ou seja, os números do Cadastro Nacional da Pessoa Jurídica – CNPJ e da Inscrição Estadual – IE);

- `Pedido`: esta classe modela a estrutura de um pedido de venda do sistema SisVenda. Internamente, seus atributos armazenam o número do pedido, a data da venda, o vendedor que incluiu o pedido no sistema, qual foi o cliente que solicitou o pedido, quais são os endereços de entrega e de faturamento, além livros que foram comprados; e

- `Item`: esta classe representa um item do pedido de venda do sistema SisVenda. Esse item deve identificar qual foi o livro comprado, seu preço de venda e quantas unidades foram solicitas pelo cliente. Vale lembrar que no contexto do pedido de venda, cada item está relacionado a somente um livro.

Conforme descrito na Seção 4.1.2, a biblioteca DynaSchema baseia seu mecanismo de persistência de dados de acordo com a interface da JPA, onde as classes relacionadas com a abordagem de mapeamento objeto-relacional devem ser marcadas com as anotações de persistência. Nesse sentido, pode-se observar a presença de alguns esteriótipos na estrutura das classes ilustradas na Figura 39 para representar essas anotações. De acordo com essas marcações no modelo UML do sistema SisVenda, a biblioteca DynaSchema deve ser capaz de extrair as seguintes informações de persistência:

Identificação das entidades e seus atributos. As classes relacionadas com as funcionalidades de persistência de dados devem ser marcadas com a anotação `@Entity` da JPA. No modelo UML ilustrado na Figura 39, as classes com essa anotação possuem o esteriótipo «`entity`» sobre o seu nome. No contexto do mecanismo de persistência da biblioteca DynaSchema, essa informação é necessária para indicar quais são as tabelas que devem ser criadas no banco de dados, dependendo da estratégia de herança utilizada pelo SaS. Por enquanto, a biblioteca suporta a estratégia `TABLE_PER_CLASS` (veja a enumeração `DSInheritanceType` descrita na Seção 4.2.4), onde cada classe é mapeada em uma tabela do banco de dados. Esse mapeamento não consiste em uma tarefa direta, pois as classes abstratas não podem ser instanciadas e consequentemente não possuem uma tabela correspondente no banco de dados. Como exemplo, observe a hierarquia entre as classes `Pessoa`, `Autor`, `Funcionario`, `Cliente`, `Fisica` e `Juridica` do modelo UML contido na Figura 39. Nesse caso, as classes `Pessoa` e `Cliente` são abstratas, assim só é necessário criar as tabelas para as classes `Autor`, `Funcionario`, `Fisica` e `Juridica`.

Em paralelo, vale destacar que a estratégia de herança também possui influência na definição do esquema estrutural das tabelas do banco de dados. No caso da estratégia `TABLE_PER_CLASS`, como cada classe possui sua própria tabela, é necessário rastrear sua hierarquia de herança para identificar todos os seus campos/atributos. Como exemplo, para a classe `Fisica` ilustrada na Figura 39, deve ser criada uma tabela com os campos `id`, `nome`, `telefone`, `email` e `cpf`. Os dois primeiros campos (ou seja, `id` e `nome`) foram herdados da classe `Pessoa`, os dois campos seguintes (ou seja, `nome`, `telefone`) foram embutidos pela classe `Contato` e o último campo pertence diretamente a classe `Fisica`. Nessa direção, também é possível observar a influência de outras anotações da JPA no processo de criação das tabelas do banco de dados. A classe `Contato` está marcada com a anotação `@Embeddable` (representado na Figura 39 com o esteriótipo «`embeddable`»), indicando que essa entidade não possui uma tabela correspondente e seus campos/atributos devem ser embutidos nas tabelas das classes que contém um atributo com o tipo `Contato`. Para que isso seja possível, esse atributo deve ser marcado com a anotação `@Embedded` da JPA, cuja representação é indicada na Figura 39 através do esteriótipo «`embedded`» (veja o atributo `contato` da classe `Cliente`).

Localização das chaves primárias. Como a JPA se baseia em uma abordagem de mapeamento objeto-relacional, as classes assinaladas com a anotação `@Entity` devem possuir um atributo marcado com a anotação `@Id` da JPA. Essa informação será utilizada posteriormente para definir as chaves primárias das tabelas do banco dados, estabelecendo um meio para identificar cada registro armazenado de maneira única. Além disso, as chaves primárias são necessárias criar as associações entre esses registros, cujos relacionamentos são definidos através das chaves estrangeiras presentes nas tabelas do banco de dados. Vale salientar que os atributos marcados com a anotação `@Id` são assinalados na Figura 39 com o esteriótipo «`id`». Assim como no processo de identificação das entidades, a localização das chaves primárias também não é uma atividade direta. Voltando ao exemplo da classe `Fisica`, de acordo com modelo UML ilustrado na Figura 39, sua estrutura não possui diretamente um atributo relacionado com a chave primária. Isso só é possível por causa da hierarquia de herança de classes de `Fisica`, onde sua superclasse `Pessoa` já contém um atributo chamado `id` que foi marcado com a anotação `@Id`.

Busca dos relacionamentos entre entidades. As anotações de relacionamento de cardinalidade da JPA são necessárias para a criação das chaves estrangeiras e das tabelas associativas do banco de dados. Na enumeração `DSRelationship` do pacote `schema:model` que foi ilustrada na Figura 30 da Seção 4.2.4, foram listados os possíveis tipos de relacionamentos que estão associados às anotações `@OneToOne`, `@OneToMany`, `@ManyToOne` e `@ManyToMany` da JPA. Em relação ao modelo UML apresentado na Figura 39, os atributos assinalados com essas anotações foram marcados respectivamente com os esteriótipos «`one-to-one`», «`one-to-many`», «`many-to-one`» e «`many-to-many`». Retornando ao exemplo da classe `Fisica`, pode-se observar na Figura 39 que sua superclasse `Cliente` contém o atributo `enderecos` que estabelece um relacionamento do tipo "um para muitos" com a classe `Endereco`. Isso significa que um cliente do tipo pessoa física pode possuir mais de um endereço cadastrado no sistema SisVenda. Nesse caso, deve ser criada uma chave estrangeira que aponta para um cliente na tabela que mapeia a classe `Endereco`. A partir desse exemplo, pode-se observar que as anotações de relacionamento possuem um papel relevante na criação do esquema de banco de dados da aplicação.

Rastreamento das definições de valores dos atributos. A JPA também fornece algumas anotações para indicar como os valores dos atributos das classes marcadas com a anotação `@Entity` devem ser tratadas. Como exemplo, na Figura 39 é possível observar que o atributo `cpf` da classe `Fisica` foi assinalado com o esteriótipo «`unique`». Isso indica que esse atributo foi marcado com a anotação `@Column(unique=true)`, indicando que só pode existir um único registro com um determinado número de CPF.

Conforme reportado na Seção 4.1.1, existe uma relação de dependência entre as classes das entidades de software do SaS e as tabelas contidas no seu banco de dados. Isso significa que pode existir uma possível correspondência estrutural entre as classes do SaS e seu esquema de banco de dados. Fazendo um paralelo com o modelo UML ilustrado na Figura 39, grande parte da informação estrutural das tabelas do banco de dados do sistema SisVenda pode ser extraída das classes contidas nesse modelo UML. Entretanto, as informações de persistência de dados fornecidas pelas anotações da JPA também fornecem os conhecimentos complementares sobre a estrutura dessas tabelas. Nesse sentido, pode-se dizer que a biblioteca DynaSchema deve considerar todas essas informações para apoiar a evolução do esquema de dados do SaS de acordo com as mudanças promovidas pelos seus ciclos de autoadaptação.

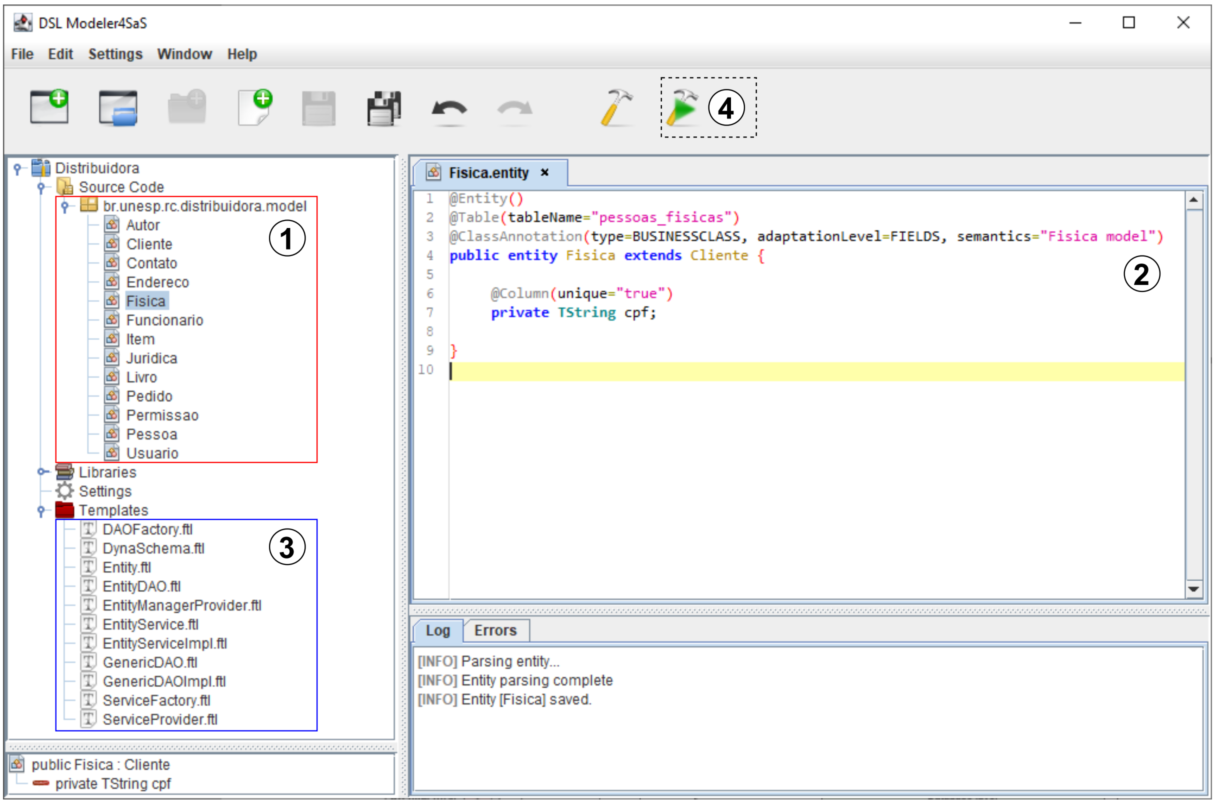
5.2 Implementação do Sistema Alvo

Baseado no modelo UML ilustrado na Figura 39, pode-se utilizar a ferramenta de modelagem DSLModeler4SaS para instanciar o sistema SisVenda de acordo com a arquitetura de referência RA4SaS (Seção 2.3). Na Figura 40 é ilustrado o projeto que foi criado na ferramenta DSLModeler4SaS para modelar o referido sistema, cujas entidades estão destacadas nessa figura através de uma caixa de linha vermelha **(1)**. Um exemplo de definição de uma dessas entidades pode ser visto na parte central da ferramenta **(2)**, onde está aberta a aba do editor de código fonte da entidade `Fisica`. Inicialmente, são declaradas duas anotações da JPA para marcar a classe dessa entidade (**linhas 1 e 2**). A anotação `@Entity` (**linha 1**) indica que a classe da entidade `Fisica` está relacionada com a funcionalidade de persistência de dados e a anotação `@Table` (**linha 2**) especifica o nome da tabela correspondente a essa classe no banco de dados. Além disso, a entidade `Fisica` também é marcada com a anotação `@ClassAnnotation` da RA4SaS (**linha 3**) para definir seu nível de adaptação. Em seguida, essa entidade é declarada como "filha" da superentidade `Cliente` (**linha 4**), cujo corpo de definição entre chaves (**linhas 4 a 9**) só especifica um único campo chamado `cpf` (**linha 7**). Tal campo foi marcado com a anotação `@Column` da JPA (**linha 6**) para indicar que seu valor deve ser único, assim não podem existir duas instâncias da entidade `Fisica` com o mesmo número de CPF. Após a declaração de todas as entidades, podem ser criados (ou importados para a ferramenta DSLModeler4SaS) os *templates* que serão utilizados gerar o código fonte do sistema SisVenda. Tais *templates* refletem as decisões iniciais de projeto e estabelecem a arquitetura do sistema, sendo destacados na Figura 40 através de uma caixa de linha azul **(3)**. Quando necessário, o botão de "construção" **(4)** pode ser pressionado para gerar o código do sistema (veja o processo ilustrado na Figura 6).

As classes do sistema SisVenda que foram geradas pela ferramenta DSLModeler4SaS podem ser vistas na Figura 41. Conforme discutido na Seção 2.3, pode-se observar como a DSLModeler4SaS facilitou o processo de instanciação da implementação desse sistema, uma vez que só foi necessária a definição das entidades de software da aplicação e dos *templates* de geração de código. Uma descrição de cada pacote do sistema SisVenda é apresentado a seguir:

Pacote `model`. Neste pacote estão contidas as classes de modelo para o sistema SisVenda. Como os detalhes desse modelo foram apresentados na Figura 40, suas classes correspondentes foram resumidas no diagrama da Figura 41 através da classe `Entity`. Nesse sentido, as classes cinzas que aparecem nesse diagrama devem ser entendidas como um *template* estrutural que deve ser aplicado para cada entidade do modelo de dados do sistema SisVenda (exceto para as entidades abstratas ou "embutidas" e para as enumerações), onde o termo "`Entity`" deve ser substituído pelo nome da entidade. Como exemplo, para a entidade `Fisica` devem existir as classes `Fisica` no pacote `model` e `FisicaDAO` no pacote `dao`, além das classes `FisicaService` e `FisicaServiceImpl` pertencentes ao pacote `service` do sistema SisVenda.

Figura 40 – Projeto do sistema SisVenda na ferramenta de modelagem DSLModeler4SaS

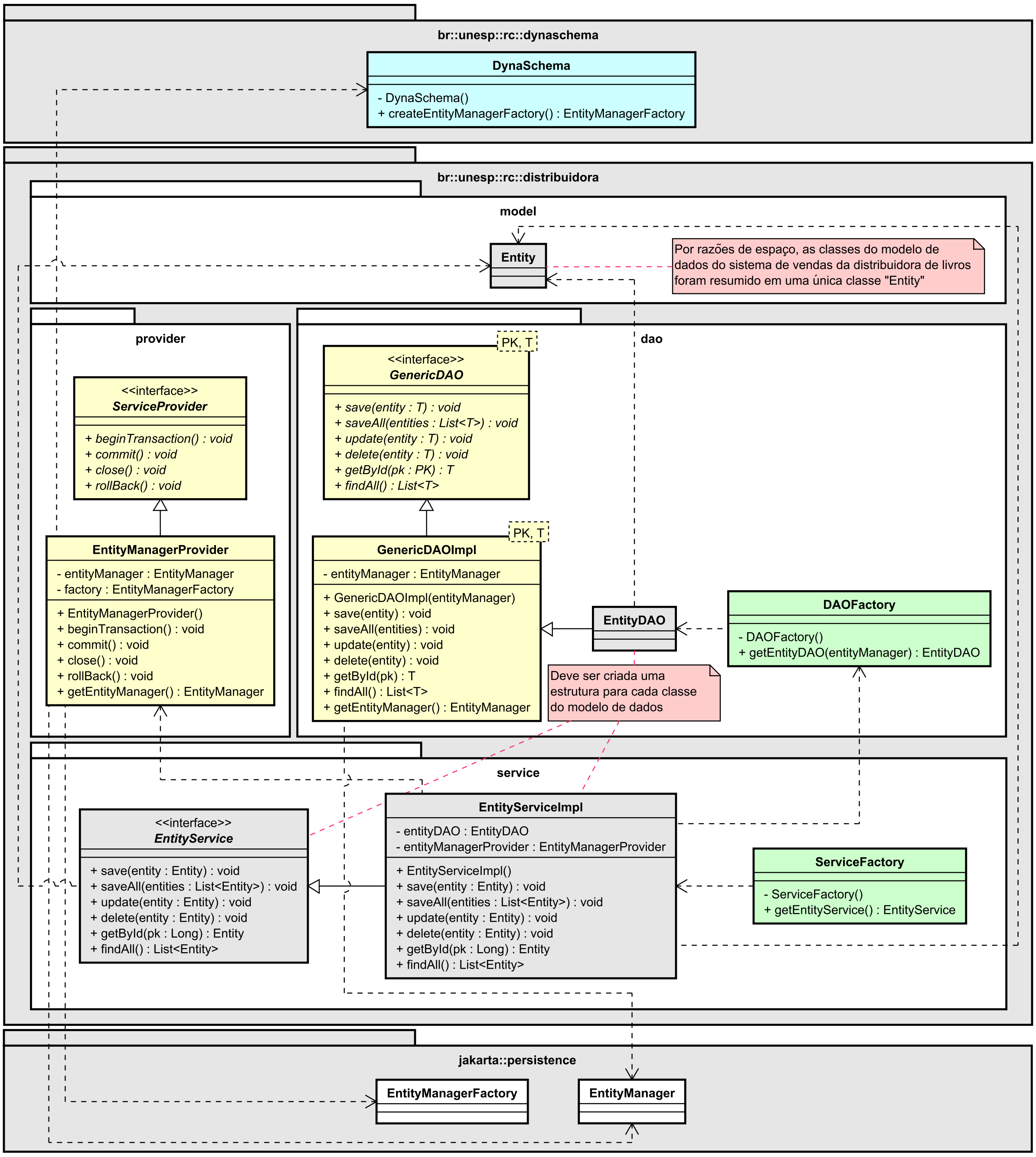


Fonte: Elaborada pelo autor.

Pacote `provider`. Neste pacote ficam armazenadas as classes que estabelecem o meio de manipulação do banco de dados. Nesse sentido, a interface `ServiceProvider` estabelece um padrão para criar um provedor de conexão com o banco de dados. A classe `EntityManagerProvider` é um exemplo de provedor concreto que utiliza a interface da biblioteca DynaSchema (ou seja, a classe `DynaSchema`) para obter uma instância com a interface `EntityManager`. Conforme reportado na Seção 4.2.5, as classes concretas que implementam essa interface são responsáveis por fornecer as funcionalidades de persistência de dados (como a inserção das informações de um objeto no banco de dados, por exemplo). No caso da biblioteca DynaSchema, é retornada uma instância da classe `DSEntityManager` (que está contida no pacote `jpa` do diagrama de classes da Figura 31) para a classe `EntityManagerProvider` do sistema SisVenda.

Pacote dao. As classes deste pacote estão relacionadas com os objetos de acesso a dados (em inglês, *<u>D</u>ata <u>A</u>ccess <u>O</u>bject* – DAO), os quais são responsáveis efetivamente pela execução das funcionalidades de persistência de dados. Nesse contexto, a interface `GenericDAO` define as funcionalidades de um objeto genérico de acesso a dados, como a inserção,

Figura 41 – Diagrama de classes do sistema SisVenda



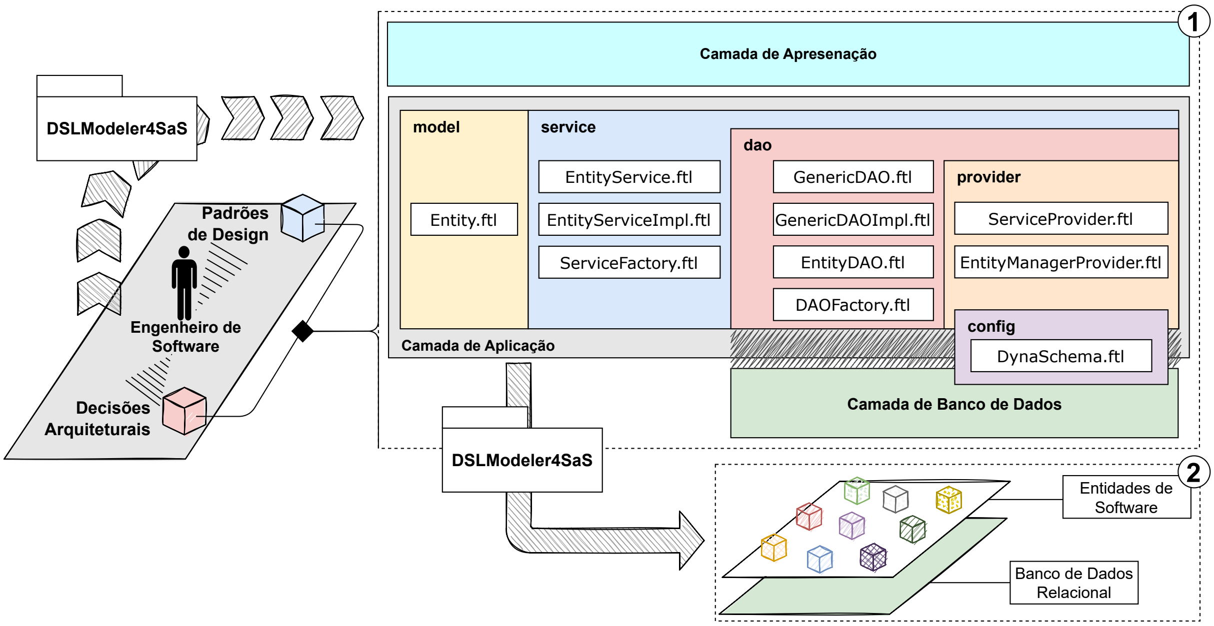
Fonte: Elaborada pelo autor.

atualização, remoção e recuperação dos dados de um objeto do banco de dados. A classe `GenericDAOImpl` fornece uma implementação padrão para essa interface, que é utiliza como base para criar as outras classes que modelam os objetos de acesso a dados do sistema SisVenda. Como exemplo, a classe `FisicaDAO` lida especificamente com a persistência de dados das instâncias da classe `Fisica` do modelo de dados desse sistema. Vale lembrar que a classe `EntityDAO`, representada na cor cinza, resume graficamente a existência das outras classes para lidar com a persistência das instâncias do modelo de dados no sistema SisVenda. Além disso, a classe `DAOFactory` fornece os métodos fábrica para instanciar os objetos de acesso a dados disponibilizados pelo pacote `dao` da aplicação.

Pacote `service`. As classes contidas neste pacote estão relacionadas com os serviços disponíveis no sistema SisVenda. Nesse sentido, tais serviços implementam a lógica de negócio da aplicação baseado no modelo de dados do sistema. Dentro deste pacote foi gerado uma estrutura de serviços básicos para as classes do modelo de dados desse sistema que fornecem somente as funcionalidades de persistência. Tal estrutura segue um padrão de uma interface e uma classe concreta para cada serviço, sendo representada na Figura 41 através da interface `EntityService` e da classe `EntityServiceImpl` que implementa essa interface. Como exemplo, para a classe `Fisica` do modelo de dados do sistema SisVenda é disponibilizado um serviço com a interface `FisicaService` e a classe concreta `FisicaServiceImpl`. Vale lembrar que essa organização foi estabelecida nos *templates* de geração de código fonte utilizados na ferramenta de modelagem DSLModeler4SaS. No cenário abordado por este estudo de caso, os serviços contidos na aplicação alvo são mais simples e oferecem somente as funcionalidades de persistência de dados. Além disso, no pacote `service` do sistema SisVenda está contida a classe `ServiceFactory` que fornece os métodos fábrica para instanciar os serviços da aplicação.

A ferramenta de modelagem DSLModeler4SaS fornece uma abordagem de desenvolvimento que permite projetar um SaS em um nível mais alto de abstração baseado em uma DSL (do inglês, *Domain-Specific Language*) chamada eLanguage (do inglês, *entity Language*). Nesse sentido, os engenheiros de software podem concentrar seus esforços em duas atividades principais, a saber: (i) a definição das entidades de software do SaS e seus níveis de adaptação; e (ii) a construção dos *templates* de geração de código fonte que estabelecem os padrões de *design* e as decisões arquiteturais da aplicação. Vale destacar que esses *templates* são criados de acordo com as necessidades de um determinado SaS e podem ser reutilizados em outros projetos semelhantes com possíveis ajustes. Na Figura 42 pode ser visto um esquema que ilustra o processo de geração do código fonte do sistema SisVenda no contexto da ferramenta de modelagem DSLModeler4SaS. De acordo com esse esquema, o engenheiro de software utiliza a DSLModeler4SaS para desenvolver o projeto do SaS **(1)**, cuja organização interna pode ser dividida nas camadas de apresentação, aplicação e banco de dados, por exemplo. De acordo com a Figura 42, os *temaplates* de geração de código fonte possuem a extensão `ftl` e estão agrupados em cada camada de aplicação, conforme a organização do pacotes no sistema SisVenda. Além disso, existe uma configuração entre as camadas de aplicação e de banco de dados, onde o *template* `DynaSchema.ftl` é responsável pela geração inicial de um arquivo XML de metadados com as informações de conexão com o banco de dados. Vale lembrar que esse arquivo é utilizado pela biblioteca DynaSchema para coordenar o funcionamento dos seus módulos internos. Conforme reportado na Seção 2.3, antes instanciar a implementação do SaS, a ferramenta DSLModeler4SaS transforma as entidades do modelo de dados da aplicação em metamodelos (Figura 7). Em seguida, esses metamodelos e os *templates* contidos na DSLModeler4SaS são envidados para o módulo de Código Fonte da arquitetura de referência RA4SaS, cuja funcionalidade é responsável pela geração do código fonte do sistema SisVenda de acordo com o processo que foi ilustrado na Figura 6. Como resultado, são geradas as entidades de software do SaS que estão preparadas para persistirem suas informações no banco de dados **(2)**.

Figura 42 – Geração do sistema SisVenda através da ferramenta de modelagem DSLModeler4SaS



Fonte: Elaborada pelo autor.

O arquivo XML de metadados que foi gerado pelo *template* `DynaSchema.ftl` da ferramenta de modelagem DSLModeler4SaS pode ser visto na Listagem 2. Dentro desse arquivo são declaradas cinco propriedades de propósito geral (contidas na *tag* `<properties>`) para configurar a biblioteca DynaSchema, a saber: (i) o nome do *driver* de banco de dados (**linha 4**); (ii) o endereço de conexão com o banco de dados (**linha 5**); (iii) o nome de usuário para a autenticação da conexão (**linha 6**); (iv) a senha desse usuário (**linha 7**); e (v) o nome da estratégia de dialeto SQL para interagir com o banco de dados (**linha 8**). Vale ressaltar que esse arquivo será manipulado posteriormente pela biblioteca DynaSchema para coordenar as funcionalidades fornecidas pelos seus módulos internos.

Listagem 2 – Arquivo XML de metadados gerado através da ferramenta de modelagem DSLModeler4SaS

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dynaschema>
    <properties>
        <property name="dynaschema.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
        <property name="dynaschema.jdbc.url"
            ↪ value="jdbc:mysql://localhost:3306/dynaschema"/>
        <property name="dynaschema.jdbc.user" value="user"/>
        <property name="dynaschema.jdbc.password" value="pass"/>
        <property name="dynaschema.dialect"
            ↪ value="br.unesp.rc.dynaschema.dialect.MariaDB"/>
    </properties>
</dynaschema>
```

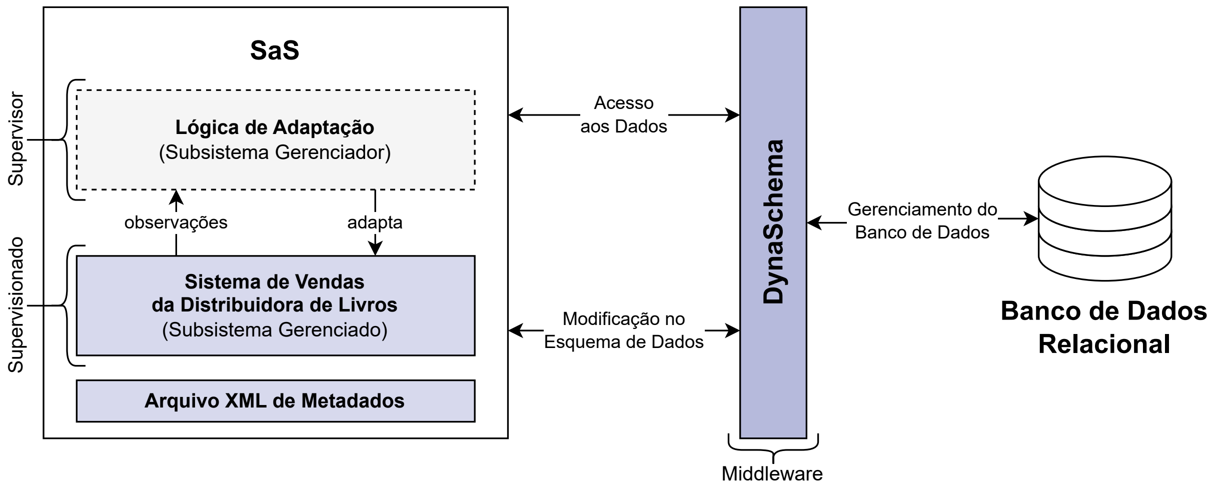
Fonte: Elaborada pelo autor

Em um cenário real de execução, a lógica de adaptação deve ser acoplada ao sistema SisVenda para que a aplicação seja um SaS completo, conforme ilustrado na Figura 43. Entretanto, este estudo de caso avaliou o processo de evolução do esquema de dados desse sistema alvo de maneira manual, fazendo com que as funcionalidades da biblioteca DynaSchema sejam apresentadas em um maior nível de detalhe e clareza. Por esse motivo, a lógica de adaptação foi representada na Figura 43 através de uma caixa de cor cinza com a borda tracejada. De acordo com a Seção 4.1.2, vale lembrar que a lógica de adaptação deve informar à biblioteca DynaSchema quais alterações foram feitas no esquema de dados da aplicação e quando cada uma das etapas do processo de evolução do esquema de dados do SaS pode ser executada.

5.3 Primeira Versão do Esquema de Dados do Sistema Alvo

Inicialmente, o sistema SisVenda não possui um esquema de dados rastreado pela biblioteca DynaSchema. Em resumo, isso significa que essa biblioteca não pode realizar as funcionalidades de persistência de dados, pois o esquema de dados da aplicação não é conhecido. Para isso, foi criada uma classe `Main` para criar a primeira versão do esquema de dados do sistema, cuja estrutura pode ser visualizada na Listagem 3. Vale destacar que o funcionamento geral dessa classe segue o mesmo comportamento da Listagem 1 da Seção 4.2.7. Nesse sentido, a classe `Main` implementa os métodos da interface `NotificationObserver` (**linha 1**) para responder aos eventos lançados pela biblioteca DynaSchema. Além disso, seu método `main()` (**linhas 3**

Figura 43 – Contexto de funcionamento do sistema SisVenda



Fonte: Elaborada pelo autor.

a 11) estabelece o "ponto de entrada" que será chamado quando a classe `Main` for executada. Dentro da declaração desse método, a classe `Main` se registra na biblioteca DynaSchema (**linha 5**), adiciona a entidade `Pedido` no esquema de dados da aplicação (**linha 7**), salva as alterações que foram feitas nesse esquema (**linha 8**) e solicita a primeira migração do esquema de dados (**linha 10**) para o sistema SisVenda.

Listagem 3 – Criação da primeira versão do esquema de dados do sistema SisVenda

```
1  public class Main implements NotificationObserver {
2
3      public static void main(String[] args) {
4          // Configuração de notificação
5          DynaSchema.attachObserver(this);
6          // Alteração do esquema
7          DynaSchema.addEntity(Pedido.class);
8          DynaSchema.saveState();
9          // Migração do esquema
10         DynaSchema.migrateSchema();
11     }
12
13     public void onSchemaMigrationEnd() {
14         // Migração dos dados
15         DynaSchema.migrateData();
16     }
17
18     public void onDataMigrationEnd() {
19         // Troca do esquema
20         DynaSchema.switchSchema();
21     }
22
23     public void onSchemaSwitchEnd() {
```

```
24         // Declaração dos serviços
25         EnderecoService enderecoService = ServiceFactory.getEnderecoService();
26         FisicaService fisicaService = ServiceFactory.getFisicaService();
27         // Informações de endereço
28         Endereco endereco = new Endereco();
29         endereco.setDescricao("Endereço Principal");
30         endereco.setCep("13506700");
31         endereco.setEstado("SP");
32         endereco.setCidade("São Paulo");
33         endereco.setBairro("Bela Vista");
34         endereco.setRua("Avenida 24 A");
35         endereco.setNumero(808);
36         endereco.setComplemento("Casa");
37         // Dados pessoais do cliente
38         Fisica fisica = new Fisica();
39         fisica.setNome("José");
40         fisica.getContato().setTelefone("1129345281");
41         fisica.getContato().setEmail("jose@gmail.com");
42         fisica.setCpf("10352471867");
43         fisica.addEndereco(endereco);
44         // Persistência de dados
45         fisicaService.save(fisica);
46         enderecoService.save(endereco);
47     }
48
49     // Os outros métodos de interceptação das notificações foram omitidos ...
50
51 }
```

Fonte: Elaborada pelo autor

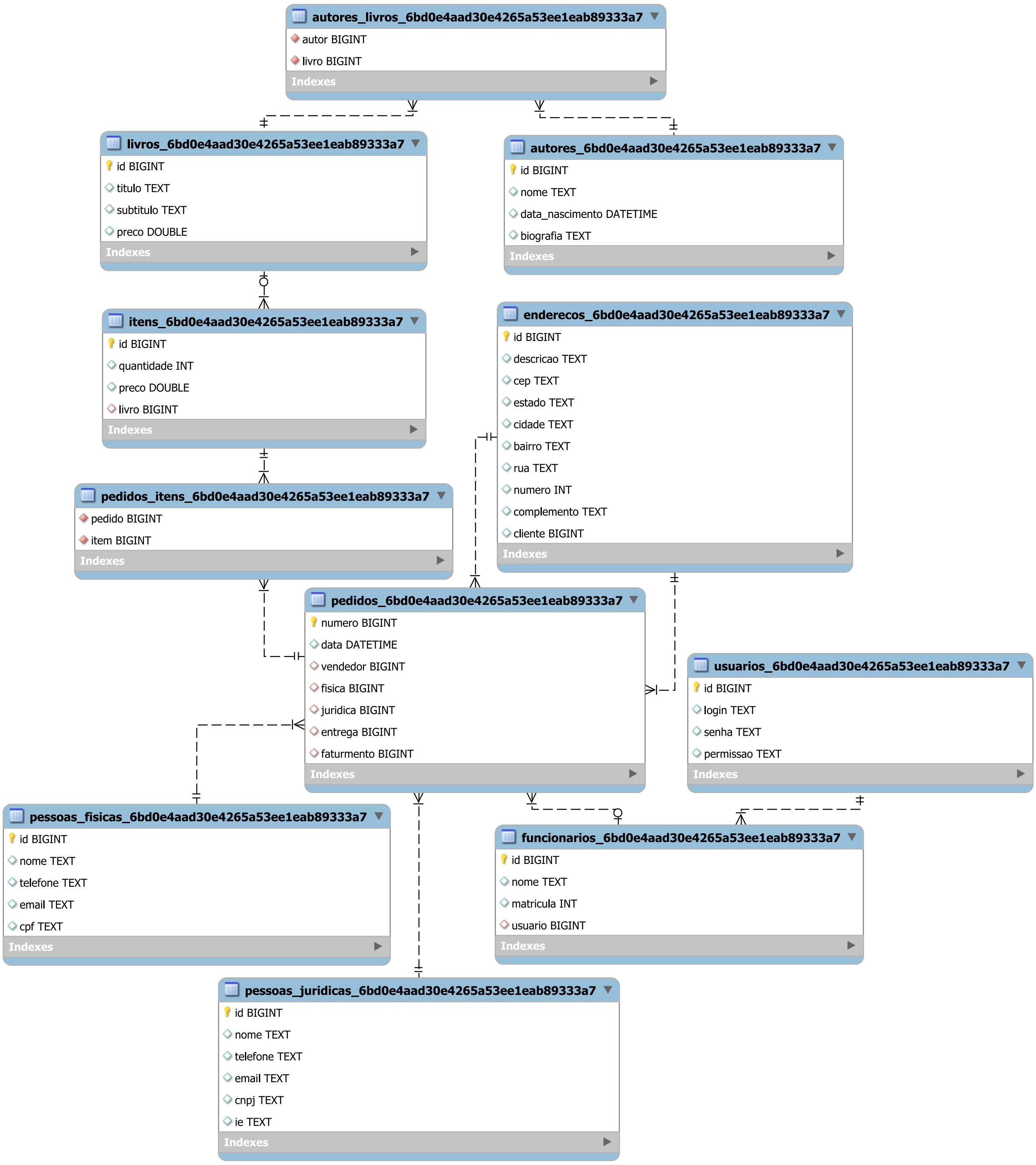
No caso abordado na Listagem 3, só foi necessário adicionar a entidade `Pedido` no esquema de dados do sistema SisVenda, pois a biblioteca DynaSchema é capaz de rastear todas a entidades que dependem direta ou indiretamente da entidade `Pedido`. Para isso, as dependências dessa entidade são localizadas de maneira recursiva nos tipos dos seus atributos, na sua superclasse, nos tipos dos atributos dessa superclasse e assim sucessivamente. De acordo com o modelo UML ilustrado na Figura 39, pode-se chegar as outras entidades desse modelo a partir da entidade `Pedido` através dos relacionamentos entre entidades e/ou das hierarquias de herança. Quando a migração do esquema for concluída, o método `onSchemaMigrationEnd()` da classe `Main` (**linhas 13 a 16**) será chamado, fazendo com que seja solicitada a migração dos dados da aplicação (**linha 15**). Em seguida, quando essa classe interceptar a mensagem de finalização da migração dos dados, será chamado o método `onDataMigrationEnd()` (**linhas 18 a 21**) que é responsável por solicitar a troca do esquema da aplicação (**linha 20**).

Ao final do processo da troca de esquema do sistema SisVenda, a biblioteca lançará uma notificação que será interceptada pelo método `onSchemaSwitchEnd()` da classe `Main` (**linhas 23 a 47**). Dentro do corpo desse método está contido um exemplo de inserção dos dados de um cliente do tipo pessoa física no banco de dados. Inicialmente são declarados os dois serviços para lidar com a persistência de dados dos endereços (**linha 25**) e dos clientes do tipo pessoa física (**linha 26**). Em seguida, são criados dois objetos com as informações de endereço (**linhas 28 a 36**) e com os dados pessoais do cliente (**linhas 38 a 43**). Finalmente, os objetos do endereço e do cliente são persistidos no banco de dados (**linhas 45 e 46**) através dos serviços declarados no início do método. De acordo com a estrutura de classes ilustrada na Figura 41, vale lembrar que as instâncias dos serviços podem ser obtidas através dos métodos fábricas da classe `ServiceFactory` que expõem somente a interface desses serviços. Internamente, cada método fábrica é responsável por retorna uma instância da classe que implementa a interface do serviço correspondente a esse método.

Na Figura 44 podem ser visualizadas as tabelas que foram criadas no banco de dados pela biblioteca DynaSchema de acordo com o modelo UML ilustrado na Figura 39. Conforme apresentado na Seção 5.1, essa biblioteca deve se basear tanto na estrutura do modelo de dados da aplicação quanto nas informações de persistência desse modelo para criar o esquema de banco de dados de maneira correta. Vale ressaltar que essas informações de persistência são fornecidas pelas anotações da JPA que foram utilizadas para marcar a estrutura das classes do modelo de dados do sistema SisVenda. Com relação ao nome das tabelas, pode-se observar na Figura 44 que foi utilizado um padrão de nomenclatura, onde o identificador do plano de migração (veja a classe `Migration` do pacote `model` que foi apresentada na Figura 33) é utilizado como sufixo. Tal padrão de nomenclatura foi adotado para evitar a criação de tabelas com o mesmo nome e para identificar o plano de migração que deu origem a essas tabelas.

A partir do exemplo de execução apresentado na Listagem 3 foram exploradas quatro funcionalidades da biblioteca DynaSchema. Em relação à funcionalidade **F1**, a classe `DynaSchema` dessa biblioteca foi utilizada como um *middleware* entre o sistema SisVenda e seu banco de dados relacional, constituindo uma camada intermediária responsável pelo gerenciamento da evolução do esquema de dados da aplicação e pelo acesso aos dados contidos no banco. Na Listagem 3 foram abordadas as solicitações para a migração dos dados e a troca da versão de esquema da aplicação, cujas atividades estão relacionadas respectivamente com as funcionalidades **F2** e **F6** da biblioteca DynaSchema. Além disso, também foi possível observar um exemplo de utilização dos métodos que "escutam" as notificações enviadas por essa biblioteca, estabe-

Figura 44 – Primeira versão do esquema de banco de dados para o sistema SisVenda



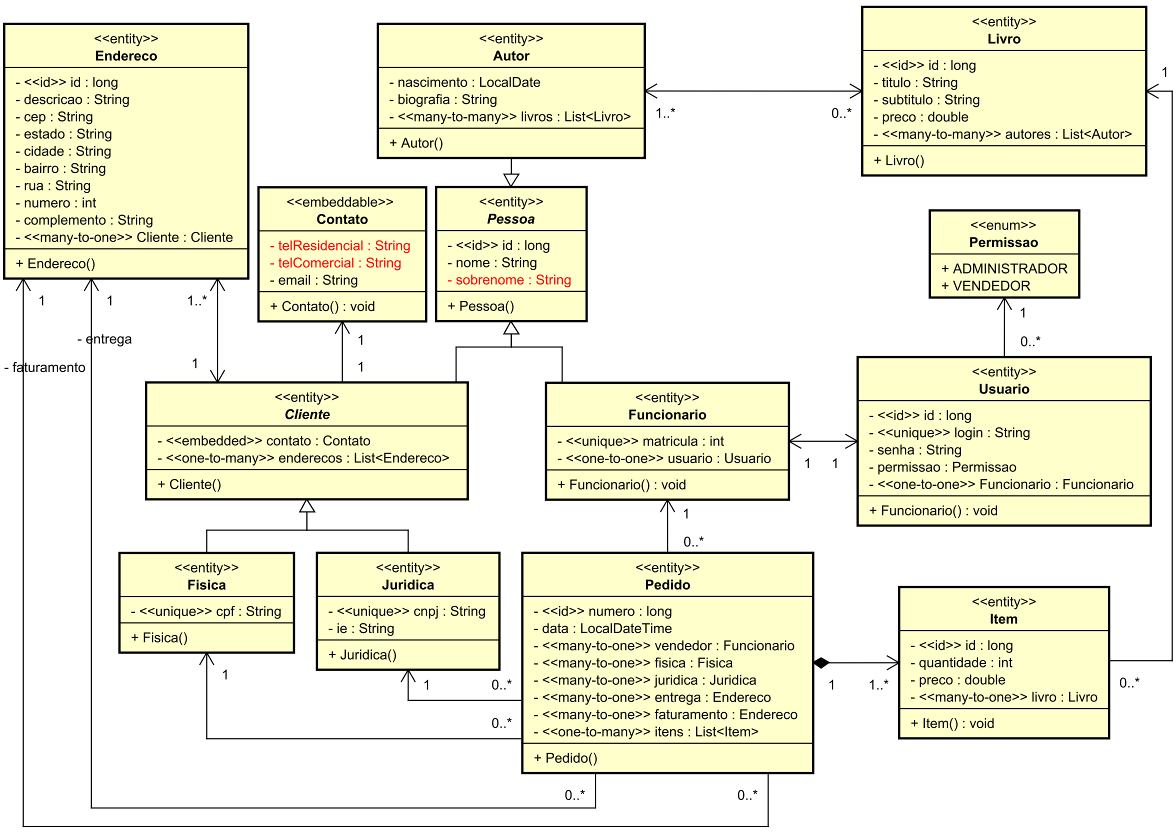
Fonte: Elaborada pelo autor.

lecendo um mecanismo assíncrono para a execução de outras tarefas da classe `Main`. Por fim, na Listagem 2 foi reportada a maneira de especificar a estratégia de dialeto SQL utilizada pela biblioteca DynaSchema, cuja configuração é aplicada no exemplo de execução da Listagem 3 e está relacionada com a funcionalidade **F5** dessa biblioteca.

5.4 Alteração do Esquema de Dados do Sistema Alvo

Para simular um ciclo de autoadaptação, foram feitas duas mudanças no modelo de dados do sistema SisVenda. A nova versão do modelo UML desse sistema pode ser visto na Figura 45, cujas alterações estão destacadas em vermelho. De acordo com as novas mudanças, o atributo `telefone` da classe `Contato` foi removido e em seu lugar foram adicionados os atributos `telResidencial` e `telComercial`. Tais atributos foram incluídos para criar duas categorias de telefone, sendo que o atributo `telResidencial` representa um número residencial e o atributo `telComercial` armazena um outro número de contato do tipo comercial. Além disso, na classe `Pessoa` foi adicionado o atributo `sobrenome` para permitir que o sistema SisVenda possa armazenar o nome completo dos seus funcionários, clientes e autores de livros.

Figura 45 – Segunda versão do modelo UML do sistema SisVenda



Fonte: Elaborada pelo autor.

Dentro da ferramenta DSLModeler4SaS, o modelo de dados do sistema SisVenda foi alterado de acordo com a Figura 45 e seu código fonte foi gerado novamente. Como resultado, esse sistema possui a mesma configuração geral da arquitetura ilustrada na Figura 41, porém as classes do seu modelo de dados estão de acordo com a nova versão estrutural que foi apresentada na Figura 45. Para executar essa mudança no esquema de dados da aplicação, foram feitas algumas alterações na classe `Main`, cuja nova estrutura pode ser vista na Listagem 4.

Listagem 4 – Alteração do esquema de dados do sistema SisVenda

```java
public class Main implements NotificationObserver {

    public static void main(String[] args) {
        // Configuração de notificação
        DynaSchema.attachObserver(this);
        // Alteração do esquema
        DynaSchema.removeEntityField(Contato.class, "telefone");
        DSField telResidencial = new DSField();
        telResidencial.setName("telResidencial");
        telResidencial.setType(String.class.getCanonicalName());
        telResidencial.setColumnName("tel_residencial");
        DynaSchema.addEntityField(Contato.class, telResidencial);
        DSField telComercial = new DSField();
        telComercial.setName("telComercial");
        telComercial.setType(String.class.getCanonicalName());
        telComercial.setColumnName("tel_comercial");
        DynaSchema.addEntityField(Contato.class, telComercial);
        FieldMapping telResidencialMapping = new FieldMapping(Contato.class, "telResidencial");
        telResidencialMapping.setSourceField("telefone");
        DynaSchema.addFieldMapping(telResidencialMapping);
        DSField sobrenome = new DSField();
        sobrenome.setName("sobrenome");
        sobrenome.setType(String.class.getCanonicalName());
        sobrenome.setColumnName("sobrenome");
        DynaSchema.addEntityField(Pessoa.class, sobrenome);
        DynaSchema.saveState();
        // Migração do esquema
        DynaSchema.migrateSchema();
    }

    public void onSchemaMigrationEnd() {
        // Migração dos dados
        DynaSchema.migrateData();
    }

    public void onDataMigrationEnd() {
        // Configuração para utilizar o esquema candidato
        DynaSchema.setProperty(StateProperties.DYNASCHEMA_USE_CANDIDATE, true);
        // Declaração dos serviços
        EnderecoService enderecoService = ServiceFactory.getEnderecoService();
        FisicaService fisicaService = ServiceFactory.getFisicaService();
        // Informações de endereço
        Endereco endereco = new Endereco();
        endereco.setDescricao("Endereço de Casa");
        endereco.setCep("13506723");
```

```
46          endereco.setEstado("SP");
47          endereco.setCidade("São Paulo");
48          endereco.setBairro("Bela Vista");
49          endereco.setRua("Avenida 22 A");
50          endereco.setNumero(554);
51          endereco.setComplemento("Casa");
52          // Dados pessoais do cliente
53          Fisica fisica = new Fisica();
54          fisica.setNome("Maria");
55          fisica.setSobrenome("Carvalho");
56          fisica.getContato().setTelResidencial("1136212333");
57          fisica.getContato().setTelComercial("1127563934");
58          fisica.getContato().setEmail("maria@gmail.com");
59          fisica.setCpf("50938635808");
60          fisica.addEndereco(endereco);
61          // Persistência de dados
62          fisicaService.save(fisica);
63          enderecoService.save(endereco);
64          // Troca do esquema
65          DynaSchema.switchSchema();
66      }
67
68      // Os outros métodos de interceptação das notificações foram omitidos ...
69
70  }
```

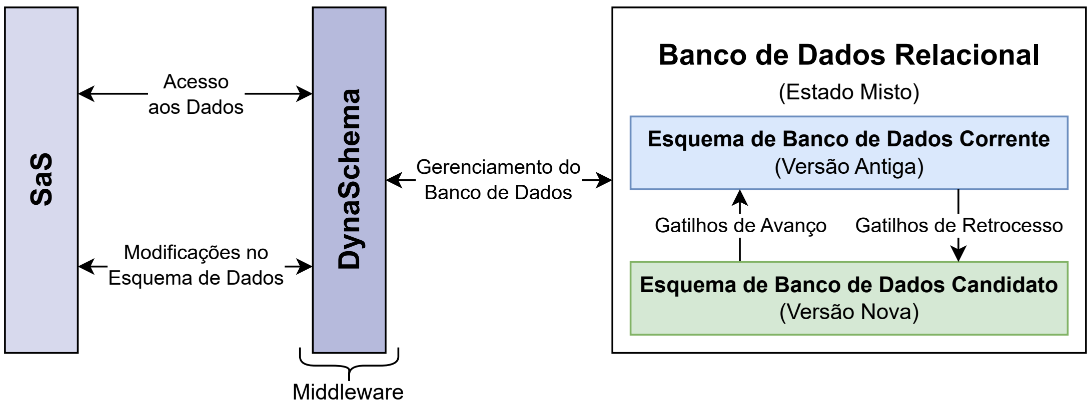
Fonte: Elaborada pelo autor

A **primeira mudança** foi feita no corpo do método `main()` da classe `Main`. Na seção da alteração do esquema (**linhas 6 a 26**) foram declarados os comandos para modificar o esquema de dados da aplicação de acordo com o novo modelo UML ilustrado na Figura 45. Para a classe `Contato`, foi removido o atributo `telefone` (**linha 7**), foram adicionados os novos atributos `telResidencial` (**linhas 8 a 12**) e `telComercial` (**linhas 13 a 17**) e foi estabelecido um mapeamento de dados do atributo `telefone` da versão antiga da classe `Contato` para o atributo `telResidencial` da nova versão dessa mesma classe (**linhas 18 a 20**). Por outro lado, na classe `Pessoa` foi adicionado o atributo `sobrenome` (**linhas 21 a 25**).

A **segunda mudança** foi feita no método `onDataMigrationEnd()`, cuja execução é chamada quando a biblioteca DynaSchema envia a notificação de finalização do processo de migração dos dados. Na definição desse método, antes solicitar a troca da versão do esquema de dados da aplicação (**linha 65**), a biblioteca é configurada para utilizar o mapeamento contido no esquema de dados candidato (**linha 38**). Em seguida, é adicionado um novo cliente no sistema SisVenda (**linhas 39 a 63**) para testar a persistência de dados em uma condição de estado misto. Vale destacar que nesse estado as versões antiga e nova do esquema da aplicação (ou seja, os esquemas corrente e candidato, respectivamente) coexistem ao mesmo tempo. Assim, quando a versão antiga do esquema de dados não for mais utilizada pelo sistema SisVenda, pode-se solicitar a troca do esquema da aplicação e a remoção do esquema antigo.

A biblioteca DynaSchema foi projetada com base na abordagem criada por Jong, Deursen e Cleve (2017) para permitir que um SaS acesse seus dados em uma condição de estado misto. De acordo com o esquema ilustrado na Figura 46, dentro do banco de dados são mantidas duas versões estruturais diferentes do esquema de dados do SaS de maneira "virtual". Nesse sentido, um conjunto de tabelas com o mesmo sufixo (ou seja, o identificador de um plano de migração) corresponde a um esquema de banco de dados "virtual". Nesse sentido, quando o SaS executa o método `migrateSchema()` da classe `DynaSchema` fornecida pela biblioteca DynaSchema, são criadas as novas tabelas para armazenar os dados do esquema candidato (ou seja, a nova versão) ao mesmo tempo que é mantido o conjunto antigo de tabelas que está relacionado com o esquema corrente (ou seja, a versão antiga).

Figura 46 – Esquema de banco de dados no estado misto



Fonte: Elaborada pelo autor.

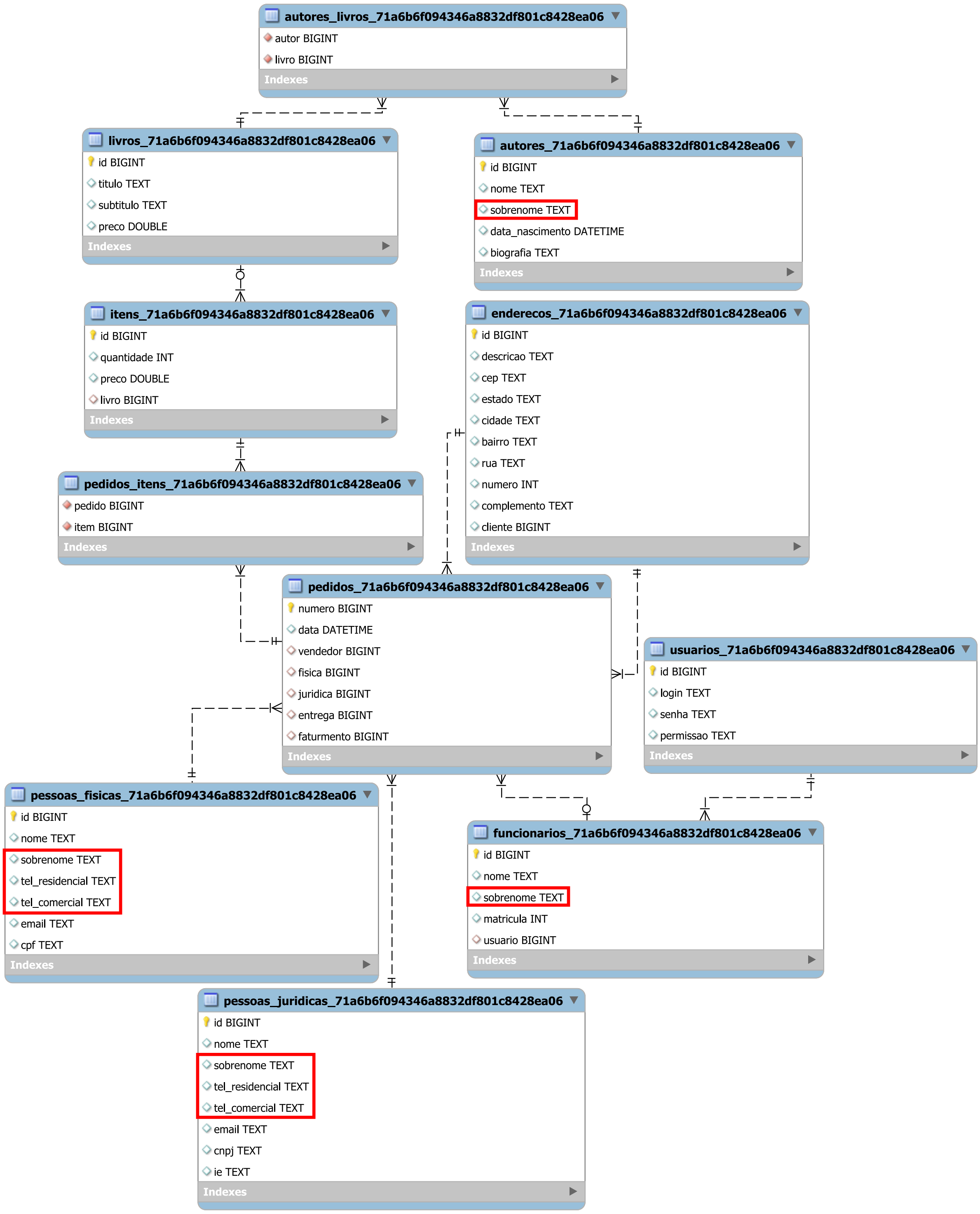
Além disso, essas tabelas possuem gatilhos (ou seja, *triggers*) especiais que replicam as ações de inserção, alteração e exclusão em uma tabela da versão antiga do esquema de banco de dados na sua tabela correspondente na nova versão. Tais gatilhos devem lidar com a diferença estrutural entre as tabelas para replicar os registros de maneira correta. Através desse mecanismo de múltiplas versões de tabelas com gatilhos de replicação entre si, pode-se manter os dados do SaS em um estado estrutural misto. Quando a versão antiga do esquema de banco de dados deixa de ser utilizada pelo SaS, suas tabelas podem ser excluídas e somente a versão mais nova do esquema de banco de dados precisa ser mantida.

Vale lembrar que a consulta dos dados na nova versão do esquema do SaS só pode ser executada com segurança após a chamada do `migrateData()` da classe `DynaSchema` fornecida pela biblioteca DynaSchema, pois os registros só são replicados à medida que são modificados e não é possível garantir que o SaS tenha interagido com todos os registros do banco de dados antes da operação de consulta. A bordagem criada por Jong, Deursen e Cleve (2017) foi empregada na biblioteca DynaSchema pela possibilidade de manter o estado misto dos dados do SaS em função dos recursos fornecidos pelo próprio banco de dados relacional (ou seja, através das tabelas e dos gatilhos armazenados no banco de dados).

A segunda versão do esquema de banco de dados do sistema SisVenda pode ser visualizada na Figura 47. As caixas de linha vermelha destacam as mudanças que foram feitas na estrutura das tabelas desse esquema, conforme o novo modelo UML que foi apresentado na Figura 45. A primeira alteração foi feita na classe `Contato`, cujo atributo `telefone` foi removido e no seu lugar foram adicionados os atributos `telResidencial` e `telResidencial`. Tal alteração resultou na modificação das tabelas `pessoas_fisicas_71a6b6f094346a8832df801c8428ea06` e `pessoas_juridicas_71a6b6f094346a8832df801c8428ea06` onde o campo `telefone` foi substituído pelos campos `tel_residencial` e `tel_comercial`. Conforme especificado na Listagem 4, vale lembar que os valores do campo `telefone` dessas tabelas foram inseridos no novo campo `tel_residencial`. A segunda alteração do modelo de dados do sistema SisVenda foi feita na classe `Pessoa`, onde foi adicionado o atributo `sobrenome` para complementar a informação do nome dos clientes, funcionário e autores de livro. Essa alteração resultou na inserção do campo `sobrenome` nas tabelas `pessoas_fisicas_71a6b6f094346a8832df801c8428ea06`, `pessoas_juridicas_71a6b6f094346a8832df801c8428ea06`, `funcionarios_71a6b6f094346a8832df801c8428ea06` e `autores_71a6b6f094346a8832df801c8428ea06`.

Os dados dos clientes do tipo pessoa física podem ser vistos na Figura 48 em uma condição de estado misto. Esse cenário representa o instante de tempo em que a classe `Main` da Listagem 4 ainda não executou a troca do esquema de dados utilizado pelo sistema SisVenda. A tabela contida na parte superior da Figura 48 possui a versão estrutural da primeira versão do esquema de banco de dados que foi ilustrada na Figura 44. Nesse caso, essa tabela apresenta a coluna `telefone` e o registro do cliente "`José`" foi inserido pela classe `Main` da Listagem 3. Em relação a tabela localizada na parte inferior da Figura 48, sua estrutura está de acordo com a segunda versão do esquema de banco de dados que foi apresentada na Figura 47. Nessa nova versão, a coluna `telefone` foi removida e no seu lugar foram adicionadas as colunas `tel_residencial` e `tel_comercial`. De acordo com a ordem de execução dos métodos da

Figura 47 – Segunda versão do esquema de banco de dados para o sistema SisVenda



Fonte: Elaborada pelo autor.

classe `Main` da Listagem 4, inicialmente foi inserido o registro da cliente "`Maria`" com seu respectivo sobrenome. Durante essa inserção, o gatilho de replicação interceptou essa mudança e também inseriu o registro da cliente "`Maria`" na tabela antiga. Em seguida, foi solicitada a migração dos dados da aplicação, fazendo com que o registro do cliente "`José`" fosse migrado para a tabela com a versão estrutural mais nova. A partir deste ponto, as tabelas com os dados dos clientes do tipo pessoa física estão sincronizadas e seus gatilhos serão responsáveis pela replicação dos dados entre as tabelas antiga e nova.

Figura 48 – Dados da tabela de clientes do tipo pessoa física em estado misto

pessoas_fisicas_6bd0e4aad30e4265a53ee1eab89333a7

id	nome	telefone	email	cpf
1	José	1129345281	jose@gmail.com	10352471867
2	Maria	1136212333	maria@gmail.com	50938635808

pessoas_fisicas_71a6b6f094346a8832df801c8428ea06

id	nome	sobrenome	tel_residencial	tel_comercial	email	cpf
1	José	*NULL*	1129345281	*NULL*	jose@gmail.com	10352471867
2	Maria	Carvalho	1136212333	1127563934	maria@gmail.com	50938635808

Fonte: Elaborada pelo autor.

No exemplo de execução abordado na Listagem 4 foram exploradas mais duas funcionalidades da biblioteca DynaSchema em relação a Listagem 3. Nesse cenário foi apresentada a inserção dos dados de clientes do tipo pessoa física em uma condição de estado misto, onde o acesso e a manutenção dos dados nessa condição estão relacionados com a funcionalidade **F3** da biblioteca. Na Listagem 4 também foi especificado um mapeamento de dados entre o atributo `telefone`, removido da classe `Contato`, e o atributo `telResidencial`, que foi inserido no seu lugar. Esse tipo de recurso está relacionado com a funcionalidade **F4** da biblioteca DynaSchema. Além disso, em relação a funcionalidade **F6**, pode ser testada a remoção do antigo esquema de banco de dados após a troca do esquema do sistema SisVenda para sua versão mais nova.

5.5 Discussão de Resultados

Neste capítulo foi apresentado um estudo de caso para avaliar o comportamento biblioteca DynaSchema em relação aos requisitos estabelecidos na Seção 4.1 e aos cenários de execução contidos neste estudo de caso. Para isso, nas Seções 5.1 e 5.2 foi utilizado um sistema alvo fictício chamado SisVenda que adota essa biblioteca como um meio para evoluir seu esquema de dados. Na Seção 5.2 foi apresentado o cenário da primeira versão do esquema do sistema SisVenda, onde deve ser completado um ciclo de migração completa (ou seja, a migração do esquema e dos dados, além da troca de esquema) para permitir que os dados da aplicação possam ser persistidos. Em seguida, na Seção 5.3 foi explorado o cenário da alteração do esquema de dados do sistema SisVenda, onde foi apresentado um estado transitório que contempla um contexto de execução que utiliza simultaneamente as versões (antiga e nova) do esquema de dados. A partir desses cenários, foi possível observar que o comportamento da biblioteca DynaSchema está de acordo com as funcionalidades propostas nas suas especificações de projeto que foram abordadas na Seção 4.1.3.

Na Figura 49 é ilustrado um quadro comparativo que visa estabelecer um paralelo entre a biblioteca DynaSchema e o SMS elaborado no Capítulo 3. Para isso, tal quadro utiliza as QPs levantadas no SMS para comparar a biblioteca DynaSchema com os estudos mais completos desse SMS (ou seja, aqueles que responderam a todas as QPs do SMS). De acordo com esse quadro comparativo, um resumo do relacionamento entre as QPs elaboradas nesse SMS e a biblioteca DynaSchema desenvolvida neste projeto de mestrado é apresentado a seguir:

Figura 49 – Quadro comprativo entre a biblioteca DynaSchema e os estudos mais completos do SMS

Estudos	**QP2 Solução**	**QP3 Domínio**	**QP4 Modelo de Dados**	**QP5 Apresenta Etapa da Abordagem de Evolução**		**QP6 Apresenta Atributo Não Funcional**	**QP7 Apresenta Método de Avaliação**
				Migração do Esquema	**Migração dos Dados**		
E05	Framework	Tradução de Modelos	Orientado a Objetos	✓	✓	✓	✓
E10	SGBD	Sistema de Banco de Dados	Banco de Dados	✓	✓	✓	✓
E11	Ferramenta	Implementação de Software	Orientado a Objetos	✓	✓	✓	✓
E15	Middleware	Evolução do Banco de Dados	Banco de Dados	✓	✓	✓	✓
E17	Middleware	Implementação de Software	Banco de Dados	✓	✓	✓	✓
Dyna-Schema	Middleware	Implementação de Software	Banco de Dados e Objeto-Relacional	✓	✓	✓	✓

Fonte: Elaborada pelo autor.

- **QP2:** a biblioteca DynaSchema é uma solução para a evolução do esquema de dados do SaS do tipo *middleware* (Seção 4.1). Nesse sentido, tal biblioteca atua como uma camada intermediária entre o SaS e seu banco de dados, fornecendo dois conjuntos principais de funcionalidade (Seção 4.1.2), a saber: (i) *a persistência de dados no banco relacional* e (ii) *a evolução do esquema de dados da aplicação*;

- **QP3:** a biblioteca DynaSchema utiliza uma abordagem de implementação de software para apoiar o SaS. Para isso, suas funcionalidades são empacotadas em uma biblioteca escrita na linguagem Java. Assim, o projeto de um SaS deve importar essa biblioteca para acessar suas funcionalidades de persistência de dados e evolução do esquema de dados;

- **QP4:** o processo de evolução do esquema de dados do SaS está relacionado com dois tipos de esquema (Seção 4.1.3). O primeiro tipo aborda o esquema orientado a objetos das classes que modelam a estrutura do SaS, já o segundo tipo trata do esquema de banco de dados que especifica a estrutura das tabelas que armazenam os dados desse SaS. Nesse sentido, a biblioteca DynaSchema trabalha com dois tipos de modelos de dados, a saber: (i) *o modelo de banco de dados*, que está associado às operações para evoluir o esquema de banco de dados; e (ii) *o modelo objeto-relacional*, que está ligado com a abordagem de mapeamento objeto relacional da JPA que foi implementada pela biblioteca DynaSchema para lidar com a persistência de dados;

- **QP5:** a biblioteca DynaSchema divide o processo de evolução do esquema de dados em etapas (Seção 4.1.2), permitindo que o SaS escolha o melhor momento para executá-las. Essas etapas estão relacionadas com a abordagem reportada na Seção 3.1.5, que inclui as atividades de migração do esquema do banco de dados e dos dados nele contido;

- **QP6:** o projeto da biblioteca DynaSchema se preocupou em fornecer uma abordagem baseada no versionamento do esquema de dados do SaS ao longo do tempo que evita a pertubação de execução do SaS (ou seja, evitando afetar no desempenho e no tempo de inatividade da aplicação). Isso faz com que o seu funcionamento interno tenha mecanismos de proteção contra falhas no processo de evolução do esquema de dados do SaS, além de fornecer uma interface simples/transparente ao usuário final; e

- **QP7:** os requisitos de projeto, as funcionalidades e o comportamento da biblioteca DynaSchema foram avaliados através de um estudo de caso apresentado neste capítulo, sendo uma técnica de avaliação das soluções propostas que foi identificada do SMS.

Em relação às soluções para a evolução do esquema de dados do SaS apresentadas no quadro comparativo da Figura 49, podem ser observadas algumas tendências para a elaboração de soluções do tipo *middleware* (**QP2**) e com domínio de aplicação relacionado a implementação de software (**QP3**). Tais tendências possuem uma predisposição mais natural, pois o *middleware* encapsula as funcionalidades que não são fornecidas nativamente pelo sistema alvo (Seção 3.1.2) e as provas de conceito iniciais geralmente podem ser implementadas de maneira mais simples e rápida via software. Os estudos listados na Figura 49 apresentam uma solução que só aborda um aspecto da evolução do esquema de dados do SaS. Nesse sentido, tais soluções lidam com as mudanças dentro da implementação do SaS (ou seja, no modelo orientado a objetos) ou dentro do banco de dados (ou seja, no modelo de banco de dados) (**QP4**). Por outro lado, a biblioteca DynaSchema lida com as mudanças que foram feitas tanto na estrutura do esquema de dados do SaS quanto no esquema de banco de dados. Além disso, todas as soluções contidas no quadro ilustrado na Figura 49 apresentaram uma abordagem de evolução que engloba as atividades de migração do esquema e migração dos dados da aplicação (**QP5**), estabelecem um ou mais atributos não funcionais no seu projeto de desenvolvimento (**QP6**) e utilizam alguma metodologia para avaliar/validar seus requisitos de projeto (**QP7**).

Com relação aos 17 estudos selecionados no SMS apresentado no Capítulo 3, vale destacar que a biblioteca DynaSchema é a primeira iniciativa efetiva para atuar como uma solução projetada especificamente para atender as necessidades do domínio de SaS. Por fim, podem ser destacados quatro aspectos importantes sobre o comportamento da biblioteca DynaSchema no estudo de caso elaborado neste capítulo, a saber:

Funcionamento baseado em modelos. As funcionalidades de evolução do esquema de banco de dados são executadas pela biblioteca DynaSchema com base em dois modelos principais. O primeiro modelo está relacionado com a representação conceitual do esquema de dados do SaS (tanto da versão corrente quanto da versão candidata), cujas classes estão contidas no pacote `schema::model` do módulo Gerenciador de Esquema (veja o diagrama da Figura 30). Por outro lado, o segundo modelo estabelece a representação conceitual do plano de migração que contém as informações para executar a evolução do esquema de banco de dados, cujas classes estão reunidas no pacote `migration::model` do módulo Gerenciador de Migração (veja o diagrama da Figura 33).

As informações contidas nesses modelos estabelecem a lógica da atividade de evolução do esquema de banco de dados que é executada pela biblioteca DynaSchema. Nesse sentido, o SaS pode modificar as informações desses modelos em tempo de execução, permitindo que a biblioteca possa modificar as tabelas do banco de dados de acordo com os ciclos de autoadaptação estrutural desse SaS. Para isso, a biblioteca DynaSchema utiliza o primeiro modelo para migrar o esquema de banco de dados (ou seja, para criar as tabelas da nova versão do esquema de banco de dados) e realiza a migração dos dados com base nos mapeamentos especificados no segundo modelo.

Expressividade das alterações nos modelos. Conforme discutido no item anterior, a biblioteca DynaSchema se baseia nas informações contidas nos modelos de esquema de dados do SaS e de plano de migração para executar a evolução do esquema de banco de dados.

Em relação ao primeiro modelo, o módulo Gerenciador de Esquema já fornece alguns métodos para adicionar uma nova entidade no esquema de dados, remover uma entidade existente e adicionar um novo campo, renomear um campo, trocar o nome do campo ou alterar o tipo do campo de uma entidade existente (veja a classe `SchemaManager` no pacote `schema` do diagrama ilustrado na Figura 30). Essas funcionalidades permitem que a biblioteca DynaSchema possa lidar com as mudanças no nível estrutural que precisam ser feitas no banco de dados para acompanhar os ciclos de autoadaptação do SaS, porém a biblioteca ainda não é capaz de atender as mudanças no nível comportamental, como a evolução dos procedimentos armazenados (em inglês, *stored procedures*), por exemplo.

Por outro lado, a personalização do mapeamento de dados entre as versões antiga e nova do esquema de dados do SaS pode ser definida no segundo modelo. Nesse sentido, é possível indicar se o campo de uma entidade deve ser preenchido com os dados de outro campo da mesma ou de outra entidade, se não deve ser preenchido ou se existe a necessidade de realizar um mapeamento de valores entre os dados antigos e novos (veja as classes `FieldMapping` e `ValueMapping` no pacote `migration::model` do diagrama ilustrado na Figura 33). Atualmente, a biblioteca DynaSchema ainda não trata o caso de mapeamento entre um campo multivalorado para monovalorado e vice-versa.

Vale salientar que as funcionalidades supracitadas podem ser combinadas para expressar o que deve ser feito para evoluir o esquema de banco de dados de maneira adequada, porém não é possível garantir que a biblioteca DynaSchema consiga atender a todas as necessidades de um SaS da vida real. O projeto da biblioteca DynaSchema foi baseado nas descobertas obtidas no SMS conduzido no Capítulo 3 e podem existir outros cenários de autoadaptação não usuais de um SaS que não foram cobertos por esse SMS.

Gatilho das modificações no esquema de dados da aplicação. No estudo de caso apresentado neste capítulo foram reportados alguns exemplos de execução manuais, contidos nas Listagens 3 e 4, para facilitar a compreensão do funcionamento da biblioteca DynaSchema. Entretanto, conforme discutido na Seção 5.3, o subsistema gerenciador do SaS constituí a parte da aplicação que contém a lógica de adaptação, sendo responsável por definir quais mudanças devem ser feitas no subsistema gerenciado (ou seja, na lógica de negócio da aplicação). Em um cenário real de execução (cujo contexto pode ser visto na Figura 43), o código apresentado nas Listagens 3 e 4 não é estático e deve ser definido de maneira dinâmica de acordo com o plano de adaptação estabelecido pelo subsistema gerenciador do SaS. Nesse sentido, a lógica de adaptação do SaS deve se comunicar com a biblioteca DynaSchema, informando quais alterações devem ser feitas no esquema de dados da aplicação e quando as atividades para a evolução do esquema podem ser executadas.

Generalização de utilização. A biblioteca DynaSchema foi projetada para apoiar a evolução do esquema de banco de dados no domínio de SaS, porém acredita-se que essa biblioteca também possa atender às necessidades em um domínio geral de software. Nesse sentido, podem ser destacadas duas características de projeto que promovem a generalização do uso da biblioteca DynaSchema em outros domínios de software, a saber:

1. A primeira característica está relacionada com o mecanismo de persistência baseado na JPA, pois essa API já possui um longo histórico de aceitação e validação pela comunidade Java, sendo empregada no desenvolvimento de diversos tipos de sistema de software, como aplicações *web* e microsserviços, por exemplo; e
2. A segunda característica está relacionada com a maneira como a biblioteca DynaSchema executa o processo de evolução do esquema de banco de dados. Nesse sentido, o SaS deve informar para a biblioteca quais mudanças devem ser feitas no esquema de dados da aplicação e qual é o momento mais apropriado para executar cada um das etapas para a evolução do esquema de banco de dados.

Em um domínio de software tradicional (ou seja, que não requer evolução em tempo de execução), essas atividades podem ser coordenadas pelos desenvolvedores através de declarações manuais de código estático, assim como foi feito nos exemplos de execução contidos nas Listagens 3 e 4 deste estudo de caso.

5.6 Considerações Finais

Neste capítulo foi apresentado um estudo de caso para avaliar o comportamento da biblioteca DynaSchema em relação aos seus requisitos de projeto. Inicialmente, na Seção 5.1 foi elaborado um sistema alvo fictício para avaliar essa biblioteca, cujo contexto de funcionamento e seus detalhes de implementação foram abordados na Seção 5.2. Em seguida, nas Seções 5.3 e 5.4 foram apresentados os casos de aplicação da biblioteca DynaSchema para apoiar a evolução do esquema de dados do sistema alvo proposto neste estudo de caso. Tais casos serviram para avaliar a conformidade das funcionalidades contidas no projeto dessa biblioteca. Por fim, na Seção 5.5 foram reportados os resultados obtidos no estudo de caso conduzido neste capítulo.

6 Conclusão

Neste projeto de mestrado acadêmico foi apresentada uma nova solução que é capaz de lidar com o problema da evolução do esquema de dados no domínio de SaS. Em síntese, essa solução foi batizada com o nome DynaSchema e suas funcionalidades foram empacotadas em uma biblioteca que pode ser reutilizada nos projetos de SaS. Para isso, tal biblioteca atua como uma camada intermediária (ou seja, um *middleware*) entre o SaS e seu banco de dados relacional, onde o SaS deve informar a biblioteca quais mudanças foram feitas no esquema de dados da aplicação e quando devem ser executadas etapas da evolução desse esquema de dados. Nesse sentido, a biblioteca DynaSchema possui três responsabilidades principais, a saber: (i) estabelecer o meio para persistir as informações no banco de dados; (ii) modificar o esquema de dados de acordo com as alterações estruturais exigidas pelo SaS; e (iii) notificar o SaS em relação à execução das etapas para a evolução do esquema de dados da aplicação.

Do ponto de vista operacional, o SaS que utiliza a biblioteca DynaSchema como um mecanismo de persistência tem a capacidade de inserir, alterar, remover e consultar suas informações em um banco de dados relacional. Durante os ciclos de autoadaptação do SaS, essa biblioteca é utilizada para evoluir o esquema de banco de dados em tempo de execução, permitindo que o SaS possa acessar suas informações armazenadas mesmo após uma mudança estrutural no seu modelo de dados. Por sua vez, essa biblioteca notifica o SaS em relação ao estado de execução das etapas para a evolução do esquema de dados da aplicação, assim o SaS pode detectar as mudanças no seu ambiente operacional.

De acordo com o projeto apresentado nesta dissertação de mestrado, o desenvolvimento desse tipo de solução envolve uma atividade complexa com uma carga elevada de conceitos e conhecimentos de mais de um domínio de software. Inicialmente, no Capítulo 2 foram apresentados os conceitos e as definições abordados no projeto da biblioteca DynaSchema. Em seguida, no Capítulo 3 foi conduzido um SMS sobre a evolução do esquema de dados para o domínio de SaS, cujos resultados evidenciaram a relevância do tema de pesquisa desta dissertação de mestrado e contribuíram para o projeto da biblioteca DynaSchema. Nos Capítulos 4 e 5 foram reportados os detalhes de projeto e da avaliação dessa biblioteca, respectivamente. Por fim, neste capítulo são apresentadas as conclusões desta dissertação de mestrado da seguinte maneira: na Seção 6.1 são abordadas as contribuições dessa dissertação; na Seção 6.2 são reportadas as principais dificuldades enfrentadas durante o desenvolvimento da biblioteca DynaSchema e suas limitações; e na Seção 6.3 são apresentados as atividades para trabalhos futuros.

6.1 Contribuições

No Capítulo 3 foi conduzido um SMS baseado em sete QPs que seguiu as diretrizes propostas por Petersen, Vakkalanka e Kuzniarz (2015). Com base nas evidências levantas nos resultados desse SMS, foi possível estabelecer um panorama amplo sobre a evolução do esquema de dados no domínio de SaS (ou seja, a temática de pesquisa deste projeto de mestrado acadêmico) que revelou as lacunas e os assuntos que precisam ser investigados com mais detalhes. Embora a principal contribuição deste projeto de mestrado seja a biblioteca DynaSchema, a revisão da literatura apresentada no Capítulo 3 também pode ser destacada com uma contribuição tanto para os pesquisadores quanto para os profissionais da indústria de software, pois esse SMS pode ser utilizado com um guia para nortear o desenvolvimento e/ou aprimoramento de novas soluções relacionadas com a evolução do esquema de dados no domínio de SaS.

A biblioteca DynaSchema desenvolvida neste projeto de mestrado acadêmico constitui a principal contribuição para os pesquisadores e os profissionais da indústria de software. Nesse sentido, essa biblioteca apresenta uma nova solução para a evolução do esquema de dados no domínio de SaS que condensa as melhores evidências desse tema de pesquisa, além de abordar as principais lacunas de investigação relacionadas com essa temática. Vale lembrar que a biblioteca DynaSchema foi projetada para atuar como uma camada intermediária entre o SaS e seu banco de dados relacional, assim estabelecendo uma interface de utilização mais abstrata e transparente aos desenvolvedores de software. Além disso, vale destacar que essa biblioteca adotou alguns padrões de projeto e boas práticas de engenharia, permitindo a extensão dessa solução sem a necessidade de grandes esforços de desenvolvimento. De acordo com o escopo deste projeto de mestrado, o *design* da biblioteca DynaSchema foi planejado para maximizar a rapidez de ampliação das capacidades dessa solução para lidar com novos interesses (por exemplo, como a facilidade de adição de novas estratégias de dialeto SQL).

O estudo de caso conduzido no Capítulo 5 também pode ser considerado uma contribuição deste projeto de mestrado, pois fornece alguns aspectos importantes sobre a aplicabilidade da biblioteca DynaSchema em um cenário real de funcionamento. Apesar do estudo de caso abordar a aplicação dessa biblioteca em um sistema específico, as operações de migração do esquema e dos dados contidas no processo de evolução do esquema de dados do SaS são consideradas essenciais e devem ser executadas em qualquer tipo de sistema, independentemente do seu domínio de software. Além disso, o estudo de caso apresentado neste projeto de mestrado acadêmico teve como objetivo primário a validação da biblioteca DynaSchema como uma solução factível para lidar com a evolução do esquema de dados no domínio de SaS, cujos resultados permitiram a coleta de parâmetros relevantes sobre seu comportamento e a aderência aos requisitos de projeto.

A biblioteca DynaSchema foi projetada originalmente para operar no domínio de SaS, porém acredita-se que essa solução possui o potencial para atuar em outros domínios de software. Nesse sentido, tal biblioteca apresenta um conjunto de funcionalidades que atende as necessidades de um sistema de software tradicional, além de possuir uma interface abstrata que pode ser utilizada de maneira genérica em qualquer aplicação (ou seja, independente de projeto). Portanto, a biblioteca DynaSchema pode servir como uma referência na elaboração de futuras iniciativas para lidar com a evolução do esquema de dados em diferentes domínios de software.

6.2 Dificuldades e Limitações

De acordo com as evidências obtidas no SMS conduzido no Capítulo 3, foram observados vários aspectos que podem ser explorados em relação a evolução do esquema de dados no domínio de SaS. Devido a restrição de tempo para a conclusão deste projeto de mestrado acadêmico, foi estabelecido um escopo de trabalho (Seção 4.1) para projetar e desenvolver a biblioteca DynaSchema que constituí uma nova solução para lidar com esse problema. Nesse sentido, tal escopo priorizou a compatibilidade dessa biblioteca com diferentes tipos de SaS existentes na literatura (como os exemplos contidos no repositório mantido por Vogel (2023)), além de considerar as necessidades do nosso grupo de pesquisa. Vale salientar que o projeto de *design* da biblioteca DynaSchema foi direcionado para a possibilidade de generalização e flexibilização dessa solução em relação ao suporte de novas funcionalidades. Apesar desses cuidados, essa biblioteca não pode ser considerada uma solução do tipo "bala de prata", que consegue cobrir todas as necessidades exigidas por um SaS do mundo real.

Devido à natureza multidisciplinar da biblioteca DynaSchema, foram encontrados vários desafios durante a implementação prática dessa solução, a saber: o desenvolvimento de modelos eficazes para representar a estrutura das entidades do SaS e o plano de migração; a necessidade de implementar a interface da JPA como maneira de aumentar a aceitação da biblioteca através do reaproveitamento de conceitos bem estabelecidos; a execução das operações de evolução do esquema de dados da aplicação em tempo de execução; a utilização intensiva da API de reflexão da linguagem Java para obter as informações estruturais e de persistência das entidades de software do SaS; a criação de múltiplas versões "virtuais" de esquema de banco de dados que precisam de um mecanismo de sincronização de dados entre si; e o processo de migração de dados que deve lidar com duas versões estruturais diferentes do esquema de banco de dados e que pode exigir algum tipo de mapeamento de dados.

Por fim, o estudo de caso conduzido no Capítulo 5 apresentou uma maneira de avaliar a biblioteca DynaSchema, cujos resultados revelaram um cenário favorável sobre o funcionamento e a viabilidade dessa solução em uma aplicação prática. Entretanto não foi possível realizar neste momento uma análise em relação às questões de desempenho dessa biblioteca. Essa limitação ocorreu por diferentes fatores, a saber: (i) a restrição de tempo inerente a um projeto de mestrado acadêmico; (ii) a complexidade do projeto e implementação da biblioteca DynaSchema; e (iii) pela própria natureza desse tipo de atividade.

6.3 Trabalhos Futuros

Durante a condução das atividades prevista para este projeto de mestrado, foram identificadas algumas tarefas que podem ser realizadas como trabalhos futuros, a saber:

- **Condução de outros estudos de casos e experimentos:** esta tarefa está relacionada com a avaliação do comportamento da biblioteca DynaSchema em outros cenários de utilização, como na verificação da experiência do usuário em relação a facilidade de utilização dessa biblioteca e a investigação do seu funcionamento em exemplos práticos da vida real, por exemplo. Nesse sentido, tais experimentos estão interessados na observação da completude dos processos de migração do esquema e dos dados da aplicação, na validação da suficiência das operações de manipulação do esquema e/ou dos dados e na aferição da performance da biblioteca em lidar com diferentes tamanhos de massas de dados;

- **Condução de estudos de casos em outros domínios de software:** no Capítulo 5 foi apresentado um estudo de caso para avaliar a biblioteca DynaSchema em relação a um sistema no contexto da arquitetura de referência RA4SaS voltada para o domínio de sistemas da informação. Nesse sentido, avalia-se como uma atividade viável a verificação do comportamento dessa biblioteca em aplicações voltadas para o domínio de sistemas críticos, como um sistema para cuidado de pacientes dentro de um ambiente doméstico (CAMARGO *et al.*, 2024), por exemplo. Esses sistemas possuem níveis de criticidade mais elevado em relação a aplicação explorada no Capítulo 5, pois demandam alta confiabilidade e resiliência. No caso do sistema citado como exemplo, a ocorrência de falhas/anomalias pode resultar em sérios danos ao paciente que está sendo monitorado; e

- **Acompanhar a evolução do tema de pesquisa:** esta tarefa consiste na atualização do SMS reportado no Capítulo 3. Dessa maneira, espera-se observar as tendencias sobre a evolução do esquema de dado no domínio de SaS, além de monitorar os interesses da comunidade científica e da indústria em relação a esse tipo de solução.

REFERÊNCIAS

AFFONSO, F. J.; LEITE, G.; NAKAGAWA, E. Y. Dms-modeler: A tool for modeling decision-making systems for self-adaptive software domain. In: *The 28th International Conference on Software Engineering & Knowledge Engineering (SEKE' 16)*. [s.n.], 2016. p. 617–622. ISSN 2325-9086. Disponível em: <https://doi.org/10.18293/SEKE2016-184>. Citado na página 13.

AFFONSO, F. J. *et al.* A framework based on learning techniques for decision-making in self-adaptive software. In: *The 27th International Conference on Software Engineering and Knowledge Engineering. (SEKE' 2015)*. Wyndham Pittsburgh University Center, Pittsburgh, USA: Knowledge Systems Institute Graduate School, 2015. p. 24–29. ISSN 2325-9086. Disponível em: <https://doi.org/10.18293/seke2015-125>. Citado na página 12.

AFFONSO, F. J.; NAKAGAWA, E. Y. A reference architecture based on reflection for self-adaptive software. In: *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse*. [s.n.], 2013. p. 129–138. Disponível em: <https://doi.org/10.1109/SBCARS.2013.24>. Citado 4 vezes na(s) página(s) 12, 20, 25 e 26.

AFFONSO, F. J.; PASSINI, W. F.; NAKAGAWA, E. Y. A reference architecture to support the development of mobile applications based on self-adaptive services. *Pervasive and Mobile Computing*, v. 53, p. 33–48, 2019. ISSN 1574-1192. Disponível em: <https://doi.org/10.1016/j.pmcj.2019.01.001>. Citado 3 vezes na(s) página(s) 11, 13 e 20.

ANGELOV, S.; GREFEN, P.; GREEFHORST, D. A framework for analysis and design of software reference architectures. *Information and Software Technology*, v. 54, n. 4, p. 417 – 431, 2012. Disponível em: <https://doi.org/10.1016/j.infsof.2011.11.009>. Citado na página 20.

ASSOUDI, H.; LOUNIS, H. Self-healing data exchange process under evolving schemas: A new mapping adaptation approach based on self-optimization. In: *2011 IEEE 13th International Symposium on High-Assurance Systems Engineering*. [s.n.], 2011. p. 188–190. ISSN 1530-2059. Disponível em: <https://doi.org/10.1109/HASE.2011.42>. Citado 2 vezes na(s) página(s) 43 e 49.

ATZENI, P. *et al.* A runtime approach to model-generic translation of schema and data. *Information Systems*, v. 37, n. 3, p. 269–287, 2012. ISSN 0306-4379. Disponível em: <https://doi.org/10.1016/j.is.2011.11.003>. Citado 3 vezes na(s) página(s) 41, 48 e 51.

AZEROUAL, O.; JHA, M. Without data quality, there is no data migration. *BIG DATA AND COGNITIVE COMPUTING*, v. 5, n. 2, JUN 2021. Disponível em: <https://doi.org/10.3390/bdcc5020024>. Citado na página 51.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice*. 2. ed. [S.l.]: Addison-Wesley Professional, 2003. ISBN 978-03-211-5495-8. Citado na página 20.

BENATO, G. S.; AFFONSO, F. J.; NAKAGAWA, E. Y. Infrastructure based on template engines for automatic generation of source code for self-adaptive software domain. In: *The 29th International Conference on Software Engineering & Knowledge Engineering (SEKE' 17)*. Wyndham Pittsburgh University Center, Pittsburgh, USA: [s.n.], 2017. p. 1–6. ISSN 2325-9086. Disponível em: <https://doi.org/10.18293/SEKE2017-147>. Citado 4 vezes na(s) página(s) 13, 26, 27 e 28.

BEURER-KELLNER, L. *et al.* A transformational approach to managing data model evolution of web services. *IEEE Transactions on Services Computing*, p. 1–1, 2022. ISSN 1939-1374. Disponível em: <https://doi.org/10.1109/TSC.2022.3144613>. Citado 3 vezes na(s) página(s) 41, 48 e 51.

BHATTACHERJEE, S. *et al.* Bullfrog: Online schema evolution via lazy evaluation. In: ____. *Proceedings of the 2021 International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 2021. p. 194–206. ISBN 9781450383431. Disponível em: <https://doi.org/10.1145/3448016.3452842>. Citado 3 vezes na(s) página(s) 43, 48 e 52.

BRUN, Y. *et al.* Engineering self-adaptive systems through feedback loops. In: ____. *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 48–70. Disponível em: <https://doi.org/10.1007/978-3-642-02161-9_3>. Citado 3 vezes na(s) página(s) 12, 21 e 23.

CAMARGO, M. P. d. O. *et al.* Ra4self-cps: A reference architecture for self-adaptive cyber-physical systems. *IEEE Latin America Transactions*, v. 22, n. 2, p. 113–125, 2024. Disponível em: <https://doi.org/10.1109/TLA.2024.10412036>. Citado na página 136.

CHENG, B. H. C. *et al.* Software engineering for self-adaptive systems: A research roadmap. In: ____. *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 1–26. ISBN 978-3-642-02161-9. Disponível em: <https://doi.org/10.1007/978-3-642-02161-9_1>. Citado 6 vezes na(s) página(s) 11, 12, 21, 23, 24 e 35.

COCKBURN, A. *Hexagonal architecture*. 2005. Disponível em: <https://web.archive.org/web/20220919205430/https://alistair.cockburn.us/hexagonal-architecture/>. Acesso em: 26 out. 2022. Citado na página 18.

DATE, C. J. *Sistemas de banco de dados*. 8. ed. [S.l.]: Elsevier Editora Ltda., 2004. ISBN 978-85-3521-273-0. Citado 4 vezes na(s) página(s) 30, 31, 32 e 33.

DIESTE, O.; GRIMÁN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, v. 14, n. 5, p. 513–539, Oct 2009. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-008-9091-7>. Citado na página 146.

DYBA, T.; DINGSOYR, T.; HANSSEN, G. K. Applying systematic reviews to diverse study types: An experience report. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. [s.n.], 2007. p. 225–234. Disponível em: <https://doi.org/10.1109/ESEM.2007.59>. Citado na página 146.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. 7. ed. [S.l.]: Pearson Education do Brasil, 2019. ISBN 978-85-430-2500-1. Citado 4 vezes na(s) página(s) 30, 31, 32 e 33.

FOWLER, M. *Padrões de arquitetura de aplicações corporativas*. [S.l.]: Bookman, 2006. ISBN 978-85-363-0638-4. Citado 3 vezes na(s) página(s) 16, 17 e 18.

GAMMA, E. *et al. Padrões de projetos: soluções reutilizáveis de software orientado a objetos*. [S.l.]: Bookman, 2000. ISBN 978-85-7307-610-3. Citado 2 vezes na(s) página(s) 18 e 19.

GöTZ, S.; KüHN, T. Models@run.time for object-relational mapping supporting schema evolution. In: . [s.n.], 2015. v. 1474, p. 41–50. Cited By 1. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954564351&partnerID=40&md5=ff323df66409bddb8c99cfda6e4737da>. Citado 2 vezes na(s) página(s) 41 e 48.

HILLENBRAND, A. *et al.* Self-adapting data migration in the context of schema evolution in nosql databases. *Distributed and Parallel Databases*, 2021. ISSN 1573-7578. Disponível em: <https://doi.org/10.1007/s10619-021-07334-1>. Citado 2 vezes na(s) página(s) 42 e 52.

HILLENBRAND, A. *et al.* Towards self-adapting data migration in the context of schema evolution in nosql databases. In: *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*. [s.n.], 2020. p. 133–138. Disponível em: <https://doi.org/10.1109/ICDEW49219.2020.00013>. Citado 4 vezes na(s) página(s) 11, 12, 51 e 52.

IBM. *An architectural blueprint for autonomic computing*. 2005. [*On-line*], *World Wide Web*. Disponível em: <https://web.archive.org/web/20200209045605/http://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf>. Acesso em: 15 mar. 2020. Citado na página 25.

JONG, M. de; DEURSEN, A. van; CLEVE, A. Zero-downtime sql database schema evolution for continuous deployment. In: *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. [s.n.], 2017. p. 143–152. Disponível em: <https://doi.org/10.1109/ICSE-SEIP.2017.5>. Citado 4 vezes na(s) página(s) 42, 52, 122 e 123.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. v. 2, jan 2007. Disponível em: <https://www.researchgate.net/publication/302924724_Guidelines_for_performing_Systematic_Literature_Reviews_in_Software_Engineering>. Citado 2 vezes na(s) página(s) 13 e 35.

KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-based software engineering. In: *Proceedings of the 26th International Conference on Software Engineering*. USA: IEEE Computer Society, 2004. (ICSE '04), p. 273–281. ISBN 0769521630. Disponível em: <https://doi.org/10.1109/ICSE.2004.1317449>. Citado 2 vezes na(s) página(s) 13 e 35.

KRUPITZER, C. *et al.* A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, v. 17, p. 184–206, 2015. ISSN 1574-1192. 10 years of Pervasive Computing' In Honor of Chatschik Bisdikian. Disponível em: <https://doi.org/10.1016/j.pmcj.2014.09.009>. Citado 3 vezes na(s) página(s) 12, 21 e 23.

LEMOS, R. de *et al.* Software engineering for self-adaptive systems: A second research roadmap. In: ____. *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 1–32. Disponível em: <https://doi.org/10.1007/978-3-642-35813-5_1>. Citado na página 11.

MAES, P. Concepts and experiments in computational reflection. In: . New York, NY, USA: Association for Computing Machinery, 1987. (OOPSLA '87), p. 147–155. ISBN 0897912470. Disponível em: <https://doi.org/10.1145/38765.38821>. Citado na página 25.

MARKS, R. M.; STERRITT, R. A metadata driven approach to performing complex heterogeneous database schema migrations. *Innovations in Systems and Software Engineering*, v. 9, n. 3, p. 179, 2013. ISSN 1614-5054. Disponível em: <https://doi.org/10.1007/s11334-013-0217-8>. Citado 3 vezes na(s) página(s) 41, 47 e 51.

MARTIN, R. C. *Arquitetura limpa: O Guia do Artesão para Estrutura e Design de Software*. [S.l.]: Alta Books, 2019. ISBN 978-85-5080-460-6. Citado na página 18.

MITRPANONT, J. L.; FUGKEAW, S. Direct access versioning for multidimensional database schema creation. In: *The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*. [s.n.], 2006. p. 17–17. Disponível em: <https://doi.org/10.1109/CIT.2006.79>. Citado 4 vezes na(s) página(s) 43, 48, 51 e 52.

NAMDEO, B.; SUMAN, U. A model for relational to nosql database migration: Snapshot-live stream db migration model. In: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. [s.n.], 2021. v. 1, p. 199–204. ISSN 2575-7288. Disponível em: <https://doi.org/10.1109/ICACCS51430.2021.9441829>. Citado 3 vezes na(s) página(s) 43, 47 e 52.

NEAMTIU, I. *et al.* Improving cloud availability with on-the-fly schema updates. In: *Proceedings of the 19th International Conference on Management of Data*. Mumbai, Maharashtra, IND: Computer Society of India, 2013. (COMAD '13), p. 24–34. Disponível em: <https://dl.acm.org/doi/10.5555/2694476.2694487>. Citado 2 vezes na(s) página(s) 43 e 48.

PALERMO, J. *The Onion Architecture*. 2008. Disponível em: <https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1>. Acesso em: 26 out. 2022. Citado na página 18.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, v. 64, p. 1–18, 2015. ISSN 0950-5849. Disponível em: <https://doi.org/10.1016/j.infsof.2015.03.007>. Citado 5 vezes na(s) página(s) 13, 35, 36, 133 e 150.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. 8. ed. [S.l.]: AMGH Editora Ltda., 2016. ISBN 978-85-8055-533-2. Citado 3 vezes na(s) página(s) 16, 17 e 18.

PUKALL, M. *et al.* Javadaptor – flexible runtime updates of java applications. *Software - Practice and Experience*, v. 43, n. 2, p. 153–185, 2013. Cited By 34. Disponível em: <https://doi.org/10.1002/spe.2107>. Citado 2 vezes na(s) página(s) 41 e 48.

RA, Y.-G.; RUNDENSTEINER, E. A. Oodb support for providing transparent schema changes. In: *Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 1994. (CASCON '94), p. 57. Disponível em: <https://dl.acm.org/doi/10.5555/782185.782242>. Citado 3 vezes na(s) página(s) 43, 48 e 51.

RAHM, E.; BERNSTEIN, P. A. An online bibliography on schema evolution. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 35, n. 4, p. 30–31, dec 2006. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1228268.1228273>. Citado na página 36.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, Association for Computing Machinery, New York, NY, USA, v. 4, n. 2, maio 2009. ISSN 1556-4665. Disponível em: <https://doi.org/10.1145/1516533.1516538>. Citado 9 vezes na(s) página(s) 11, 12, 21, 22, 23, 24, 25, 35 e 60.

SANTOS, R. J.; BERNARDINO, J.; VIEIRA, M. 24/7 real-time data warehousing: A tool for continuous actionable knowledge. In: *2011 IEEE 35th Annual Computer Software and Applications Conference*. [s.n.], 2011. p. 279–288. ISSN 0730-3157. Disponível em: <https://doi.org/10.1109/COMPSAC.2011.44>. Citado 2 vezes na(s) página(s) 42 e 47.

SCHULER, R.; KESSELMAN, C. Chisel: a user-oriented framework for simplifing database evolution. *Distributed and Parallel Databases*, v. 39, n. 2, p. 483, 2021. ISSN 1573-7578. Disponível em: <https://doi.org/10.1007/s10619-020-07314-x>. Citado 4 vezes na(s) página(s) 46, 50, 51 e 53.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de banco de dados*. 7. ed. [S.l.]: GEN LTC, 2021. ISBN 978-85-9515-733-0. Citado 2 vezes na(s) página(s) 29 e 31.

SOMMERVILLE, I. *Engenharia de software*. 10. ed. [S.l.]: Pearson Education do Brasil Ltda., 2019. ISBN 978-85-430-2497-4. Citado 4 vezes na(s) página(s) 16, 17, 18 e 54.

SONG, Y. *et al.* Automatic adaptation of software applications to database evolution by graph differencing and aop-based dynamic patching. In: *2012 IEEE 36th Annual Computer Software and Applications Conference*. [s.n.], 2012. p. 111–118. ISSN 0730-3157. Disponível em: <https://doi.org/10.1109/COMPSAC.2012.21>. Citado 2 vezes na(s) página(s) 42 e 48.

SOUNDARARAJAN, G.; AMZA, C. Online data migration for autonomic provisioning of databases in dynamic content web servers. In: *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 2005. (CASCON '05), p. 268–282. Disponível em: <https://dl.acm.org/doi/10.5555/1105634.1105654>. Citado 2 vezes na(s) página(s) 43 e 52.

STIEMER, A. *et al.* Polymigrate: Dynamic schema evolution and data migration in a distributed polystore. In: GADEPALLY, V. *et al.* (Ed.). *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Cham: Springer International Publishing, 2021. p. 42–53. ISBN 978-3-030-71055-2. Disponível em: <https://doi.org/10.1007/978-3-030-71055-2_4>. Citado 3 vezes na(s) página(s) 50, 51 e 52.

STÖRL, U.; KLETTKE, M.; SCHERZINGER, S. Nosql schema evolution and data migration: State-of-the-art and opportunities. *Advances in Database Technology - EDBT*, v. 2020-March, p. 655–658, 2020. Disponível em: <https://doi.org/10.5441/002/edbt.2020.87>. Citado na página 12.

VOGEL, T. *Software Engineering for Self-Adaptive Systems exemplars repository*. 2023. [Online]. Disponível em: <http://self-adaptive.org/exemplars>. Acesso em: 15 mar. 2024. Citado 2 vezes na(s) página(s) 12 e 134.

WANG, Q.; DU, Z.; LIU, N. Design and realization of database online migration. In: *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*. [s.n.], 2012. p. 1195–1198. Disponível em: <https://doi.org/10.1109/ICCSNT.2012.6526138>. Citado na página 42.

WEYNS, D. Software engineering of self-adaptive systems. In: ____. *Handbook of Software Engineering*. Cham: Springer International Publishing, 2019. p. 399–443. Disponível em: <https://doi.org/10.1007/978-3-030-00262-6_11>. Citado na página 11.

WEYNS, D. *et al.* On patterns for decentralized control in self-adaptive systems. In: ____. *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 76–107. ISBN 978-3-642-35813-5. Disponível em: <https://doi.org/10.1007/978-3-642-35813-5_4>. Citado 4 vezes na(s) página(s) 12, 21, 22 e 23.

APÊNDICE A – Lista de Estudos Primários

Na Tabela 1 são listados os estudos primários incluídos no SMS detalhado no Apêndice B. Essa tabela é composta por quatro colunas, a saber: (i) *ID*, que contém o identificador do estudo; (ii) *Referência*, que indica a referência para o estudo; (iii) *Ano*, que informa o ano em que o estudo foi publicado; e (iv) *FB*, que indica em qual FB foi encontrado o estudo. Vale lembrar que os detalhes sobre as FBs podem ser consultados na Seção B.3 do Apêndice B.

Tabela 1 – Estudos primários selecionados pelo SMS

ID	**Referência**	**Ano**	**FB**
E01	*Santos, Bernardino e Vieira*. 24/7 Real-Time Data Warehousing: A Tool for Continuous Actionable Knowledge	2011	ZDT
E02	*Marks e Sterritt*. A metadata driven approach to performing complex heterogeneous database schema migrations	2013	SAS
E03	*Namdeo e Suman*. A Model for Relational to NoSQL database Migration: Snapshot-Live Stream Db Migration Model	2021	ZDT
E04	*Atzeni et al*. A runtime approach to model-generic translation of schema and data	2012	SAS
E05	*Beurer-Kellner et al*. A Transformational Approach to Managing Data Model Evolution of Web Services	2022	SAS
E06	*Song et al*. Automatic Adaptation of Software Applications to Database Evolution by Graph Differencing and AOP-Based Dynamic Patching	2012	SAS
E07	*Bhattacherjee et al*. BullFrog: Online Schema Evolution via Lazy Evaluation	2021	ZDT
E08	*Wang, Du e Liu*. Design and realization of database online migration	2012	ZDT
E09	*Mitrpanont e Fugkeaw*. Direct Access Versioning for Multidimensional Database Schema Creation	2006	ZDT
E10	*Neamtiu et al*. Improving Cloud Availability with On-the-Fly Schema Updates	2013	ZDT

Continua na próxima página

Continuando da página anterior

ID	Referência	Ano	FB
E11	*Pukall et al*. JavAdaptor – Flexible runtime updates of Java applications	2013	SAS
E12	*Götz e Kühn*. Models@run.time for object-relational mapping supporting schema evolution	2015	SAS
E13	*Soundararajan e Amza*. Online Data Migration for Autonomic Provisioning of Databases in Dynamic Content Web Servers	2005	SAS
E14	*Ra e Rundensteiner*. OODB Support for Providing Transparent Schema Changes	1994	SAS
E15	*Hillenbrand et al*. Self-adapting data migration in the context of schema evolution in NoSQL databases	2021	SAS
E16	*Assoudi e Lounis*. Self-Healing Data Exchange Process under Evolving Schemas: A New Mapping Adaptation Approach Based on Self-Optimization	2011	SAS
E17	*Jong, Deursen e Cleve*. Zero-Downtime SQL Database Schema Evolution for Continuous Deployment	2017	ZDT

Fonte: Elaborada pelo autor.

APÊNDICE B – Protocolo de Pesquisa

Neste apêndice são apresentados os detalhes do protocolo de pesquisa adotado no SMS conduzido neste projeto de mestrado acadêmico. Inicialmente, na Seção B.1 são listadas as QPs que guiaram esse SMS no processo da busca de evidências para o tema da evolução do esquema de dados para o domínio de SaS. Em seguida, na Seção B.2 é apresentada a estratégia de pesquisa adotada nesse SMS para a seleção dos estudos primários. Posteriormente, na Seção B.3 é revelada a maneira como essa pesquisa foi conduzida, detalhando o processo de filtragem dos estudos relevantes. Na Seção B.4 é informado como foi realizada a extração e síntese das informações levantadas no SMS conduzido neste projeto de mestrado. Finalmente, na Seção B.5 são abordas algumas possíveis limitações do mapeamento elaborado nesse SMS.

B.1 Questões de Pesquisa

O SMS conduzido neste projeto de mestrado tem como objetivo a identificação de como as comunidades cientifica e industrial têm lidado com o tema da evolução do esquema de dados para o domínio de SaS. Tendo em vista a busca de evidências para esse tema de interesse, foram elaboradas sete QPs para guiar esse levantamento, a saber:

QP1: Quais são os aspectos demográficos dos estudos?

*Esta questão de pesquisa tem como objetivo a identificação das características de procedência das publicações que abordam soluções para a evolução do esquema de dados. Dentre essas características de interesse estão inclusos os anos de veiculação (**QP1.1**), tipos de publicação (**QP1.2**), quantidade de estudos por FBs (descritas na Seção B.3) (**QP1.3**) e instituições relacionadas aos estudos (**QP1.4**).*

QP2: Quais tipos de soluções são abordadas pelos estudos?

Esta questão de pesquisa tem como objetivo a identificação da natureza das soluções para a evolução do esquema de dados. Tal natureza está relacionada com as ideias centrais que estão por traz das soluções propostas, como abordagens, técnicas, processos, modelos, arquiteturas, bibliotecas, frameworks, ferramentas, entre outros tipos.

QP3: Em quais domínios de aplicação as soluções propostas pelos estudos estão inseridas?

Esta questão de pesquisa tem como objetivo a identificação de quais domínios de aplicação utilizaram as soluções para a evolução do esquema de dados. A partir desses domínios é possível compreender em que contextos e cenários de aplicação as soluções propostas estão inseridas.

QP4: Quais tipos de modelos de dados são abordados pelos estudos?

Esta questão de pesquisa tem como objetivo a identificação de quais tipos de modelos de dados foram abordados pelas soluções para a evolução do esquema de dados. Tais modelos estão relacionados com a maneira que as soluções propostas lidam com o gerenciamento e armazenamento dos seus dados.

QP5: Quais tipos de abordagens são utilizadas pelos estudos para tratar o processo de evolução do esquema de dados?

Esta questão de pesquisa tem como objetivo a identificação das abordagens utilizadas pelas soluções para gerenciar o processo de evolução do esquema de dados. A partir dessas abordagens é possível compreender como as soluções propostas percebem as atividades que estão envolvidas na evolução do esquema de dados.

QP6: Quais atributos não funcionais foram considerados no projeto das solução propostas pelos estudos?

Esta questão de pesquisa tem como objetivo a identificação dos atributos não funcionais que foram considerados para guiar o projeto e desenvolvimento das soluções para a evolução do esquema de dados.

QP7: Como as soluções propostas pelos estudos foram avaliadas?

Esta questão de pesquisa tem como objetivo a identificação das metodologias que foram utilizadas para avaliar as soluções para a evolução do esquema de dados.

B.2 Estratégia de Pesquisa

Baseado nas QPs apresentadas na Seção B.1, foi adotada uma estratégia de pesquisa para guiar a elaboração do SMS conduzido neste projeto de mestrado. Os detalhes adotados nessa estratégia foram esquematizados em seis itens, a saber:

1. **Seleção das bases de pesquisa:** foram adotados alguns critérios para a escolher as bases digitais de publicação científica utilizadas como fontes de pesquisa do SMS, a saber: (i) a regularidade de atualização da base; (ii) a disponibilização de trabalhos na sua integridade; (iii) a obtenção de resultados com precisão e qualidade; e (iv) a flexibilidade de exportação dos resultados obtidos (por exemplo, arquivos com a extensão `bib`, `ris` ou `pdf`). Tais critérios estão alinhados com as sugestões propostas por Dieste, Grimán e Juristo (2009);

2. **Bases de pesquisa:** de acordo com os critérios estabelecidos no item anterior, na Tabela 2 são listadas a seis bases digitais de publicação científica que foram escolhidas como fontes de pesquisa do SMS. Tal tabela é composta por três colunas, a saber: (i) *Nome da Base*, que contém o nome completo da base de pesquisa; (ii) *Sigla*, que referencia o nome da base através de uma sigla de identificação mais curta; e (iii) *Endereço de Acesso*, que informa o endereço eletrônico para acessar a base. É importante salientar que as bases listadas na Tabela 2 foram sugeridas por Dyba, Dingsoyr e Hanssen (2007), sendo consideradas eficazes para conduzir uma revisão sistemática no contexto de engenharia de software;

Tabela 2 – Lista das bases de pesquisa consultadas pelo SMS

Nome da Base	Sigla	Endereço de Acesso
ACM Digital Library	ACM	<https://dl.acm.org>
IEEE Xplore	IEEE	<https://ieeexplore.ieee.org>
ScienceDirect	SD	<https://www.sciencedirect.com>
Scopus	SC	<https://www.scopus.com>
Springer Link	SL	<https://link.springer.com>
Web of Science	WoS	<https://www.webofscience.com>

Fonte: Elaborada pelo autor.

3. **Idioma dos estudos:** visando uma maior abrangência do corpo de conhecimento, somente foram considerados os estudo escritos em língua inglesa, pois esse idioma é geralmente adotado na maioria das publicações científicas;

4. **Palavras-chave:** o SMS considerou a utilização das palavras-chave *schema*, *evolution*, *self-adaptive* e *zero-downtime*. A lista dos termos utilizados e seus respectivos sinônimos são apresentados a seguir:

    - ***Schema*:** *schemas*, *database* e *databases*;

- ***Evolution*:** *evolving*, *co-evolution*, *coevolution*, *migrating* e *migration*;
- ***Self-adaptive*:** *self-adaptable*, *self-adaptation* e *self-adapting*; e
- ***Zero-Downtime*:** *zero downtime*, *downtime-free*, *downtime free*, *zero-breakdown*, *zero breakdown*, *real-time* e *real time*.

5. **String de pesquisa:** as palavras-chave apresentadas no item anterior e seus respectivos sinônimos foram combinados com os operadores booleanos `AND` e `OR` para formar as *strings* de pesquisa. Para ampliar a cobertura do SMS, foram elaboradas duas *strings* de pesquisa para a busca dos estudos primários relevantes. Na Tabela 3 pode ser observada a primeira *string* de pesquisa que se baseia na palavra-chave *self-adaptive*. Tal *string* é mais restrita e tem o objetivo de encontrar os estudos sobre evolução do esquema de dados que abordem especificamente os aspectos da área de SaS. Por outro lado, a segunda *string* de pesquisa gira em torno da palavra-chave *zero-downtime* e pode ser vista na Tabela 4. Essa *string* é mais abrangente e tem o objetivo de encontrar os estudos que abordam a evolução do esquema de dados em outras áreas de aplicação. Tais estudos podem não ser específicos para a área de SaS, porém são capazes de apresentar soluções que estão alinhadas com as necessidades e preocupações de um SaS, como a característica de do tempo de inatividade zero (em inglês, *zero-downtime*), por exemplo; e

Tabela 3 – *String* de pesquisa baseada na palavra-chave *self-adaptive*

```
("self-adaptive"OR "self-adaptable"OR "self-adaptation"OR
"self-adapting") AND
("schema"OR "schemas"OR "database"OR "databases") AND
("evolution"OR "evolving"OR "co-evolution"OR "coevolution"OR
"migrating"OR "migration")
```

Fonte: Elaborada pelo autor.

6. **Critérios de inclusão e exclusão:** de acordo com as QPs apresentadas na Seção B.1, foram estabelecidos alguns CI (Critério de Inclusão) e CEs (Critério de Exclusão) para guiar o processo de seleção dos estudos primários relevantes para o SMS. Foi elaborado um único critério de inclusão (**`CI1`**) que considera os estudos que apresentaram uma abordagem para a evolução do esquema de dados, mas não necessariamente pertencentes ao domínio de SaS. Os critérios de exclusão são listados a seguir:

- **`CE1:`** o estudo não está relacionado com o tema da evolução de esquema de dados;

Tabela 4 – *String* de pesquisa na palavra-chave *zero-downtime*

`("zero-downtime"OR "zero downtime"OR "downtime-free"OR "downtime free"OR "zero-breakdown"OR "zero breakdown"OR "real-time"OR "real time") AND`
`("schema"OR "schemas"OR "database"OR "databases") AND`
`("evolution"OR "evolving"OR "co-evolution"OR "coevolution"OR "migrating"OR "migration")`

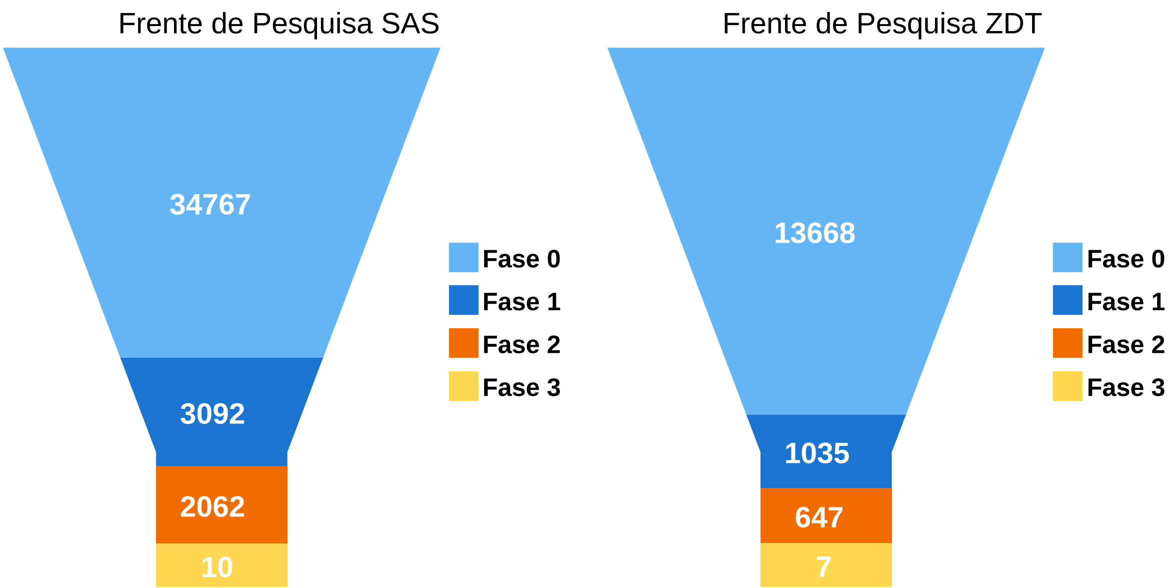
Fonte: Elaborada pelo autor.

- **CE2:** o estudo é um editorial, artigo de opinião, notas de palestra, tutorial, poster, painel ou trabalhos similares;
- **CE3:** o estudo não foi escrito em língua inglesa;
- **CE4:** o estudo é um resumo ou não foi disponibilizado de forma completa;
- **CE5:** o estudo é similar a um trabalho anterior desenvolvido pelo mesmo autor, sendo considerado apenas o estudo mais recente ou a versão mais completa; e
- **CE6:** o estudo é do tipo secundário (por exemplo, um SMS).

B.3 Condução da Pesquisa

O processo de busca nas bases digitais de publicação listadas na Tabela 2 foi realizado no período de 07/02/2022 a 15/02/2022. Tal processo foi dividido em duas FBs isoladas, sendo uma FB para a *string* de pesquisa SAS apresentada na Tabela 3 e outra para a *string* ZDT especificada na Tabela 4. Vale salientar que ambas as FBs foram conduzidas da mesma maneira, apesar de serem processos de busca independentes. Essas *strings* de pesquisa (Tabela 3 e Tabela 4) foram ajustadas e executadas para cada uma das bases digitais visando a exploração de três campos de busca, a saber: (i) *título*; (ii) *resumo*; e (iii) *palavras-chave*. Os estudos primários das duas FBs foram selecionados através de um processo composto por quatro fases de filtragem, conforme ilustrado na Figura 50. Inicialmente, na **Fase 0** estão contidos os estudos que foram retornados pelas buscas nas bases digitais, representando o corpo bruto de estudos para serem filtrados. Em seguida, na **Fase 1** foram removidos os estudos duplicados para cada base de pesquisa, consistindo em uma remoção de duplicadas baseada em uma estratificação individual por base de pesquisa. Além disso, um filtro especial foi aplicado nos estudos retornados pela SL, pois essa base aplica a *string* de pesquisa em todo o conteúdo dos estudos. Tal filtragem é necessária para normalizar a busca nessa base de publicação, considerando somente os campos de título, resumo e palavras-chave dos estudos. Na **Fase 2**, todos os estudos foram reunidos em uma única lista e as duplicidades foram removidas, resultando na exclusão de duplicatas que considera o conjunto de todas as bases digitais. Finalmente, na **Fase 3** foram aplicados o CI e os CEs para selecionar os estudos primários relevantes. Vale ressaltar que no início dessa fase, foram lidos o título, resumo, introdução e conclusão para selecionar os estudos candidatos. Posteriormente, o CI e os CEs foram aplicados para filtrar esses estudos candidatos.

Figura 50 – Quantidade de estudos primários por fase de seleção



Fonte: Elaborada pelo autor.

A quantidade de estudos por base de publicação foi contabilizada durante cada etapa de seleção para as duas frentes de busca (SAS e ZDT), sendo rastreado o número de estudos desde a lista bruta de trabalhos até a lista final de estudos relevantes. Na Tabela 5 pode ser observado a síntese desse levantamento, detalhando a quantidade de estudos primários selecionados por fase de filtragem, base publicação científica e FB. Vale salientar que não foi adotado nenhum critério de qualidade para a seleção desses estudos, pois o SMS conduzido neste projeto de mestrado foi conduzido por um número reduzido de pesquisadores. Após todo o processo de busca e filtragem, foram selecionados 17 estudos que estão listados na Tabela 1 do Apêndice A.

Tabela 5 – Quantidade de estudos primários selecionados por fase de filtragem, base de publicação e frente de pesquisa

Nome da Base	Fase 0 (SAS/ZDT)	Fase 1 (SAS/ZDT)	Fase 2 (SAS/ZDT)	Fase 3 (SAS/ZDT)
ACM Digital Library	164 / 50	121 / 45	/	2 / 2
IEEE Xplore	752 / 296	741 / 296	/	3 / 5
ScienceDirect	191 / 92	144 / 45	– / –	1 / 0
Scopus	1.531 / 450	1.305 / 372	/	2 / 0
Springer Link	31.029 / 12.525	104 / 26	/	2 / 0
Web of Science	1.100 / 255	677 / 176	/	0 / 0
Total (SAS/ZDT)	34.767 / 13.668	3.092 / 1.035	2.062 / 647	10 / 7

Fonte: Elaborada pelo autor.

B.4 Extração e Síntese dos Dados

Conforme sugerido por Petersen, Vakkalanka e Kuzniarz (2015), na Tabela 6 é apresentado o formulário de extração de dados que foi criado para sintetizar as informações apresentadas pelos estudos primários listados na Tabela 1 do Apêndice A. Vale salientar que os campos desse formulário foram elaborados para responder as QPs apresentadas na Seção B.1.

Tabela 6 – Formulário de extração de dados

Tópico de Pesquisa	Descrição	QP
ID	Identificador do estudo	–
Título	Título do estudo	–
Ano	Ano de publicação do estudo	QP1.1
Local de Publicação	Onde foi publicado o estudo	QP1.2
FB	Nome da FB que recuperou o estudo	QP1.3
Base de Pesquisa	Nome da base de pesquisa que recuperou o estudo	QP1.3
Instituição(ões)	Nome(s) da(s) Instituição(ões) do(s) autor(es) do estudo	QP1.4
Solução	Tipo de solução apresentada pelo estudo	QP2
Domínio	Domínio e/ou subdomínio em que está inserido o estudo	QP3

Continua na próxima página

Continuando da página anterior

Tópico de Pesquisa	Descrição	QP
Modelo(s) de Dados	Tipo(s) de modelo de dados abordado pelo estudo	QP4
Abordagem de Evolução	Descrição da abordagem de evolução utilizada pelo estudo	QP5
Atributo(s)	Atributo(s) não funcional(is) considerado(s) pelo estudo	QP6
Avaliação	Descrição da(s) metodologia(s) de avaliação utilizada(s) pelo estudo	QP7

Fonte: Elaborada pelo autor.

B.5 Limitações

Devido à natureza da atividade de mapeamento do SMS conduzido neste projeto de mestrado acadêmico, algumas possíveis limitações podem ser listadas:

- **Omissão de artigos:** o SMS conduzido neste projeto de mestrado utilizou um mecanismo automatizado de busca fornecido pelas seis bases de publicação listadas na Tabela 2, sendo que a *strings* de pesquisa apresentada nas Tabelas 3 e 4 precisaram ser adaptadas para cada base. Infelizmente, não é possível garantir que todos os estudos relacionados com o tema de pesquisa deste projeto de mestrado foram retornados, apesar de todos os esforços tomados para tornar esse SMS mais confiável. Por esse motivo, foram criadas diferentes versões de *strings* de pesquisa para validar e calibrar os estudos retornados, sendo que a versão final das *strings* foram ajustadas posteriormente para cada base de pesquisa. Além disso, foram estabelecidas duas FBs para encontrar estudos específicos da área de SaS e outras áreas de pesquisa correlatas que compartilham as mesmas necessidades e preocupações do domínio de SaS. Dessa maneira, acredita-se que nenhum estudo relevante tenha sido excluído do SMS conduzido neste projeto de mestrado;
- **Confiabilidade dos pesquisadores:** os pesquisadores envolvidos neste projeto de mestrado são da área de Engenharia de Software. Portanto, não foi incluído no SMS nenhum estudo primário de mesma autoria, de grupos de pesquisa ou pesquisadores que este-

jam relacionados com o autor e orientador desta dissertação de mestrado. Dessa maneira, acredita-se que nenhum viés involuntário esteja contido nos resultados do SMS conduzido neste projeto de mestrado acadêmico;

- **Parcialidade:** o SMS conduzido neste projeto de mestrado adotou diretrizes para garantir a imparcialidade de seus resultados. Porém, não é possível afirmar que todas as informações coletadas nesse mapeamento são imparciais, pois a interpretação dos dados está sujeita às percepções do pesquisador que conduziu o mapeamento;

- **Extração dos dados:** existe uma limitação nos dados extraídos no SMS conduzido neste projeto de mestrado, pois as informações contidas nos estudos que respondem às QPs nem sempre são tratadas de maneira explícita, sendo necessário algum grau de interpretação em alguns casos. Além disso, quando houveram dúvidas/divergências na condução do mapeamento, o autor e o orientador deste projeto estabeleceram um consenso para aplicar o CI e os CEs ou para extrair as informações de um determinado estudo. Por esse motivo, foi criado o formulário de extração de dados apresentado na Tabela 6 para coletar e sintetizar as informações de maneira padronizada;

- **Avaliação da qualidade:** a qualidade dos estudos não foi avaliada, pois o SMS conduzido neste projeto de mestrado tem o objetivo de identificar as abordagens de evolução de esquema de dados para o domínio de SaS. Porém, vale salientar a importância da avaliação da qualidade dos estudos, podendo ser considerada em uma versão futura do SMS; e

- **Replicabilidade:** as bases de publicação adotadas como fontes de pesquisa do SMS conduzido neste projeto de mestrado estão disponíveis de maneira *on-line* e são atualizadas de forma constante. Tais características podem influenciar em pesquisas futuras, fazendo com que os estudos retornados pelas buscas não sejam exatamente iguais aos deste projeto, dificultando a sua reprodução.