UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Marcus Vinícius Silva Cruz

Engenharia Reversa Baseada em Modelos para Aplicações de Simulação, Controle e Operação de Plantas na Indústria Petroquímica

Florianópolis
2021

Marcus Vinícius Silva Cruz

Engenharia Reversa Baseada em Modelos para Aplicações de Simulação, Controle e Operação de Plantas na Indústria Petroquímica

Dissertação submetido ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Dr. Leandro Buss Becker
Coorientador: Eng. Me. Thaise Poerschke Damo

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Cruz, Marcus Vinícius Silva
Engenharia Reversa Baseada em Modelos para Aplicações de Simulação, Controle e Operação de Plantas na Indústria Petroquímica / Marcus Vinícius Silva Cruz ; orientador, Leandro Buss Becker, coorientador, Thaise Poerschke Damo, 2021.
105 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2021.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Engenharia Reversa Dirigida a Modelos. 3. Engenharia Reversa. 4. M4PIA. 5. Indústria Petroquímica. I. Becker, Leandro Buss. II. Damo, Thaise Poerschke. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Marcus Vinícius Silva Cruz

Engenharia Reversa Baseada em Modelos para Aplicações de Simulação, Controle e Operação de Plantas na Indústria Petroquímica

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Dr. Fábio Paulo Basso
Universidade Federal do Pampa

Prof. Dr. Ricardo José Rabelo
Universidade Federal de Santa Catarina

Prof. Dr. Julio Elias Normey-Rico
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Werner Kraus Junior
Data: 28/10/2021 09:16:37-0300
CPF: 531.085.239-53
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Werner Kraus Junior
Coordenador do Programa

Documento assinado digitalmente
Leandro Buss Becker
Data: 28/10/2021 07:09:07-0300
CPF: 688.357.870-68
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Leandro Buss Becker
Orientador

Florianópolis, 16 de Agosto de 2021.

Este trabalho é dedicado a minha família: meus pais (Antonio Cruz e Luciene Rocha) e minha irmã (Ana Cruz). Mesmo que não entendam nada sobre este trabalho, cada ação feita para que ele existisse foi provido pelo apoio e amor de vocês.

RESUMO

A indústria petroquímica está se tornando cada vez mais complexa e, como resultado, diferentes plataformas de *software* são utilizadas para auxiliar no projeto de sistemas. Geralmente, ocorre que o mesmo componente da planta física é remodelado em diferentes plataformas de *software*, de forma que o reaproveitamento destes modelos se torna difícil, compromete a interoperabilidade do sistema e gera a necessidade de retrabalho nos projetos. Para sistemas legados nas indústrias petroquímicas a reutilização de códigos demanda um esforço muito grande na reengenharia dos projetos tornando o processo de desenvolvimento mais difícil de manusear para fins de operação, controle, operação e supervisão. Trabalhos anteriores propuseram a infraestrutura denominada *Model-Driven Engineering for Petrochemical Industry Automation* (M4PIA), que permite representar plantas industriais através de modelos diferenciados, compatíveis e orientados a objetos. O M4PIA é composto por três metamodelos em dois níveis de abstração (independente e dependente de plataforma). Por meio de transformações de modelos, a infraestrutura oferece suporte à geração automática de código de um modelo de abstração de alto nível para plataformas de *software* específicas. O presente trabalho propõe a integração de uma engenharia reversa baseada em modelos na infraestrutura M4PIA de forma que a partir uma modelagem de sistemas legados desenvolvidas no mais baixo nível, neste caso, no código-fonte seja possível, através das transformações *Text-to-Model* - Texto para Modelo (T2M) e *Model-to-Model* - Modelo para Modelo (M2M) obter um modelo de mais alto nível de abstração. A infraestrutura M4PIA tem suporte para duas plataformas de domínio muito utilizadas na indústria petroquímica. As plataformas Módulo de Procedimentos Automatizados (MPA), para aplicações de operação, automação e controle de processos industriais; e o *Environment for Modeling, Simulation and Optimization* (EMSO), para simulação de processos petroquímicos. Foi realizada uma revisão das tecnologias utilizadas para desenvolvimento da proposta e uma revisão na literatura de trabalhos científicos relacionados. Uma prova conceito envolvendo a modelagem no nível mais baixo de abstração em ambos os sistemas, MPA e EMSO, para um sistema de compressão de gás de uma plataforma de produção petroquímica é apresenta em quatro cenários distintos abrangendo todas as possibilidades de transformações possíveis dentro da infraestrutura, visando ilusrar a solução, bem como realizar uma análise empírica buscando encontrar pontos de desacordo nas transformações. A solução apresentou-se adequada para a realização de engenharia reversa possibilitando e maior facilidade para reengenharia de sistemas legados na indústria petroquímica.

Palavras-chave: Engenharia Reversa Dirigida a Modelos. Engenharia Reversa. M4PIA. Indústria Petroquímica

ABSTRACT

The petrochemical industry is becoming increasingly complex and, as a result, different software platforms are used to aid in system design. Generally, it happens that the same component of the physical plant is remodelled in different software platforms, so that the reuse of these models becomes difficult, compromising the system's interoperability and generating the need for rework in the projects. For legacy systems in the petrochemical industries, code reuse demands a lot of effort in reengineering projects, making the development process more difficult to handle for operation, control, operation and supervision purposes. Previous works proposed the infrastructure called Model-Driven Engineering for Petrochemical Industry Automation (M4PIA), which allows representing industrial plants through differentiated, compatible and object-oriented models. The M4PIA is composed of three metamodels at two levels of abstraction (independent and platform dependent). Through model transformations, the infrastructure supports automatic code generation from a high-level abstraction model for specific software platforms. The present work proposes the integration of a model-based reverse engineering in the M4PIA infrastructure so that from a modelling of legacy systems developed at the lowest level, in this case, in the source code, it is possible, through Text-to-Model transformations. Text to Model (T2M) and Model to Model (M2M) obtain a model with a higher level of abstraction. The M4PIA infrastructure supports two domain platforms widely used in the petrochemical industry. The Automated Procedures Module (MPA) platforms, for industrial process operation, automation and control applications; and EMSO, for simulating petrochemical processes. A review of the technologies used to develop the proposal and a literature review of related scientific works was carried out. A proof-of-concept involving modelling at the lowest level of abstraction in both systems, MPA and EMSO, for a compression system of a petrochemical production platform is presented in four distinct scenarios covering all possible transformation possibilities within the infrastructure in order to illustrate the use of the solution, as well as performing an empirical analysis seeking to find points of disagreement in the transformations. The solution proved to be suitable for performing reverse engineering, enabling reuse and greater ease in reengineering legacy systems in the petrochemical industry.

Keywords: Model-Driven Reverse Engineering. Reverse Engineering. M4PIA. Petrochemical Industry.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TRECHOS DE CÓDIGO

SIGLAS

4GL	*Fourth Generation Language*
AADL	*Architecture Analysis and Design Language*
AGREE	*Assume Guarantee REasoning Environment*
AMD	*Architecture-Driven Modernization*
ANTLR	*ANother Tool for Language Recognition*
ASM	*Abstract Syntax Model*
AST	*Abstract Syntax Tree*
ATL	*Atlas Transformation Language*
Cenpes	Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello
CLP	Controlador Lógico Programável
DAS	Departamento de Automação e Sistemas
DLL	*Dynamic-link library*
DSL	*Domain Specific Language* - Linguagens de Domínio Específico
EMF	*Eclipse Modeling Framework*
EMSO	*Environment for Modeling, Simulation and Optimization*
GPL	*General-Purpose Programming Languages*
Gra2MoL	*Grammar-to-Model Transformation Language*
GUI	*Graphical User Interface* - Interface Gráfica de Usuário
IEC	*International Electrotechnical Commission*
IMD	*Intermediate Model Discoverer*
J2EE	*Java2 Enterprise Edition*
JEE	*Java Enterprise Edition*
JSP	*JavaServer Pages*
KDM	*Knowledge Discovery Metamodel*
M2M	*Model-to-Model* - Modelo para Modelo
M2T	*Model-to-Text* - Modelo para Texto
M4PIA	*Model-Driven Engineering for Petrochemical Industry Automation*
MARBLE	*Modernization Approach for Recovering Business processes from LEgacy Systems*
MDA	*Model-Driven Architecture*
MDE	*Model-Driven Engineering* - Engenharia Dirigida por Modelos
MDRE	*Model-Driven Reverse Engineering* - Engenharia Reversa Dirigida por Modelos

MIA	*Model-In-Action*
MMS	*Manufacturing Message Specification*
MoDisco	*Model Discovery*
MOF	*Meta-Object Facility*
Mof2Text	*MOF Model to Text Language*
MPA	Módulo de Procedimentos Automatizados
OCL	*Object Constraint Language*
OMG	*Object Management Group*
OPC	*Open Platform Communications*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
QVT	*Query/View/Transformation*
QVTo	QVT *Operational Mappings*
RAD	*Rapid Application Development*
RE	Engenharia Reversa
RTE	Round-Trip Engeneering
SFC	*Sequential Function Chart*
Src2MoF	*Source to Model Framework*
T2M	*Text-to-Model* - Texto para Modelo
Tecgraf	Instituto Tecgraf de Desenvolvimento de Software Técnico-Científico da PUC-Rio
UFSC	Universidade Federal de Santa Catarina
UML	*Unified Modeling Language*
UPPAAL	*Uppaal Model Checker*
VMD	*Virtual Manufacturing Device*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A dependência da utilização de *softwares* no processo de operação, monitoramento e controle é cada vez maior (KLATT; MARQUARDT, 2009) na indústria petroquímica. Seborg et al. (2010) explica que a indústria petroquímica, por meio do petróleo bruto, adquire produtos de grande valor comercial, operando em grandes volumes de produção em condições restringidas. Para que a qualidade dos produtos, a segurança da planta e a lucratividade alcançados se mantenham, a operação precisa ser controlada e mantida nas melhores condições possíveis e estáveis, o que acarreta a automação neste setor seja cada vez mais necessária.

De acordo com Campos et al. (2013), a automação industrial possibilitou a criação de complexos sistemas para a supervisão, controle e operação nos processos industriais, atuando em tempo real e com grande confiabilidade. A fim de estudar e desenvolver testes de novos controles e modos de operação em todo o mundo, centros de pesquisa estudam os processos da indústria petroquímica usando *softwares* capazes de simular o comportamento dos sistemas, processos e equipamentos de plataformas e refinarias, segundo suas características e especificidades. Todas estas aplicações de *software* têm uma semelhança: cada uma a seu modo, usam as informações obtidas dos equipamentos contidos nas plantas industriais para realizar suas funções em suas plataformas.

Devido à complexidade dos processos na indústria petroquímica, em geral, uma planta industrial é composta de dezenas de classes de equipamentos e centenas de dados e variáveis de processo a serem compartilhadas. Para a utilização das plantas a serem operadas, repetidamente, os mesmos equipamentos são representados de maneira divergente entre diversas aplicações nas plataformas de *softwares*. Estas visões distintas de uma mesma planta podem ocasionar o comprometimento da interoperabilidade entre as aplicações, acarretando em erros de comunicação e de controle que, por sua vez, compromete a segurança da planta.

Diferentes *softwares* com objetivos variados compartilham coletivamente das informações de dados reais dos equipamentos da planta. Independentemente da linguagem de programação usada em cada plataforma, uma prática bem aceita na comunidade é a representação dos equipamentos por meio da orientação a objetos, em que cada equipamento real da planta é um objeto, instância de uma classe (BECKER; JR; PEREIRA, 1999). A classe é a estrutura que representa um conjunto de objetos com aspectos semelhantes, abstraindo suas características e estados (atributos), e seu comportamento (métodos). O padrão internacional ISO-9506 para comunicação entre dispositivos e

aplicações de controle e supervisão por meio da especificação *Manufacturing Message Specification* (MMS), já tinha a proposta de usar uma estruturação das informações orientada a objetos mediate os *Virtual Manufacturing Device* (VMD) (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2003).

1.1 CONTEXTUALIZAÇÃO

O Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC) possui uma parceria com o Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello (Cenpes) para desenvolvimento de projetos de pesquisa na área de controle e otimização dos processos de produção no âmbito petroquímico. O presente trabalho foi desenvolvido dentro do escopo do projeto *Desenvolvimento de Algoritmos de Controle Preditivo Não Linear e de Avaliação de Desempenho de Controladores Preditivos para Plataformas de Produção de Petróleo - Fase 2*, que tem como objetivo desenvolver estratégias de controle avançado para plataformas de produção utilizando algoritmos de controle preditivo e também a otimização da produção. O foco principal da equipe do projeto é o estudo de estratégias de controle, mesmo assim, há diversas demandas na área de *software* para a implementação das estratégias de controladores criadas o que facilita o desenvolvimento de aplicações. Como resultado, há o aumento da produtividade, compreensão dos problemas, reuso de soluções e, também, redução do tempo de desenvolvimento de uma nova aplicação ou alteração necessária em uma já existente.

1.2 MOTIVAÇÃO

É comum haver em um processo industrial diversas operações e várias possibilidade de plataformas de operações, como, sistemas supervisórios, aplicações de controle, operações básicas de equipamentos, simuladores de processos, sistemas de gerenciamento e outros. Ocorre na atualidade, uma dificuldade no desenvolvimento de projetos em plantas industriais que é a necessidade de modelar uma mesma planta industrial para cada plataforma de *software* que irá operá-la. Carece que os projetistas de cada âmbito de aplicação tenha conhecimento sobre as características e comportamentos de cada processo e componente que compõem a planta para realizar a modelagem específica para objetivo de sua plataforma. Seguindo assim, o projetista especialista de plataforma também necessita ser especialista de domínio, com conhecimentos dos componentes a serem modelados no domínio de cada aplicação. A quantidade de equipamentos, geralmente atinge centenas de pontos de controle, instâncias de equipamentos, e variáveis de

processos e atributos disponibilizados nos bancos de dados dos servidores. Mais ainda, para cada plataforma de *software*, a modelagem de uma mesma planta industrial é refeita para cada plataforma de *software* utilizando sua linguagem própria correspondente, e, por vezes, ocorre de um equipamento ser modelado mais de uma vez, distintamente, dentro de aplicações diferentes de uma mesma plataforma. O resultado desse processo são definições de equipamentos altamente específicas para o domínio de uma aplicação e dependentes de uma única plataforma. Todos esses aspectos, consequentemente, geram dificuldades na manutenção, modificação e reutilização dos modelos produzidos, aumento de custos de tempo de desenvolvimento e retrabalho de modelagens de um mesmo domínio (DAMO, 2019).

Naturalmente, em um cenário que se tem uma gama de plataformas e diversas aplicações para uma mesma planta industrial, o desenvolvedor, por exemplo, que esteja projetando um controle de vazão de um sistema de compressão, pode ter uma visão divergente dos equipamentos do processo que o desenvolvedor da aplicação de monitoramento de riscos do mesmo processo. Visões distintas dos desenvolvedores proporcionam distintos modelos de um mesmo equipamento o que, consequentemente, ocasiona erros notáveis de cálculos, ações e respostas inesperadas, colocando em risco toda a operação e controle das plantas. Divergência das informações e modelagem também coloca em ameaça a interoperabilidade entre as plataformas.

Tomando por ponto de partida essas dificuldades, há a necessidade de adoção de uma metodologia de desenvolvimento dos equipamentos para aplicações de *softwares* de supervisão, controle e operação de plantas industriais que simplifique a programação, promova a reutilização entre plataformas e facilite a manutenção de aplicações existentes. Possibilitando que as plataformas distintas tenham a mesma perspectiva de um dado sistema e que, consequentemente, gere um aumento da compreensão e produtividade do desenvolvedor e uma melhora na exatidão e confiabilidade das soluções implementadas.

Pensando nessa necessidade, Damo (2019) desenvolveu uma infraestrutura chamada M4PIA utilizando a abordagem de MDE (*Model-Driven Engineering* - Engenharia Dirigida por Modelos). A MDE é um paradigma de desenvolvimento de *software* com enfoque nos modelos de especificação do domínio, que permite melhorar a produtividade, a compreensão dos sistemas e a facilidade de manutenção e evolução (SCHMIDT, 2006; SELIC, 2003). A proposta do paradigma é que as aplicações sejam descritas por intermédio de modelos em diferentes níveis de abstração utilizando padrões estabelecidos, como, por exemplo a linguagem *Unified Modeling Language* (UML) (UML; MOF, 2011) e *Model-Driven Architecture* (MDA) (SOLEY et al., 2000; OBJECT MANAGEMENT GROUP, 2001).

A MDE, além de ser um paradigma de projeto, tem a capacidade de gerar modelos que podem ser interpretados por ferramentas de automação que criam sistemas, esqueletos de código e testes para múltiplas plataformas alvos (SELIC, 2003). A MDE inicia-se com uma modelagem do sistema em um alto nível de abstração, o que gera o *Platform Independent Model* (PIM), que pode ser automaticamente transformado pela transformação M2M (*Model-to-Model* - Modelo para Modelo) em um *Platform Specific Model* (PSM), sobre o qual será executado o refinamento com as características da plataforma alvo específica. O PSM podem, então, ser transformado em código-fonte da aplicação para a plataforma específica, através de uma transformação *Model-to-Text* - Modelo para Texto (M2T). Com MDE as definições das camadas mais abstratas fornecem bases formais para estruturação dos modelos de camadas inferiores, sendo possível reduzir as decisões de projeto e construir artefatos de maneira automática a partir dos modelos, diminuindo tempos e custos no processo de desenvolvimento (DAMO, 2019; BRAMBILLA; CABOT; WIMMER, 2017). A infraestrutura M4PIA desenvolvida por (DAMO, 2019) propõe um modelo de alto nível de abstração para realização de uma modelagem de um sistema petroquímico industrial que seja possível a geração automática de código-fonte uma para plataforma específica. Este é um único sentido possível a ser realizado pelo M4PIA, do mais alto nível de abstração até o mais especializado.

Ocorre que o processo de MDE visa criar um método de desenvolvimento baseado em modelos, entretanto na indústria petroquímica, existem inúmeros sistemas legados que estão em atuação por muitos anos. Esses sistemas, eventualmente, exigem a necessidade de manutenções e até mesmo de adaptações para manter seu funcionamento. O mesmo esforço de trabalho explicitado para a criação retorna para o desenvolvimento da manutenção ou reengenharia de um sistema. Para que esses sistemas possam contemplar as vantagens providas pelo paradigma da MDE, é preciso o desenvolvimento de uma Engenharia Reversa (RE). A aplicação de RE em MDE é chamada de MDRE (*Model-Driven Reverse Engineering* - Engenharia Reversa Dirigida por Modelos). A MDRE inicia seu processo de desenvolvimento do resultado final da plataforma de *software* (sistema de legados, documentação, código-fonte), ou seja, do mais baixo nível de abstração até o mais alto nível, tomando o sentido contrário ao paradigma proposto pela MDE. Os níveis de abstração aumentam por meio de transformações, onde, sendo a origem do processo o código-fonte de uma aplicação, ocorre a transformação T2M (*Text-to-Model* - Texto para Modelo) que tem por resultado um nível PSM. A partir, de um modelo PSM há uma segunda transformação, desta vez, a transformação M2M que, finalmente, resulta no alto nível de abstração de um modelo independente de plataforma PIM (RUGABER; STIREWALT, 2004).

Este trabalho motiva-se ao desenvolvimento de MDRE para complementação das funcionalidades da infraestrutura M4PIA para aplicações de controle, operação e simulação na indústria petroquímica gerando assim, a reutilização entre as plataformas, aumentando a confiabilidade das soluções e manutenções, e a redução dos custos e tempo de desenvolvimento.

O M4PIA é uma infraestrutura de alto nível de abstração modelada a partir de dois *softwares* de desenvolvimento para sistemas industriais petroquímicos utilizadas pela Petrobras e Cenpes em suas plantas petroquímicas e projetos de pesquisa. O escopo deste trabalho se limita a duas plataformas de *softwares* que são suportadas pela infraestrutura:

- Módulo de Procedimentos Automatizados (MPA): plataforma de desenvolvimento com foco em automação de plataformas de petróleo, sendo um sistema de desenvolvimento e execução de aplicações de controle e automação de processos industriais;
- *Environment for Modeling, Simulation and Optimization* (EMSO): plataforma de simulação de processos dinâmicos baseada em equações.

A partir de uma modelagem desenvolvida no modelo de mais alto nível M4PIA, pode-se, por meio de uma transformação *Model-to-Model* - Modelo para Modelo (M2M), obter um modelo especializado para a plataforma de domínio MPA ou, para a plataforma de domínio EMSO. A partir destes modelos específicos, pode-se obter a geração automática de código-fonte para a plataforma MPA em linguagem LUA ou o código-fonte da plataforma EMSO em linguagem de domínio próprio denominada pelo mesmo nome EMSO.

1.3 OBJETIVOS

Dado o contexto e as motivações deste trabalho, foram definidos os objetivos geral e específicos desta dissertação.

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é o desenvolvimento de uma engenharia reversa nas plataformas de *software* MPA e EMSO afim de adicionar funcionalidades à infraestrutura M4PIA (*Model-Driven Engineering for Petrochemical Industry Automation*) para desenvolvimento de aplicações de simulação, operação, controle e supervisão de plantas industriais, especificamente para a indústria petroquímica.

1.3.2 Objetivos Específicos

Para que o objetivo geral seja satisfeito, foram definidos os seguintes objetivos específicos:

- definir o processo de construção de engenharia reversa para a infraestrutura M4PIA;
- especificar as transformações entre os modelos de nível específico de plataforma PSM para modelos de nível independente de plataforma PIM para cada plataforma associada (MPA e EMSO) - M2M;
- especificar as transformações entre os códigos-fonte para modelos de nível específico de plataforma PSM para cada plataforma associada (MPA e EMSO) - T2M;
- expor uma prova conceito com um exemplo de aplicação da infraestrutura para análise da expressividade dos metamodelos e das funcionalidades propostas;
- analisar o desempenho de transformação da infraestrutura M4PIA para os sistemas de legados;
- estabelecer as limitações e planejar as melhorias a serem realizadas em trabalhos futuros.

1.4 ORGANIZAÇÃO DO DOCUMENTO

A organização do trabalho se dá da seguinte maneira: no capítulo 2 são descritas as tecnologias relacionadas a este trabalho e revisão da literatura a trabalhos relacionados a proposta. No capítulo 3 é a apresentada a proposta de engenharia reversa para a infraestrutura com seus artefatos e utilização. No capítulo 4, apresenta-se uma prova conceito e uma avaliação para proposta. Finalmente, o capítulo 5 expõe comentários finais sobre o trabalho com conclusões e sugestões para trabalhos futuros.

2 TECNOLOGIAS E TRABALHOS RELACIONADOS

O presente capítulo apresenta os conceitos básicos de metodologias e tecnologias utilizadas para o desenvolvimento da dissertação. Os conceitos apresentados são relacionados a *Model-Driven Engineering* - Engenharia Dirigida por Modelos (MDE), *Model-Driven Reverse Engineering* - Engenharia Reversa Dirigida por Modelos (MDRE), também discorre sobre as ferramentas e linguagens escolhidas para o desenvolvimento da proposta bem como as plataformas de software utilizadas. Mais ainda, apresenta a ferramenta *Model-Driven Engineering for Petrochemical Industry Automation* (M4PIA) base de todo o trabalho.

2.1 MDE - ENGENHARIA DIRIGIDA POR MODELOS

A *Model-Driven Engineering* - Engenharia Dirigida por Modelos (MDE) (KENT, 2002; SCHMIDT, 2006), é um método de desenvolvimento de sistemas que permite construir uma aplicação usando modelos. Variadas DSL (*Domain Specific Language* - Linguagens de Domínio Específico) podem ser usadas para representar pontos de um sistema no decorrer de sua construção. MDE tem por objetivo ter o foco do desenvolvimento de *software*, normalmente centrado na programação, para o processo de modelagem do sistema, em que os modelos independentes de plataforma são transformados em modelos orientados à implementação (SELIC, 2003). Nesse método, os modelos, são os principais resultados do processo e não os programas. Os programas são automaticamente gerados a partir dos modelos (SOMMERVILLE, 2011).

De acordo com Sommerville (2011), a modelagem de um sistema consiste em elaborar modelos que descrevam sua características ou comportamentos. Um modelo é uma abstração, ou seja, uma simplificação que foca no que é importante em determinado contexto e ignora o restante. Em uma definição, modelo é uma especificação formal uma função, estrutura ou característica de um sistema.

A MDE proporciona um grande aumento de produtividade no desenvolvimento de sistemas. A qualidade dos artefatos se *software* também tende a aumentar, visto que são gerados a partir de modelos, desenvolvidos com base em metamodelos que especificam conceitos do domínio do problema, e pelo uso de regras de transformação, que favorecem a produção automática do *software*. Outrossim, os modelos podem ser mapeados em diferentes plataformas permitindo a reutilização da modelagem e a consequente redução do esforço de projeto (YUSUF; CHESSEL; GARDNER, 2006).

A definição dos modelos pela MDE, naturalmente se da separando a especificação das suas funcionalidades e de uma implementação das funcionalidades do sistema em

uma plataforma específica. Um ponto importante é que os modelos podem ser gerados em diferentes níveis de abstração separados em: modelos independentes de plataforma conhecidos por *Platform Independent Model* (PIM) e modelos específicos de plataforma chamados de *Platform Specific Model* (PSM). O PIM é uma especificação que abstrai detalhes técnicos de um dado sistema. O PSM, por sua vez, especifica como a funcionalidade descrita no PIM é feita em uma dada plataforma (BRAMBILLA; CABOT; WIMMER, 2017).

Um processo que utiliza a MDE para o desenvolvimento de um sistema, destaca Kurtev (2005), necessita está associado a uma cadeia de transformações que recebe modelos fonte, em um alto nível de abstração, e os transforma em modelos alvo, em níveis mais baixos de abstração, até chegar ao código fonte. A cadeia de transformações de modelos é tem um papel de destaque na abordagem de MDE, uma vez que viabiliza automação do processo.

O foco está em não necessitar programar manualmente as aplicações, mas reutilizar os modelos criados para gerar automaticamente o devido código. Desse modo, Os projetistas podem focar no cerne do problema, expressando os conceitos do domínio efetivamente, reduzindo o tempo desperdiçado com as complexidades da plataforma de implementação (SCHMIDT, 2006). Para Sommerville (2011), esses aspectos são as principais vantagens de usar a MDE, dado que reduzem a probabilidade de erros, aumentam os processos de implementação e possibilitam o invento de modelos reutilizáveis das aplicações. Mais ainda, facilita a adaptabilidade do sistema para uma nova tecnologia de implementação sendo preciso apenas desenvolver o mapeamento dos conceitos para a nova plataforma.

2.1.1 Modelos

Os modelos de um determinado sistema são especificados a partir de metamodelos. Estes, por sua vez, também são modelos definidos em uma linguagem de modelagem através da restrição e/ou extensão dos construtores da linguagem para representar um espaço de possibilidades de aplicações de um ou mais domínios. Um metamodelo tem a semântica e a sintaxe primordial para a construção dos modelos, assim dizendo, os construtores disponíveis da linguagem de modelagem, e as suas relações e regras de modelagem (VÖLTER et al., 2013). Na ocasião em que um modelo é especificado a partir de um metamodelo, ele representa uma instancia daquele metamodelo. Dando um exemplo, pode-se desenvolver metamodelo para representar uma plataforma de automação e a partir deste metamodelo pode-se instanciar modelos para representar

um sistema específico da plataforma de automação. Analogamente aos modelos, os metamodelos também podem ser descritos a partir de metametamodelos. A Figura 2.1 ilustra como se da essa arquitetura.

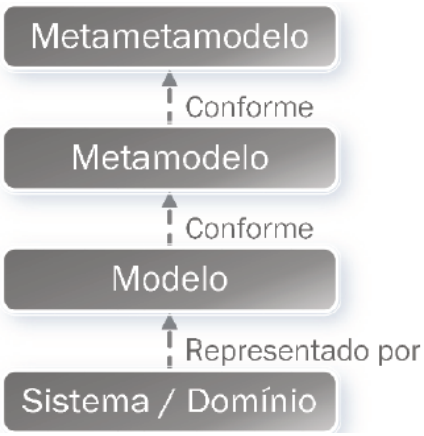

Figura 2.1: Arquitetura de desenvolvimento baseado em modelos (BRAMBILLA; CABOT; WIMMER, 2017).

Os modelos devem obedecer a uma relação de conformidade com seus metamodelos. Um modelo é dito conforme com seu metamodelo se estiver sintaticamente correto e atender às restrições impostas pelo metamodelo. Já que os modelos são uma especificação de uma aplicação, eles precisam satisfazer as propriedades da aplicação, em outras palavras, têm que ser corretos em relação aos requisitos dessa aplicação. Essa relação de correção não é garantida pela relação de conformidade, visto que os metamodelos definem cada aplicação. A garantia de correção os modelos exige o uso de técnicas de validação e verificação, ocorrentes em outros processos de desenvolvimento de *software* (BECKER et al., 2018).

2.1.2 Transformações

A transformação de modelos consiste numa operação de conversão entre modelo fonte (que é lido para ser transformado) e modelo alvo (que é gerado pela execução da transformação). Em MDE, os modelos são transformados em outros modelos mais específicos até que se chaga ao código fonte da aplicação, através de uma cadeia de transformações. Uma transformação é definida ao nível dos metamodelos e inclui um conjunto de regras que descreve como os modelos fonte conseguem ser transformados em modelos alvo. Estas regras mapeiam estruturas do metamodelo do modelo fonte em estruturas do metamodelo do modelo alvo (BECKER et al., 2018).

As transformações se classificam em dois tipos: primeiro, *Model-to-Model* - Modelo para Modelo (M2M) quando o um modelo fonte é transformado em um modelo alvo e,

segundo, *Model-to-Text* - Modelo para Texto (M2T) quando partindo de modelos fonte é gerado como saída o código de implementação do sistema.

Tipicamente, um processo começa com o desenvolvimento do modelo genérico PIM, ou a instanciação do modelo conceitual, atentando a cada estrutura que compõe o metamodelo PIM. O modelo PIM serve como entrada para o mapeamento que gera o modelo específico PSM podendo ser transformado em modelos para múltiplas plataformas. O modelo PSM, em sua posição, serve como entrada para a transformação que terá por resultado o código fonte da aplicação na linguagem de programação específica pretendida para a plataforma. A Figura 2.2 exemplifica o curso de desenvolvimento.

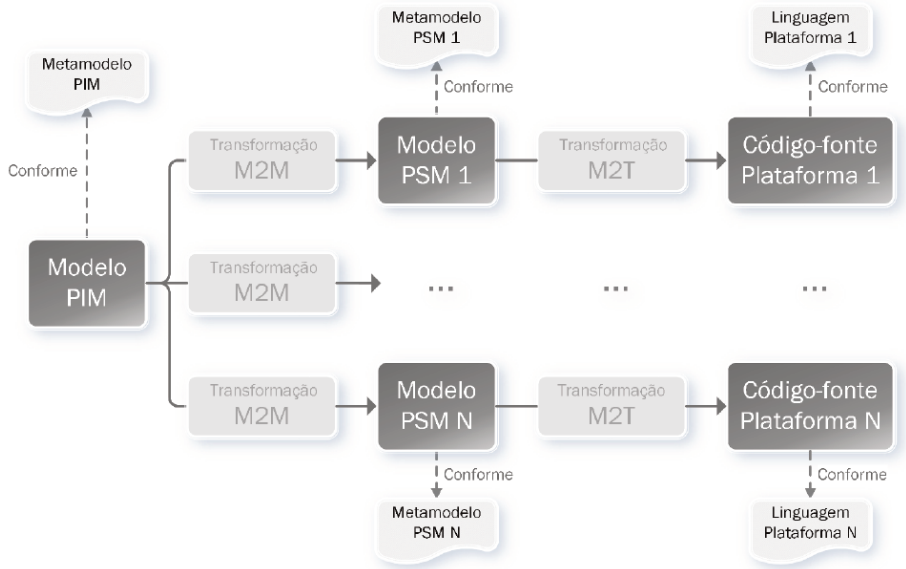

Figura 2.2: Curso de desenvolvimento de aplicações por MDE (BRAMBILLA; CABOT; WIMMER, 2017).

2.2 MDRE - ENGENHARIA REVERSA DIRIGIDA POR MODELOS

Como destacado anteriormente, a metodologia MDE consiste em utilizar de transformações partindo de um modelo alvo para o modelo fonte. O objetivo é que a partir dessas transformações, tenha-se ao final da cadeia de transformações o código fonte do sistema ou aplicação. Sendo assim, o resultado final é gerado com o intuito de escrever manualmente a menor quantidade de código, ter menos esforço e possivelmente menos erros (AMARAL; MERNIK, 2016). Mas se o objetivo for realizar o trajeto inverso? Se agora, deseja-se partir do código fonte para um modelo em mais alto nível de abstração? Para esse contexto, a solução mais viável seria a aplicação de Engenharia Reversa (RE).

Engenharia Reversa é o processo é compreender software e, a partir disso, é possível produzir um modelo em um alto nível de abstração, adequado para documentação, manutenção ou reengenharia (RUGABER; STIREWALT, 2004; BRAMBILLA; CABOT; WIMMER, 2017). Conforme Raibulet, Fontana and Zanoni (2017), a RE pode ser aplicada em diversos contextos. Os cenários mais típicos são para a compreensão de códigos, manutenção, evolução do *software*, integração ou interface com sistemas legados. Outro possível cenários é a atualização da documentação de um sistema após as alterações do código fonte. Mais possibilidades a serem exploradas são as migrações de sistemas legados para novas plataformas ou para novos paradigmas de desenvolvimento, por exemplo, para um sistema existente com o uso de MDE.

A combinação de Engenharia Reversa com MDE é chamada de *Model-Driven Reverse Engineering* - Engenharia Reversa Dirigida por Modelos (MDRE). MDRE pode ser tida como a produção de modelos descritivos de sistemas existentes que foram produzidos previamente de alguma maneira (FAVRE, 2005).

Segundo Raibulet, Fontana and Zanoni (2017), MDRE inicia com o conhecimento do código fonte, assim a solução do sistema começa do mais baixo nível de abstração, o próprio código fonte, e busca construir vistas em níveis de abstração mais altos. A geração de modelos a partir do código fonte e transformações podem ser automatizadas. Após a geração, os modelos podem ser utilizados para serem analisados ou a para iniciar uma nova fase de desenvolvimento orientada por modelo.

As abordagens e ferramentas de RE usam modelos para abstrair o sistema, o que é natural, visto que para representar o sistema sob análise de forma mais abstrata é necessário algum tipo modelo. A diferença no MDRE é que todo processo é baseado no uso sistemático de modelos para representar informações e no uso de metamodelos para descrever esses modelos. Para mais, a transformação M2M é realizada explicitamente, além do processo de descoberta, por exemplo a transformação T2M (*Text-to-Model* - Texto para Modelo).

As soluções MDRE são escaláveis, adaptáveis, portáteis e reutilizáveis. Também se beneficiam da extensibilidade, integração, cobertura, podem ser genéricas, sendo uma ótima assistência para Engenharia Reversa (SABIR et al., 2019).

2.3 FERRAMENTAS DE IMPLEMENTAÇÃO

Sommerville (2011) afirma que é necessário o uso de várias ferramentas para desenvolver etapas dos projetos com abordagem dirigida a modelos como, por exemplo, a modelagem, a metamodelagem, criar regras de mapeamento e realizas as transformações.

A complexidade dessas ferramentas é a principal causadora da pouca utilização dessa abordagem na engenharia de *software*. Entretanto, também são por elas que o desenvolvimento ocorre em menor tempo gerando aumento de produtividade e redução de custos.

As ferramentas escolhidas para o desenvolvimento do trabalho dentre as opções disponíveis distingue-se o *Eclipse Modeling Framework* (EMF) (STEINBERG et al., 2008) por possuir a maior quantidade de componentes e projetos relacionados à MDE e também a *ANother Tool for Language Recognition* (ANTLR) (PARR, 2013) para o desenvolvimento da Engenharia Reversa.

2.3.1 EMF - *Eclipse Modeling Framework*

Eclipse Modeling Framework (EMF) é a ferramenta principal do *Eclipse Modeling Project*, que abrange as ferramentas de modelagem desenvolvidas para o ambiente Eclipse, uma plataforma de código aberto e que contém uma grande comunidade de desenvolvedores e usuários. O conjunto completo de ferramentas MDE necessária para o desenvolvimento dentro do mesmo ambiente como: editores, modelos, metamodelos, definição e execução de transformação M2M e M2T. Nada obstante, EMF não possui suporte para a RE sendo necessária a utilização de uma outra ferramenta.

O *Eclipse Modeling Framework* (EMF) possibilita a definição de modelos usando linguagem Java ou UML (*Unified Modeling Language*), no entanto seu foco é o uso de seu metamodelo *Ecore*, tecnologia criada para especificar metamodelos. Ao desenvolver a metamodelagem com base no *Ecore*, o mesmo passa a ter função de metametamodelo. Por meio de diversas ferramentas, EMF possibilita a criação de editores baseados nos metamodelos, para ser feita a modelagem das aplicações de domínio.

EMF fornece provedor de persistência para XMI (*XML Metadata Interchange*) e XML (*eXtensible Markup Language*). Por padrão, o EMF usa XMI que é um padrão da OMG (*Object Management Group*) (OBJECT MANAGEMENT GROUP, 2015) para troca de informações de metadados por meio de *eXtensible Markup Language* (XML). Existem outras tecnologias desenvolvidas em cima dos modelos EMF que têm relacionamentos com alguns outros padrões OMG para modelagem de dados:

- **MOF (*Meta-Object Facility*)** (OBJECT MANAGEMENT GROUP, 2016b): linguagem para gerenciamento e definição de metamodelos. A linguagem de modelagem UML é definida com MOF, similar ao Ecore que também é baseado em um subconjunto do padrão MOF.

- **OCL (*Object Constraint Language*)** (OBJECT MANAGEMENT GROUP, 2014): linguagem padrão de expressões para especificar restrições nos modelos baseados no padrão MOF. Utilizada também para especificar outras regras do modelo do sistema, por exemplo, condições invariantes no modelo, pré e pós-condições das operações e como uma linguagem de consulta.

- **QVT (*Query/View/Transformation*)** (OBJECT MANAGEMENT GROUP, 2016a): linguagem para definição das transformações entre MOF. Usa expressões OCL para definir operações e consultas sobre o modelo. É composta por duas linguagens declarativas (*Relations/Core Mapping*) e uma linguagem imperativa (*Operational Mapping*).

- **Mof2Text (*MOF Model to Text Language*)** (OBJECT MANAGEMENT GROUP, 2008): padrão para definição das transformações de modelos MOF para artefatos de texto como código fonte e relatórios. Também faz uso de expressões OCL para definição de consultas e operações sobre o modelo.

Existem também diversas outras tecnologias que desenvolvem ferramentas, linguagens e *frameworks* baseados em MDE. Vale destacar o projeto QVTo (QVT *Operational Mappings*) utilizado para desenvolvimento deste trabalho de transformações *Model-to-Model* - Modelo para Modelo (M2M).

Eclipse QVT Operational é a implementação da linguagem QVTo (QVT *Operational Mappings*), o componente imperativo da QVT. Parte da transformação *Model-to-Model* tem por alvo viabilizar modificações e transformações de modelos, operando sobre o modelo EMF. As transformações QVT são unidirecionais, possuem um modelo fonte e um modelo alvo, cada qual referenciado na sua definição e de acordo com seu metamodelo. O mapeamento é realizado de um ou mais elementos do modelo fonte para um ou mais elementos do modelo alvo, usando expressões OCL para condições e operações em ambos os modelos.

2.3.2 ANTLR - *ANother Tool for Language Recognition*

O *ANother Tool for Language Recognition* (ANTLR) é um gerador de analisador para ler, processar, executar ou traduzir texto estruturado ou arquivos binários. É amplamente usado para construir linguagens, ferramentas e estruturas. A partir de uma gramática, ANTLR gera um analisador que pode construir e percorrer árvores de análise. A utilização do ANTLR também pode se dar para simplificar a criação de transformações

capazes de interpretar os códigos fontes e extrair as informações relevantes para objetos lógicos, em concordância com o seu respectivo metamodelo.

Primeiramente, serão brevemente explicados alguns conceitos relacionados á interpretação de arquivos textuais para facilitar o entendimento da ferramenta ANTLR. O elemento principal de tais programas é a "linguagem", que pode ser definida como um conjunto de sentenças válidas em uma estrutura, criadas a partir de frases, subfrases e símbolos de vocabulário (PARR, 2013). O conceito de "linguagem"aqui é dissociado das linguagens na naturais (definidas pelas regras de cada idioma) e verificado sua existência nas regras que regem linguagens de programação ou estruturas de dados.

Na literatura, programas que são capaz de reconhecer textos em uma determinada linguagem são chamados de *parser*. Claramente, esses programas são específicos para cada linguagem, já que dependem do conhecimento das regras que regem as construções sintáxicas, por exemplo, sendo capazes de verificar a validade de cada sentença. Fazendo um paralelo com linguagem natural, significa dizer que compreender a frase 'Eu tenho feito o jantar' é uma frase em primeira pessoa, no tempo verbal pretérito perfeito de acordo com as regras da Língua Portuguesa. Analogamente, 'int k = 7;' escrito em um arquivo Java é uma atribuição de variável.

Em linguagem natural, a identificação dessas construções realizadas pelo *parser* é similar ao processo que usamos para ler uma frase em um livro. Anteriormente, é necessário identificar os caracteres que compõem cada palavra. Essa etapa é denominada de *lexer* ou *tokenizer*, e computacionalmente corresponde a identificar, a partir de uma *stream* de caracteres, quais são as palavras ou tokens válidos para aquela determinada linguagem.

Exemplificando em linguagem natural, seria possível identificar agrupamentos de caracteres como Palavras, Espaços, Vírgulas e outros sinais de pontuação na análise de um *lexer*, enquanto o *parser* corresponderia a identificar a função de cada desses elementos nas frases, por exemplo, em Sujeito, Verbo, Adjetivo, além da categorização da própria construção verbal.

A Figura 2.3 ilustra um fluxo típico de uma aplicação de análise sintática. O arquivo é interpretado pelo *lexer* caractere a caractere gerando uma *stream* de *tokens* que identificam conjuntos léxicos válidos para a linguagem. Como ilustrado no exemplo, 'sp' corresponde a um `ID`, '=' a um `Igual`, '100' simboliza um dado `Inteiro` e ';' é identificado como `EndOfLine`.

Uma vez identificadas pelo *lexer*, as sequências de *tokens* são então categorizadas pelo *parser* através de regras específicas da linguagem. Na Figura 2.4 é possível verificar que existe uma regra *assing* na definição da linguagem que corresponde à sequência

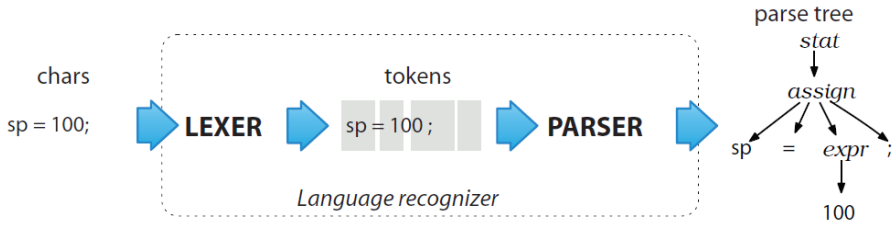

Figura 2.3: Exemplo de fluxo de análise sintática do ANTLR (PARR, 2013).

de *tokens*: `Id -> Igual -> Expr -> EndOfLine`. Interessante notar que enquanto, `Id`, `Igual`, e `EndOfLine` são *tokens*, `Expr` corresponde a uma nova regra de linguagem, neste caso, pode ser composta por um *token* `Inteiro` ou por outras sequências de *tokens* não apresentadas na imagem.

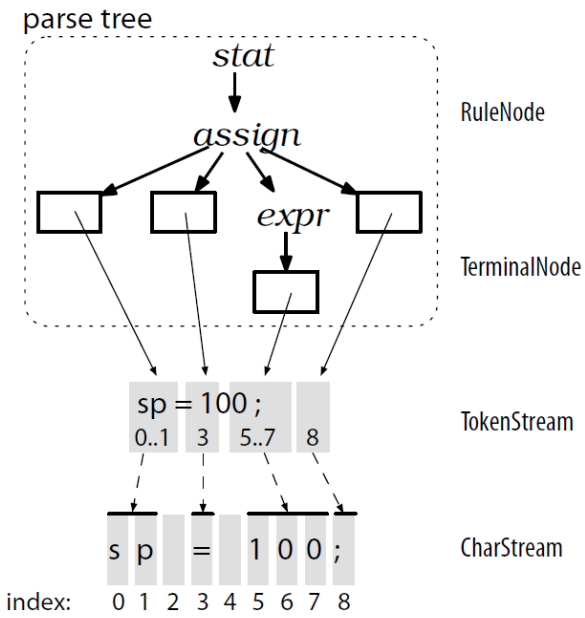

Figura 2.4: Exemplo de fluxo de análise sintática do ANTLR (PARR, 2013).

Sendo assim, o resultado a etapa de análise do *parser* é uma *Abstract Syntax Tree* (AST), uma estrutura em árvore que identifica as expressões encontradas no arquivo de entrada, em diferentes níveis, onde cada 'nó' pode ser caracterizada por uma nova regra de linguagem e cada 'folha' é obrigatoriamente um único *token*.

Portanto, o ANTLR é útil para a construção de ferramentas de reconhecimento de linguagens que proporciona facilidades e geração automática de transformações. O usuário deve desenvolver gramáticas de *lexer* e *parser*, definindo regras de tokerização e de construção de nós da árvore de arquivos.

As gramáticas são definidas em arquivos com extensão **.g4** e podem ser compiladas para gerar automaticamente classes que executam a operação de leitura e identificação de arquivos compatíveis coma gramática. O próprio ANTLR gera funções

de entrada e saída de cada nó da árvore, na foram de *Listener*. O desenvolvedor pode sobrescrever essas funções e identificar o contexto, realizar operações ou verificar a concordância com padrões preestabelecidos. Tais funções podem ser adaptadas para a criação de transformações T2M (*Text-to-Model* - Texto para Modelo).

2.4 INFRAESTRUTURA M4PIA

Uma problemática encontrada a industria petroquímica quando se trata de sistemas de automação é o uso de vários *softwares* diversos e, a maioria, com seu padrão e linguagens próprias. Cada *software* usufrui dos dados coletados dos equipamentos presentes na planta industrial para realizar suas funções e, muitas vezes, os mesmos equipamentos são representados de modos distintos entre diferentes aplicações nas plataformas de *software*. Devido a complexidade nos processos das plantas industriais petroquímicas ser altíssima, é comumente adotada a composição de uma planta por meio de programação orientada a objetos, o que uma planta seja composta por várias classes de equipamentos e centenas de informações de processo a serem compartilhadas (BECKER; JR; PEREIRA, 1999).

Atualmente, uma planta industrial precisa ser modelada para cada plataforma de *software* que irá operar sobre a mesma. Os desenvolvedores devem ter ciência das aplicações que cada plataforma tenha e também conhecimento sobre as características e comportamentos de cada processo e equipamento para fazer sua modelagem específica para sua plataforma alvo. Desse modo, o especialista de plataforma também precisa ser especialista de domínio, com conhecimento dos componentes a serem modelados no âmbito de cada aplicação. Em cada plataforma de operação, também há bastantes equipamentos com diversos pontos de controle, instâncias, variáveis de processo e atributos de processos.

A modelagem de uma planta industrial precisa ser refeita para cada plataforma de *software*, em diferentes linguagens de programação e, dependendo da necessidade, um mesmo equipamento é modelado mais de uma vez de maneira distinta, dentro de diferentes aplicações de uma mesma plataforma. Definições de equipamentos específicos para o domínio de uma aplicação e dependentes de uma única plataforma, dificultam a manutenção, alteração e reutilização dos modelos produzidos, gerando custos de tempo de desenvolvimento e retrabalho de modelagens de um mesmo domínio (DAMO, 2019).

Como solução deste problema, Damo (2019) propôs uma infraestrutura baseada em MDE para o desenvolvimento de aplicações de simulação, operação, supervisão e controle de plantas industriais petroquímicas. A ferramenta chama-se M4PIA (*Model-*

Driven Engineering for Petrochemical Industry Automation) e foi desenvolvida abrangendo em seu escopo dois softwares: o primeiro, chamado de MPA (Módulo de Procedimentos Automatizados) utilizado na automação de plataformas de petróleo, sendo um sistema de desenvolvimento e execução de aplicações de controle e automação de processos industriais. O segundo software é o EMSO (*Environment for Modeling, Simulation and Optimization*) que é uma plataforma de simulação de processos dinâmicos baseado em equações.

A Figura 2.5 apresenta o fluxo e os artefatos que compõem a infraestrutura M4PIA: um metamodelo genérico PIM, M4PIA; dois metamodelos específicos de plataforma PSM, metamodelos MPA e EMSO; duas transformações M2M, modelo M4PIA para modelo MPA e M4PIA para modelo EMSO; e duas transformações M2T, modelo MPA para código fonte MPA e modelo EMSO para código fonte EMSO.

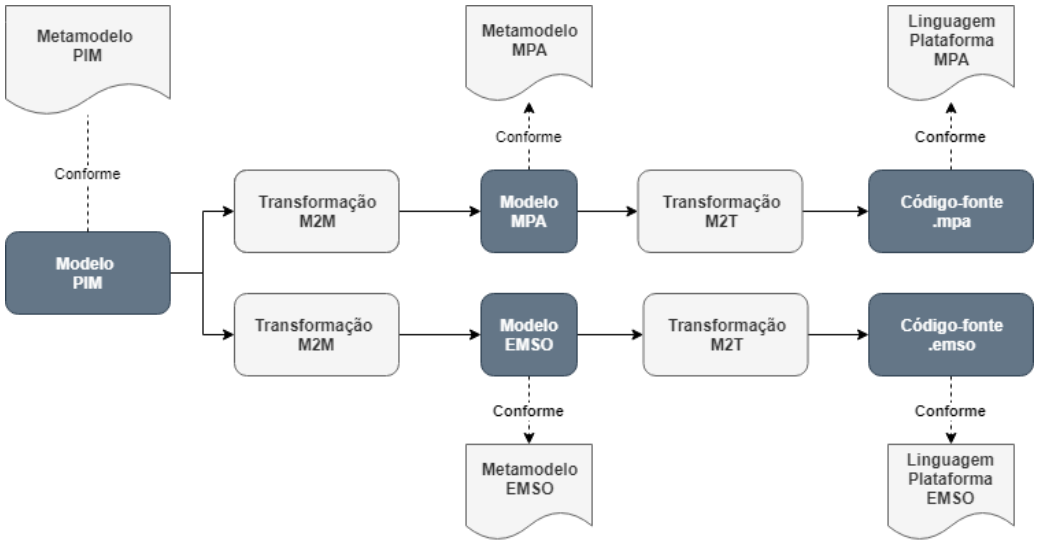

Figura 2.5: Exemplo do fluxo da infraestrutura M4PIA (DAMO, 2019).

O M4PIA parte de uma instância de maior abstração e por meio de transformação M2M chega em um modelo PSM do MPA ou EMSO e é possível fazer a transformação M2T afim de adquirir os códigos-fonte de cada ferramenta (MPA ou EMSO). A contribuição deste trabalho é integrar a funcionalidade de Engenharia Reversa, também denominada de reengenharia, ao M4PIA, atribuindo a possibilidade de transformações T2M partindo dos códigos-fonte para o modelo PIM do M4PIA. A aplicação da RE na infraestrutura possibilita a importação de códigos de legados para que códigos e bibliotecas de equipamentos existentes possam ser importados para o modelo M4PIA e disponibilizadas para todas a plataformas suportadas, ademais permite a reengenharia para modelos de mais alto nível, evitando a perda das informações adicionadas nas etapas de refinamento do modelo e código.

2.5 MPA - MÓDULO DE PROCEDIMENTOS AUTOMATIZADOS

O Módulo de Procedimentos Automatizados (MPA) (SATUF; PINTO, 2009) (INSTITUTO TECGRAF/PUC-RIO, 2020) é um sistema desenvolvido como foco em automação de plataformas de petróleo e gás, sendo uma plataforma de desenvolvimento e execução de aplicações de controle e automação de processos industriais. A plataforma desenvolvida sob encomenda da Petrobras pelo Instituto Tecgraf de Desenvolvimento de Software Técnico-Científico da PUC-Rio (Tecgraf) e também contou com a participação da Petrobras no seu desenvolvimento. Originalmente, foi criada como um módulo de partida assistida de uma plataforma de extração de petróleo no Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello (Cenpes), centro de pesquisa da Petrobras (GUISASOLA; MAIA, 2009). O MPA tem por automatizar as operações manuais através da descrição dos procedimentos operacionais padrões, executados para garantir o desempenho de determinada operação ou situação.

A plataforma contém uma linguagem *Sequential Function Chart* (SFC), ou melhor, uma linguagem gráfica de fluxogramas, definida especialmente para o MPA, para a definição de procedimentos de monitoração, diagnósticos e atuação sobre plantas industriais. Atualmente, o MPA tem sido usado em várias aplicações como, por exemplo, controle de níveis de vasos por banda, controle flotador e proteção antigolfadas (DAMO, 2019).

A composição do MPA se dá por um servidor de execução e um aplicativo de configuração e gerência. No aplicativo é realizada a modelagem das plantas industriais através de objetos e são definidos os diagramas a serem executados. O servidor é responsável pela execução das manobras de operação configuradas nos diagramas, fazendo a interação com o sistema supervisório por meio de uma ponte de comunicação específica. O sistema supervisório permite a manipulação dos equipamentos por meio de pontos de controle (variáveis lógicas, numéricas ou textuais).

O MPA tem a capacidade de atuar como um operador do sistema supervisório, monitorando as variáveis do processo e agindo para controlá-las, podendo, então, manipular diferentes descrições de processos: os equipamentos que compõem a planta, através de seus pontos de controle; e os procedimentos de automação do sistema, mediante as manobras de operação.

A Figura 2.6 apresenta as etapas de execução do MPA mais a relação com o sistema supervisório. As etapas podem ser divididas em três: pré-configuração, configuração e execução.

A primeira etapa, chamada de **Pré-Configuração** consiste no uso ou programação

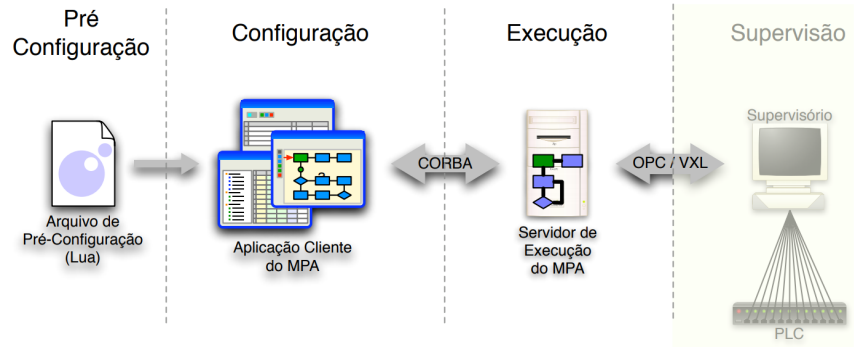

Figura 2.6: Etapas de desenvolvimento do MPA (PARR, 2013).

do arquivo de pré-configuração, onde se define as classes de equipamentos e funções auxiliares. Nas classes de equipamentos define-se atributos que especificam quais pontos de controle cada tipo de equipamento possui, podendo determinar características do equipamento ou seu relacionamento com outros equipamentos. Mais ainda, são definidas as funções para manipulação dos equipamentos, as operações sobre equipamentos.

Nas funções auxiliares são implementadas funcionalidades relacionadas à plataforma de execução, como o acesso a hora atual, sistema de arquivos e outras funções do sistema operacional, como funções mais gerais, cálculos de somatório e média a partir de uma lista de valores.

Tanto o próprio MPA como o arquivo de pré-configuração são programados em linguagem Lua, mas ainda é possível o uso de funções externas, provenientes de DLLs (em linguagem C ou C++), associadas a operações sobre equipamentos ou funções auxiliares. Mas para tal, é necessário a realizar definições específicas no arquivo de pré-configuração usando o pacote *Alien* (MASCARENHAS, 2020).

A segunda etapa, chamada de **Configuração**, se dá no uso do aplicativo de configuração e gerência para a disposição da planta e dos diagramas. Primeiramente, carrega-se o arquivo de pré-configuração com a implementação dos equipamentos e funções.

Na configuração da planta, especificam-se os equipamentos a serem manipulados e os valores seus respectivos atributos. Ou seja, realiza-se a instanciação das classes de equipamentos especificadas na fase de pré-configuração.

A configuração dos diagramas de controle usada é descrita no padrão IEC 61131 que é o padrão global para a programação de Controlador Lógico Programável (CLP), sendo assim, uma notação baseada em fluxograma. Os diagramas definem as manobras de operação a serem realizadas cobre os equipamentos, definidas como operações nas classes de equipamentos ou como funções auxiliares na frase de pré-configuração.

A terceira etapa, denominada de **Execução**, é o uso efetivo da plataforma MPA.

os dados de pré-configuração e configuração são enviados ao servidor de execução por meio de uma interface de rede e interpretadas pelo executor de diagramas do MPA, que opera os equipamentos de acordo com as programação realizada nos diagramas de controle. Operações sobre o executor podem ser feitar pelo aplicativo de gerência, permitindo iniciar, inserir *breakpoints* e acrescentar diagramas. O servidor de execução é um processo no sistema operacional da planta sendo controlada e também contém o executor de diagramas do MPA. Para realizar as manobras de operação o executor interage como sistema supervisório por uma ponte de comunicação específica. O MPA, atualmente, opera com duas pontes de comunicação: uma para o sistema operacional *Microsoft Windows* e supervisórios que sigam o padrão de automação OPC (*Open Platform Communications*) e outra para a plataforma *OpenVMS Alpha* com o supervisório VXL.

A linguagem visual no aplicativo de gerência e configuração para o desenvolvimento de aplicações de controle e operação é a grande vantagem do uso do MPA. Entretanto, na parte da pré-configuração, é necessário a programação em baixo nível das classes de equipamentos e funções auxiliares pela linguagem Lua. É nessa etapa que equipamentos reais dos processos industriais são modelados como classes específicas. Lua é uma linguagem de *script* de multiparadigma e foi projetada para estender programas escritos nas mais diversas linguagens. Apesar de não ser exatamente uma linguagem orientada a objetos, ela fornece metamecanismos para a implementação de classes e herança. Na etapa de Pré-Configuração, são criadas os atributos e métodos de cada classe, ou seja, de cada tipo de equipamento. Na parte de Configuração, é feita a instanciação dos equipamentos que compõem a planta a ser operada e, mais ainda, são modelados os diagramas de execução pela linguagem de fluxos que descrevem as manobras de operação dos equipamentos da planta, isto é, a sequência de execução das funções e métodos dos equipamentos da planta.

2.6 EMSO - PLATAFORMA DE SIMULAÇÃO

A plataforma *Environment for Modeling, Simulation and Optimization* (EMSO) (SOARES, 2003) é um simulador, otimizador e controlador genérico de processos em regime estacionários e dinâmicos baseado em equações. Contém um ambiente de desenvolvimento que permite o usuário descrever equipamentos, modelar processos, otimizar, simular e visualizar os resultados obtidos.

Segundo Soares (2003), as simulações de processos no EMSO são descritas por sistemas de equações algébrico-diferenciais. Para tal, foi desenvolvido uma linguagem

descritiva própria para a modelagem de equipamentos, com a abordagem de orientação a objetos, gerada a partir de agrupamento características de modelagem encontradas em linguagens que existiam anteriormente, obtendo como resultado uma linguagem com a maior aproximação daquela usada como meio de expressão e comunicação e com maior capacidade de reutilização de código. A Figura 2.7 apresenta uma estrutura geral do EMSO.

A linguagem de modelagem do EMSO é composta por três entidades principais (SOARES; SECCHI, 2003): *Model*, *Device* e *Flowsheet*.

O *Model* ou modelo, consiste na descrição matemática de um *device*, ou dispositivo. Ou seja, uma abstração matemática de algum equipamento real, parte de um processo ou mesmo um *software*. No modelo pode haver em sua composição parâmetros (invariáveis no tempo), variáveis, equações, condições inciais, condições de limites e submodelos. Os modelos matemáticos de equipamentos e processos utilizam de estruturas de composição, pela qual o modelo de um processo pode ser composto por submodelos de equipamentos ou subprocessos. Também pode se realizar a herança, em que um modelo tem como base outro pré-existente, somente realizando novas funcionalidades. Mais ainda, permite-se a utilização, em conjunto com os modelos projetados no EMSO de modelos e submodelos programados em C, C++, FORTRAN.

Já o *Device* ou dispositivo, é uma instância de um modelo e retrata um equipamento real do processo. Um único modelo pode ter como base diferentes equipamentos, em que tem a mesma restrutura, mas podem ter diferentes condições, como parâmetros e especificações.

Por fim, o *Flowsheet* ou fluxograma, é composto por um conjunto de *devices* que representa o processo a ser analisado. Através de uma linguagem gráfica, sua definição acontece na interface incluindo no diagrama os dispositivos e realizando suas conexões.

Além destas possibilidades de desenvolver os modelos, construir os fluxogramas, executar as simulações e ver os resultados, também é possível realizar a consistência de um projeto, estimar parâmetros e otimizar o sistema. A plataforma possui várias análises de consistência de modelos e do processo inteiro a ser simulado, apresentadas como: consistência das condições iniciais, unidades de medidas e resolubilidade do sistema de equações. Para a otimização, pode-se definir várias expressões a serem maximizadas ou minimizadas, para calcular a melhor solução conforme o estado do sistema, objetivos e restrições estabelecidas (SOARES; SECCHI, 2003).

O *software* também possui integração para geração de resultados no MATLAB (THE MATHWORKS INCORPORATED, 2021) e como aplicativo EMSO-OPC Link (FINKLER; SOARES, 2007), para troca de informações entre os diagramas de processos e servidores

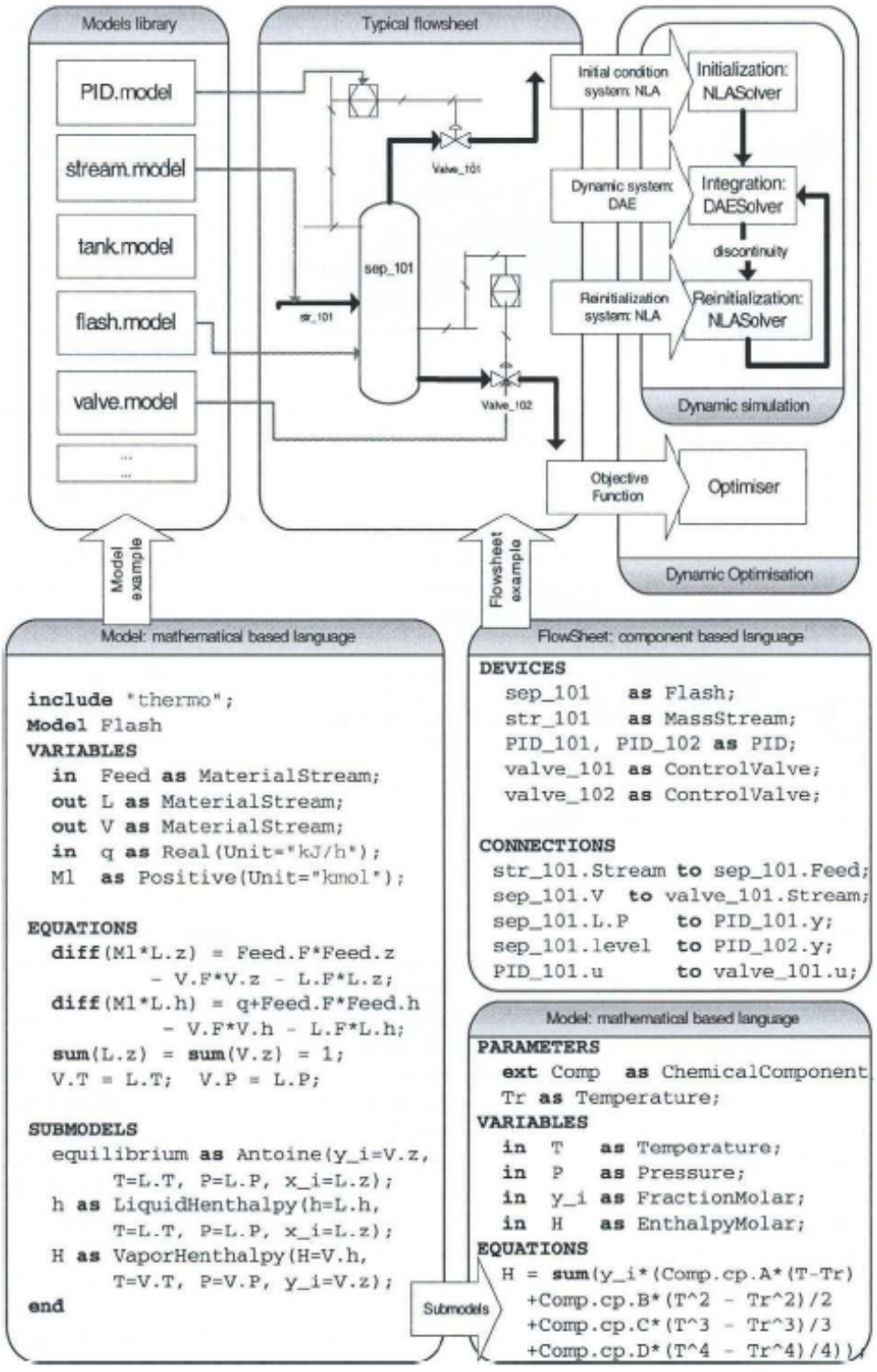

Figura 2.7: Visão geral da estrutura do EMSO e seus componentes (SOARES; SECCHI, 2003).

OPC. São estabelecidos vínculos entre as variáveis dos fluxogramas e as *tags* OPC correspondentes, possibilitando a comunicação de dados durante as simulações. Em razão dessa comunicação, pode-se usar o EMSO como simulador da planta a ser opera para testes e operação de controle gerados no MPA (DAMO, 2019).

2.7 TRABALHOS CIENTÍFICOS RELACIONADOS

Esta seção contempla uma revisão dos trabalhos encontrados na literatura científica relacionados com a temática de *Model-Driven Reverse Engineering* - Engenharia Reversa Dirigida por Modelos (MDRE) e Round-Trip Engeneering (RTE) para o domínio da automação em ambientes industriais gerais, computação e na indústria petroquímica.

O MoDisco (*Model Discovery*), apresentado nos trabalhos Barbier et al. (2010), Bruneliere et al. (2010) e Brunelière et al. (2014), é um projeto de código aberto desenvolvido para a plataforma Eclipse para MDRE. É uma abordagem baseada em metamodelo genérico e extensível para a descoberta, compreensão e transformação de modelos. O principal objetivo é fornecer suporte para atividades que lidam com sistemas legados com suporte para o leque do entendimento e documentação até a evolução, modernização e garantia de qualidade. O MoDisco possui uma arquitetura modular dividida em três camadas: infraestrutura, tecnologias e casos de uso. A camada de infraestrutura fornece os componentes genéricos utilizáveis no processo de MDRE. A camada de tecnologia oferece suporte para tecnologias legadas. Por último, os cenários de casos de uso mostram exemplos de reutilização e integração. O MoDisco provê uma estrutura genérica e várias ferramentas genéricas com capacidade de extrair automaticamente modelos de sistemas legados, modelos que podem ser manipulados através de transformações. Outra característica possível é o suporte para três tecnologias legadas diferentes: Java, JEE (*Java Enterprise Edition*), JSP (*JavaServer Pages*)) e XML (*eXtensible Markup Language*).

Cosentino et al. (2012) proporam uma abordagem que possui o objetivo de extrair regras de negócio de código-fonte Java, isolando os segmentos de códigos relativos aos processos de negócio. Para chegar a tal fim, a abordagem explora conceitos de MDRE, por meio das seguintes etapas: descoberta do modelo por meio do código Java, classificação de varáveis e criação de modelo de variáveis de domínio e extração de modelo de regras de negócios. A etapa de descoberta do modelo é implementada utilizando a ferramenta MoDisco, enquanto as etapas seguintes representam uma cadeia de transformação M2M realizada por meio da *Atlas Transformation Language* (ATL). Para a validação do trabalho ocorreu por meio de um aplicativo que simula o comportamento de animais e humanos em um campo rural, em que os atores sendo, humanos e animais, podem agir e se locomover de acordo com a sua natureza.

O trabalho realizado por Brahim, Taoufiq et al. (2013) apresenta uma abordagem de engenharia reversa para sistema de legados COBOL. Esta solução MDRE tem por objetivo identificar objetos a partir de descrições de registros em três etapas: extração

da descrição dos dados para criar os modelos PSM que estão em conformidade com o metamodelo de descrição do arquivo COBOL, mescla dos modelos PSM extraídos e geração de um modelo comum e transformação do modelo em comum e um PIM representado como um diagrama de classe de domínio e refinamento do diagrama de classe aplicando heurísticas que extraem mais informações do dados legados incorporados em arquivos de armazenamento. A validação do trabalho ocorreu através de um conjunto de códigos escritos em COBOL. Mais ainda, realizaram a comparação da abordagem MDRE com uma abordagem de agrupamento através de três métricas: *recall*, precisão e *F-score* (medida de teste de acurácia em análises estatística de classificação binária). Os resultados apresentaram que a abordagem MDRE tem um melhor desempenho do que a de *clustering*.

Fleurey et al. (2007) em seu trabalho apresenta uma abordagem semiautomática de Round-Trip Engeneering (RTE) com a finalidade da migração de *software* industrial de grande porte. A partir da necessidade de redesenvolver completo do software legado sempre que uma migração de *software* ocorre. É destacado no trabalho que a abordagem pode aumentar o processo de automação da migração de *software* e a reutilização do design existente do código legado. A solução proposta inclui a análise automática do código-fonte, a geração de modelos abstratos em modelos PSM e a geração de código para o sistema de destino. A validação se deu em um estudo de caso real, desenvolvido em COBOL, que originou na migração de um sistema bancário de grande escala de *mainframe* para *Java2 Enterprise Edition* (J2EE). Os autores fornecem para este trabalho um conjunto de ferramentas denominada *Model-In-Action* (MIA). O conjunto de ferramentas é implementado para a RTE, sendo composto por um módulo MIA-*Transformation* para transformações M2M e um módulo MIA-*Generation* para geração de código.

Em (PÉREZ-CASTILLO; GUZMÁN; PIATTINI, 2011) e (PEREZ-CASTILLO, 2012), é proposta uma abordagem semiautomática para extrair os processos de negócios de sistemas de informação legados. A abordagem é baseada no *Modernization Approach for Recovering Business processes from LEgacy Systems* (MARBLE) (PÉREZ-CASTILLO et al., 2011), um *framework* genérico e de proposito geral que descobre o processo de negócios do legado do sistema através das tecnologias *Architecture-Driven Modernization* (AMD) e *Knowledge Discovery Metamodel* (KDM). A solução inclui a extração de conhecimentos de sistemas legados mediante uma análise estática e dinâmica, a geração do modeloKDM, tendo por início o conhecimento extraído através do tecnologia QVT, e a descoberta dos processos de negócios do modelo KDM por meio da aplicação de negócios padrões e transformações de modelo QVT.

A aplicação de engenharia reversa de *software* no contexto orientado a objetos, de acordo com o padrão de MDA é proposto em (FAVRE; MARTINEZ; PEREIRA, 2009; FAVRE; MARTINEZ; PEREIRA, 2018). O foco colocado nestes trabalhos está na extração de diagramas UML do código-fonte Java. Esta abordagem propõe uma prova formal das transformações, além da reconstrução dos modelos. Por meio da prova formal, há a possibilidade de manter a consistência no processo de engenharia reversa. Em essência, esta abordagem explora a análise estática e dinâmica para gerar PSMs e PIMs com base no código e para, mais ainda, para analisar a consistência das transformações realizadas do código para os modelos e entre os modelos.

Outra abordagem de MDRE pertinente de citação decorre dos trabalhos de Ramón, Vanderdonckt and Molina (2013) e Ramón, Cuadrado and Molina (2014) para GUI (*Graphical User Interface* - Interface Gráfica de Usuário) de sistema desenvolvidos com ferramentas *Rapid Application Development* (RAD) ou ambientes *Fourth Generation Language* (4GL). Esta abordagem se destina aos GUIs para modificar, por exemplo, migrar aplicativos para plataformas avançadas, ou adaptar, por exemplo, para dispositivos móveis. Para permitir a evolução dessas GUIs, representações de alto nível delas podem ser geradas e pode sere aplicada uma abordagem MDE. Os modelos são usados em um alto nível de abstração para tornar explícito o *layout* das interfaces. A validação ocorreu por intermédio de dois aplicativos reais em dois domínios de aplicativos diferentes por desenvolvedores distintos. Os objetivos desses dois estudos foram aplicar a abordagem de engenharia reversa às GUIs disponíveis e gerar interfaces de usuários Java *Swing*.

Angyal, Lengyel and Charaf (2008) propõe uma abordagem para RTE baseada nas diferenças e mesclas de AST. O AST é considerado o PSM de acordo modelosMDA para esta proposta. A abordagem de RTE compreende duas etapas de ida e volta distintas: a primeira, entre o modelo PIM e o PSM e, a segunda, entre o modelo PSM e o código-fonte. Na tentativa de evitar as perdas de informações durante a o processo RTE, o trabalho propõe o uso de um modelo de rastreamento, que é usado para sincronizar o PIM e o PSM, ou seja, o AST. Além disso, o AST e o código-fonte são atualizados usando uma mesclagem incremental bidirecional de baixa granularidade com base na diferenciação de três vias.

No trabalho de Nirumand, Zamani and Ladani (2019) é apresentado um *framework* denominado de *VAnDroid*, baseado na abordagem de MDRE, que busca identificar os riscos e vulnerabilidades de segurança relacionados ao modelo de comunicação de aplicativos Android. O *framework* proposto é implementado como uma ferramenta baseada em Eclipse, que identifica automaticamente *Intent Spoofing* e o *Unauthorized Intent Receipt* como dois ataques relacionados ao modelo de comunicação do aplica-

tivo Android. Na estrutura da proposta, algumas informações relacionadas à segurança incluídas em um aplicativo Android são automaticamente extraídas e representadas como um modelo específico de domínio. Em seguida, ele é usado para analisar as configurações de segurança e identificar vulnerabilidades no aplicativo correspondente. A validação da ferramenta foi aplicada a vários aplicativos Android, incluindo 20 da Google Play e 100 aplicativos do repositório *F-Droid*.

Yang et al. (2021) propõem *C2AADL_Reverse*, uma abordagem de MDRE para verificação e desenvolvimento de *software* crítico para segurança. O *C2AADL_Reverse* pega o código-fonte em linguagem C multitarefa como entrada e gera o modelo dos sistema de *software* legados em *Architecture Analysis and Design Language* (AADL). Dois tipos de atividades são propostas para garantir a correção do *C2AADL_Reverse*. Primeiro, a validação da engenharia reversa e, segundo, os modelos AADL de engenharia reversa usando *Uppaal Model Checker* (UPPAAL) para estabelecer prioridades em nível de componentes e o *Assume Guarantee REasoning Environment* (AGREE) para realizar verificação composicional da arquitetura.

Outro trabalho que aplica a metodologia de engenharia reversa em MDE é o (MARAH; CHALLENGER; KARDAS, 2020) em que desenvolvem uma ferramenta chamada *RE4TinyOS* para criação ou atualização de modelos de aplicativos a partir de programas *TinyOS* para o desenvolvimento de redes de sensores sem fio. A criação dos metamodelos ocorreram a partir de um gerados de *parser* por meio da linguagem ANTLR para a criação de uma AST, o que possibilita navegar pela AST para, enfim, popular o metamodelo com os objetos do código-fonte. Para o desenvolvimento dos modelos, foi utilizado o *plug-in* EMF do Eclipse. Um estudo de caso apresenta a capacidade de criar modelos de aplicativos por meio de aplicativos *TinyOS* já existente sem modelos, o que é essencial para a integração das implementações de aplicativos *TinyOS* de terceiros nos processos MDE.

Outro trabalho a ser destacado é o levantamento do estado da arte em MDRE realizado por Raibulet, Fontana and Zanoni (2017). A revisão sistemática proposta neste trabalho é motivada para esclarecer três perguntas principais: a primeira, são quais metamodelos são usados pelas abordagens MDRE e se são propostos a resolver problemas específicos ou são reutilizados para mais de fim. O segundo questionamento pretende investigar quais as ferramentas utilizadas para implementação das abordagens, se fornecem novas ferramentas ou reutilizam ferramentas existentes. E a terceira questão investiga o nível de automação das abordagens MDRE.

Sabir et al. (2019) propõe um *framework* chamado de *Source to Model Framework* (Src2MoF) para gerar diagramas estruturais (classes) e comportamentais (atividades)

para UML a partir do código-fonte Java. O *framework* é baseado na abordagem MDE, que utiliza modelos como cidadãos de primeira classe, aliviando a complexidade dos sistemas de *software*. Um *Intermediate Model Discoverer* (IMD) foi desenvolvido usando um analisador de código-fonte para obter a representação intermediária do sistemas com base de um código Java existente. Em seguida, um mecanismo de transformação de *software* chamado de Gerador de Modelo UML é implementado usando Java, que leva esses modelos intermediários como entrada e produz modelos UML de alto nível do sistema legado.

Izquierdo and Molina (2014) proporam o *Grammar-to-Model Transformation Language* (Gra2MoL), uma DSL (*Domain Specific Language* - Linguagens de Domínio Específico) adaptada para extração de modelos de código *General-Purpose Programming Languages* (GPL). De fato, a DSL é uma linguagem de transformação T2M que pode ser aplicada a qualquer código em conformidade com uma gramática. A validação da abordagem proposta ocorre, praticamente, transformando um código escrito em linguagem Delphi em um *Abstract Syntax Model* (ASM).

Não foi possível encontrar na literatura trabalhos científicos relacionados ao foco principal desta dissertação de mestrado, que se refere ao uso de abordagens dirigidas à modelo com enfoque em engenharia reversa de equipamentos de plantas industriais, como objetivo em compartilhar informações de operação a respeito destes equipamentos em diferentes plataformas de *softwares* de simulação de processos, operação, controle, automação.

2.8 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os conceitos da *Model-Driven Engineering* - Engenharia Dirigida por Modelos (MDE) e os atributos para o desenvolvimento de aplicações com sua abordagem. Também foram apresentados *Model-Driven Reverse Engineering* - Engenharia Reversa Dirigida por Modelos (MDRE) juntamente com o acréscimo de aspectos de Engenharia Reversa em abordagens de MDE. Mais ainda, foram expostas as ferramentas escolhidas para a implementação da solução. Da mesma forma, mostrou-se o ponto de partida da implementação do trabalho com a infraestrutura M4PIA bem como seu escopo de atuação e suas limitações. Do mesmo modo, foram apresentadas as plataformas de *software* de simulação, controle e operação usadas na industria petroquímica que fazem parte da infraestrutura M4PIA presentes nesta dissertação. Também foi exposto um conjunto de trabalhos científicos relacionados com o escopo desta dissertação. No próximo capítulo, será apresentada em mais detalhes a infraestrutura

M4PIA bem como a solução de Engenharia Reversa proposta para o domínio de aplicação de operação, controle e simulação.

3 PROCESSO DE ENGENHARIA REVERSA PROPOSTO

Este capítulo apresenta a solução de Engenharia Reversa (RE) desenvolvida como a principal contribuição dessa dissertação de mestrado. A solução elaborada serve como extensão para a infraestrutura M4PIA, desenvolvida em Damo (2019) e apresentada no capítulo anterior como o principal trabalho relacionado à presente proposta.

O restante do capítulo está organizado como segue. Primeiramente se discute a forma como a estrutura de Engenharia Reversa (RE) se integra ao M4PIA. A seguir são detalhados os metamodelos que servem como base para os mecanismos de transformação desenvolvidos. Por fim são apresentados e discutidos os mecanismos de transformação desenvolvidos.

3.1 INTEGRAÇÃO DO RE À INFRAESTRUTURA M4PIA

Conforme discutido no capítulo anterior, o M4PIA foi criado como suporte ao desenvolvimento de aplicações de definição de classes de equipamentos para plataformas de operação, controle e supervisão da industria petroquímica. A mesma foi desenvolvida seguindo os conceitos de MDE. Em uma abordagem dirigida a modelos, aumenta-se o nível de abstração no desenvolvimento de aplicações através da definição de um metamodelo independente de plataforma (PIM) e os metamodelos específicos de plataforma (PSM), que são MPA e EMSO. Além dos metamodelos, são definidos os processos de transformação, de modelo para modelo (M2M) e de modelo para texto (M2T).

Originalmente, o M4PIA incluía transformações partindo do modelo com maior nível de abstração para os modelos específicos. Também era abordado a transformação dos modelos específicos para código fonte. Este processo ocorria em um único sentido, do maior nível de abstração até o código fonte. O fato de não ser possível iniciar o processo de transformação a partir do código fonte representou uma limitação do M4PIA, uma vez que em um sistema legado de um planta industrial petroquímica, existem muitos materiais já desenvolvidos e até mesmo em atuação no sistema. A aplicabilidade da RE no contexto do M4PIA visa aproveitar aplicações existentes em uma modelagem de simulação e expandi-las para o contexto de aplicação em sistemas de controle e operação.

A figura 3.1 ilustra, através das flechas em negrito, a integração dos novos caminhos possíveis pela RE. Essa nova infraestrutura permite ampliar as possibilidades de transformações. Por exemplo, a partir de códigos fontes das plataformas suportadas pelo M4PIA, o código pode ser transformado em modelo em nível específico de plataforma

e, posteriormente, pode ser transformado em um modelo independente de plataforma. Isso acontece por meio das novas transformações desenvolvidas, de texto para modelo T2M e de modelo para modelo no sentido reverso M2M.

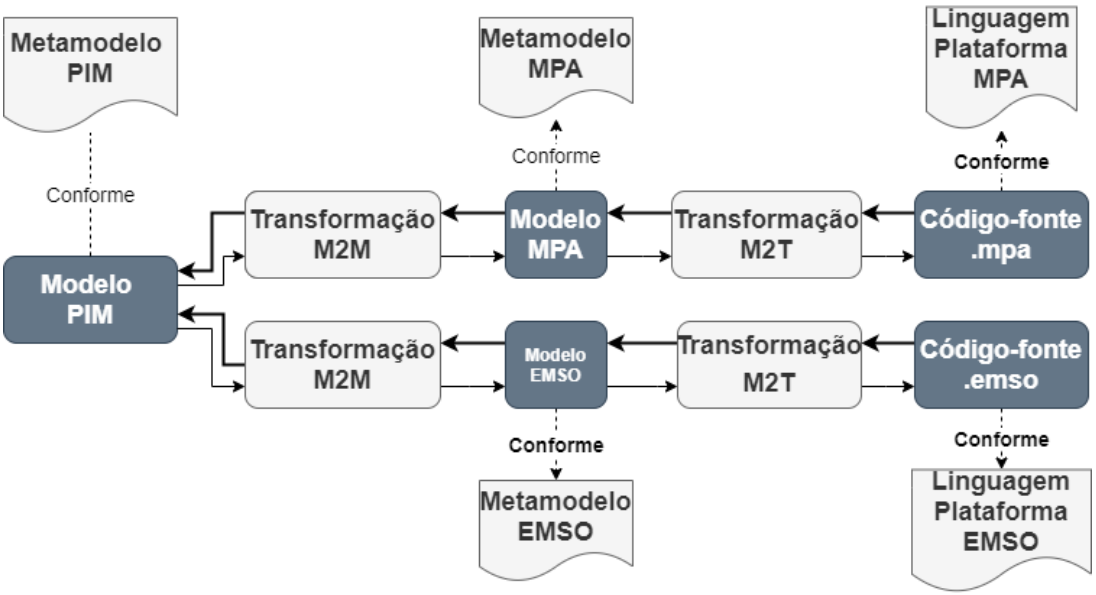

Figura 3.1: RE junto ao M4PIA

A figura 3.2 ilustra de maneira simplifica as cadeias de transformações possíveis dentro da infraestrutura M4PIA possíveis de realizar.

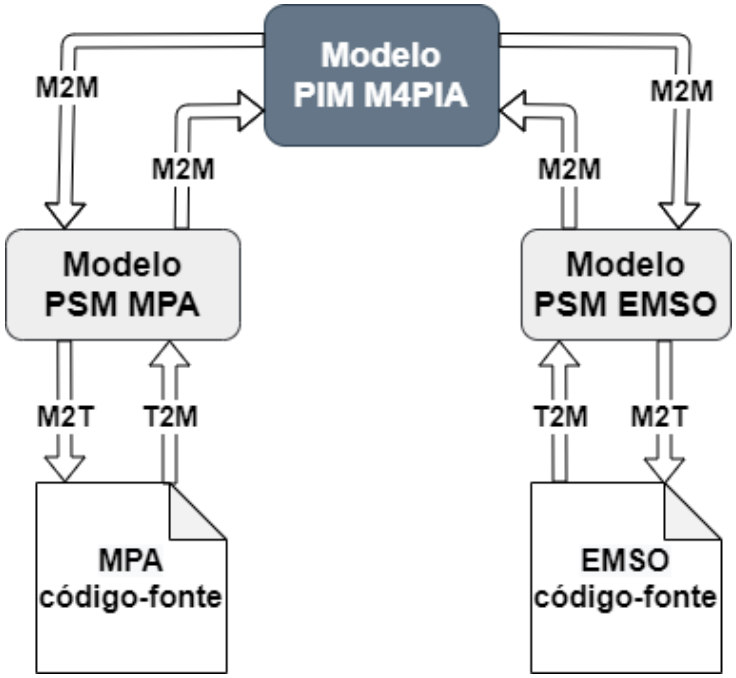

Figura 3.2: Cadeia de transformações da infraestrutura M4PIA.

Para a desenvolvimento da presente proposta, utilizou-se o ambiente *Eclipse Oxygen*, mais especificamente a aplicação *Eclipse Modeling Tools*, que tem como base o

framework EMF e o gerador de *parsers* ANTLR. Esse último auxilia no desenvolvimento da AST (*Abstract Syntax Tree*).

3.2 METAMODELOS

Os metamodelos aqui apresentados são elementos constituintes do M4PIA e foram desenvolvidos na dissertação de mestrado de Damo (DAMO, 2019).

Os equipamentos de uma planta industrial, especificamente as do sistemas de domínio definidos para a infraestrutura M4PIA, podem ser representados por modelos, como visa a abordagem MDE. Os conceitos relevantes ao domínio e seus relacionamentos foram representados em metamodelos, com os quais os modelos das diferentes aplicações de equipamentos e plantas estarão em conformidade. Fazem parte da infraestrutura M4PIA um metamodelo genérico de alto nível de abstração ou, em outras palavras, independente de plataforma e dois metamodelos específicos para as plataformas suportadas, neste caso, as plataformas EMSO e MPA. O metamodelo *Ecore* é utilizado como base para os demais metamodelos, os quais foram desenvolvidos através do EMF. A figura 3.3 ilustra a estrutura de metamodelos em questão.

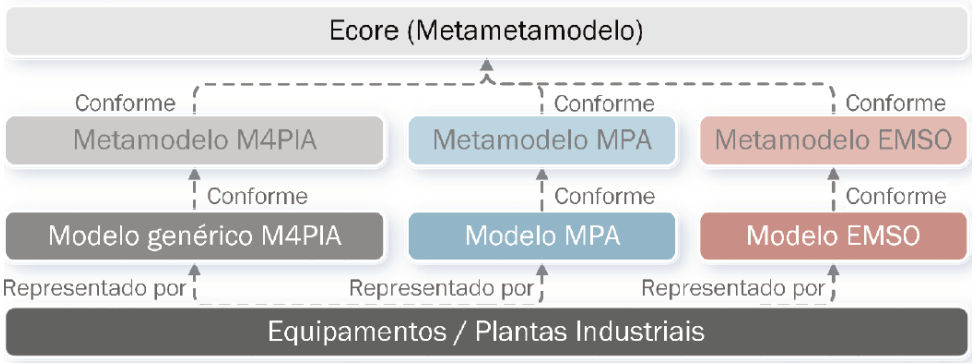

Figura 3.3: Estrutura de metamodelos do M4PIA (DAMO, 2019).

3.2.1 Metamodelo PIM

Por meio da análise de diversas aplicações voltadas para a indústria petroquímica foi especificado um conjunto de elementos genéricos, identificados como susceptível de serem compartilhados por uma ampla gama de sistemas de automação. A estruturação resultante é apresentada através de um Diagrama de Classes, conforme ilustrado na figura 3.4. O conjunto resultante de classes, interfaces e colaborações, com seus respectivos relacionamentos, constitui o metamodelo PIM do M4PIA.

Uma entidade se define como a unidade mais elementar do metamodelo. A classe `Entity` é abstrata e serve unicamente para fornecer uma estrutura de base para as

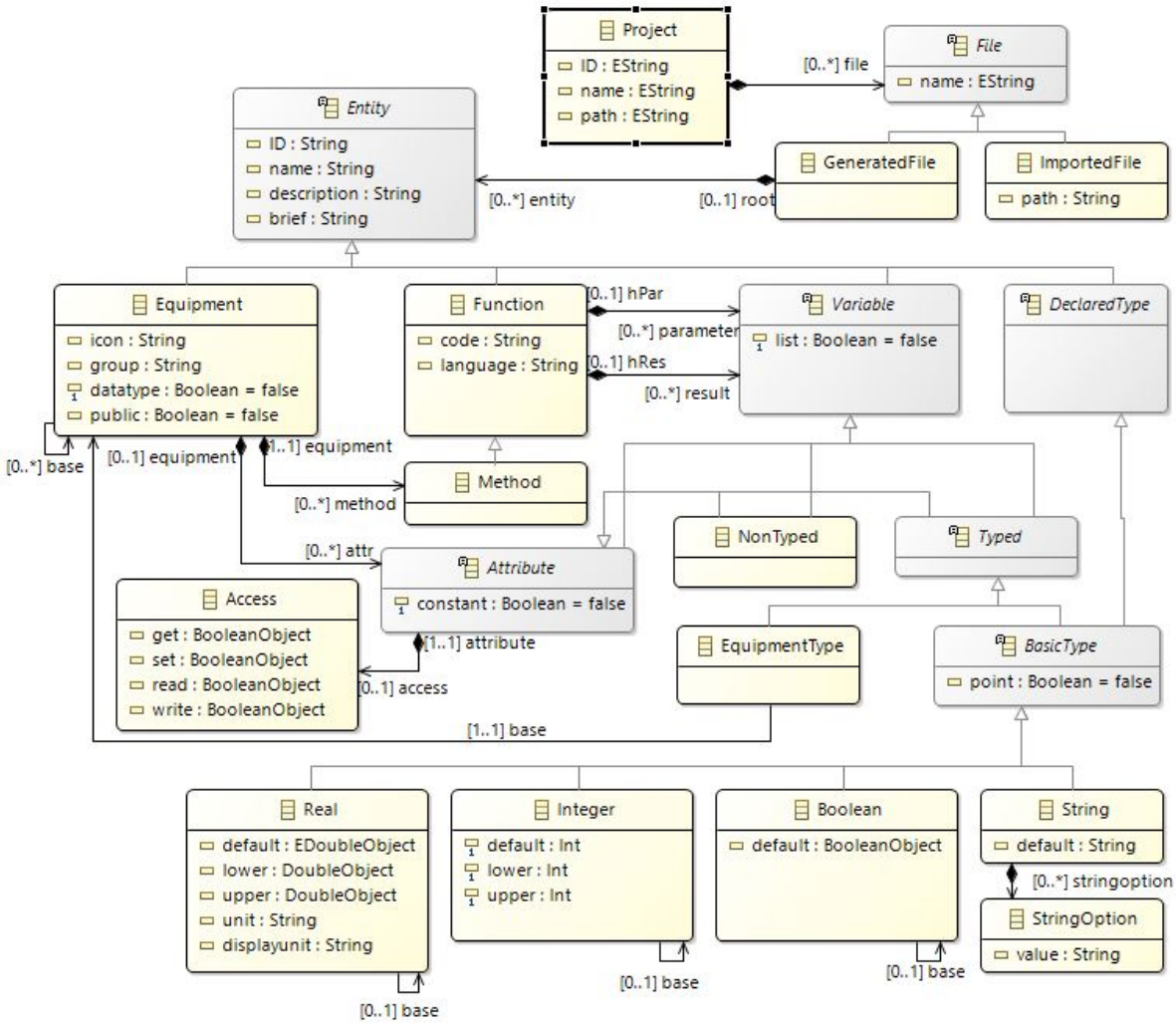

Figura 3.4: Metamodelo PIM do M4PIA (DAMO, 2019).

classes mais específicas da hierarquia. Diferente das classes específicas, uma `Entity` jamais será diretamente instanciada. Os seguintes elementos são entendidos como possíveis entidades:

- os equipamentos que compõem a planta a ser modelada;
- à operação e manutenção das aplicações no sistema como, por exemplo, esperar um determinado tempo ou notificar o operador com uma mensagem;
- variáveis necessárias para o sistema de controle ou simulação;
- novos tipos de dados.

Cada entidade possui como atributo identificador, nome, descrição e descrição resumida. Já um `Project` pode ser instanciado e também é composto por diferentes arquivos `File`. Estas classes `File` podem ser importadas como bibliotecas, `ImportedFile`,

ou funcionam como um grupo de entidades a terem seus códigos gerados pela infraestrutura, `GeneratedFile`. O `Project` é composto por um identificador, nome e diretório com o qual deverão ser armazenados as suas informações. Um `File` contém nome e, para o caso de importar bibliotecas, o diretório em que se encontra.

Um `Equiment` é um tipo de entidade que pode conter métodos e atributos, definidos como classes específicas para espelhar as propriedades físicas e as funções exercidas pelo equipamento modelado na planta. Um `Equipment` pode ter como base um ou mais equipamentos, com implementação por meio de hierarquia. Um exemplo disso seria uma *Válvula de Alívio*, que pode ter como base um equipamento *Válvula Genérica*, que também pode ser utilizado como base para um outro equipamento, *Válvula de Reciclo*. Ainda há a possibilidade de se definir um ícone (imagem) para um equipamento, determinar se ele estará sempre disponível para ser acessado e instanciado, se pode ser considerado um tipo de dado e se possui um grupo no qual pode ser classificado.

A classe abstrata `Variable` caracteriza uma variável lógica e que pode ser `NonTyped` ou de um tipo específico, `Typed`, mais precisamente dos tipos `Real`, `Integer`, `Boolean` ou `String`. Uma variável pode ser única ou um vetor (lista) de dados. Uma `Variable` do tipo equipamento necessita estar associada a um `Equipment`. Há a possibilidade de associar um valor padronizado para uma variável básica, opções de valores para uma variável do tipo texto e limites superior e inferior para variáveis numéricas, neste caso, dos tipos `Real` ou `Integer`. É possível determinar uma unidade de media padronizada no `Real` e uma unidade de medida de exibição. As variáveis básicas ainda podem ser representadas por pontos de controle se armazenarem o caminho para achar o valor em um servidor de dados, ou não, se já tem o valor diretamente disposto.

O `Method` é uma classe especializada da classe `Function` e tem ligação exclusiva com um `Equipment`. A `Function` caracteriza uma entidade lógica que pode ser tanto representar uma operação em alto nível (domínio físico da planta) quanto em baixo nível (domínio de aplicação). Dessa forma, uma `Function` tem a possibilidade de ter associadas inúmeras instâncias de `Variable`, operando como parâmetros ou resultados, assim como podem possuir um código, o qual descreve textualmente a instrução desejada para o interpretador utilizando determinada linguagem de programação.

A classe `Attribute` de um equipamento é um caso específico de uma `Variable`, o que possibilita que, através da hierarquia de relacionamentos, ele possa assumir um tipo que pode ser outro `Equipment`. Esta característica assegura ao metamodelo a aptidão de gerar cenários onde existem recursividades naturais nas relações entre equipamentos. Em outras palavras, um equipamento terá por atributo outro equipamento. Um exemplo pode ser um equipamento *Motor* poder ser um atributo de um equipamento *Compressor*,

sendo ambos equipamentos instanciados pela classe `Equipment`.

Um `Attribute` também possui um gerenciamento de permissões de acesso das aplicações ao atributo, para alterações e operações de leitura ou escrita representado pela classe `Access`. Quando o atributo é um valor básico e não um ponto de controle, o acesso à leitura e escrita é definido pelos valores lógicos de *get* e *set* respectivamente. Se o atributo for um ponto de controle, *get* e *set* irão definir o acesso ao caminho para o valor no servidor e os valores lógicos *read* e *write* definem a permissão para leitura e escrita, respectivamente, do valor do ponto no sistema supervisório em que estiver conectado.

A classe `DeclaredType` possibilita a criação de novos tipos de dados, tendo por base os tipos básicos, podem ser usados como base para variáveis e atributos. Um exemplo é a possibilidade de desenvolver um atributo com apenas valores reais positivos, chamando-o de *Real Positivo*, e tê-lo como base de um atributo real chamado *Abertura* de um equipamento *Válvula*.

O metamodelo também possui restrições de modelagem para assegurar a construção de um modelo de equipamento ou processos industriais da maneira correta. As restrições para o metamodelo M4PIA foram desenvolvidas por meio da linguagem OCL. Algumas restrições do metamodelo PIM merecem destaque como por exemplo:

- todos os identificadores podem conter apenas letras, números e símbolos "_"(*underscore*) devendo iniciar por uma letra;
- todo `File` de um `Project` precisa ter um nome único;
- O identificador dever ser único entre todas as `Entity` do tipo `Equipment` e `Function` de todos os `File` de um `Project`;
- o identificar necessita ser único entre todas as `Variable`, parâmetros ou resultados de uma `Function`;
- os `Access` *read* e *write* precisam estar desabilitados para `Attribute` do tipo `BasicType` que não seja ponto de controle ($point = false$);
- o limite inferior deve ser menor que o limite superior para dados `Real` ou `Integer`;
- o valor de um dado do tipo *default* tem que estar entre os limites inferior e superior.

3.2.2 Metamodelo PSM para MPA

Ao se desenvolver uma aplicação na plataforma MPA, é na etapa de pré-configuração que se realiza a definição de equipamentos, tipos de dados e funções. O diagrama de

classes com o metamodelo PSM para o MPA é ilustrado na figura 3.5. O mesmo foi desenvolvido a partir da combinação dos conceitos modelados e armazenados nos arquivos de pré-configuração. Modelos gerados a partir desse metamodelo é que serão transformados em código fonte, ou seja, arquivos de pré-configuração.

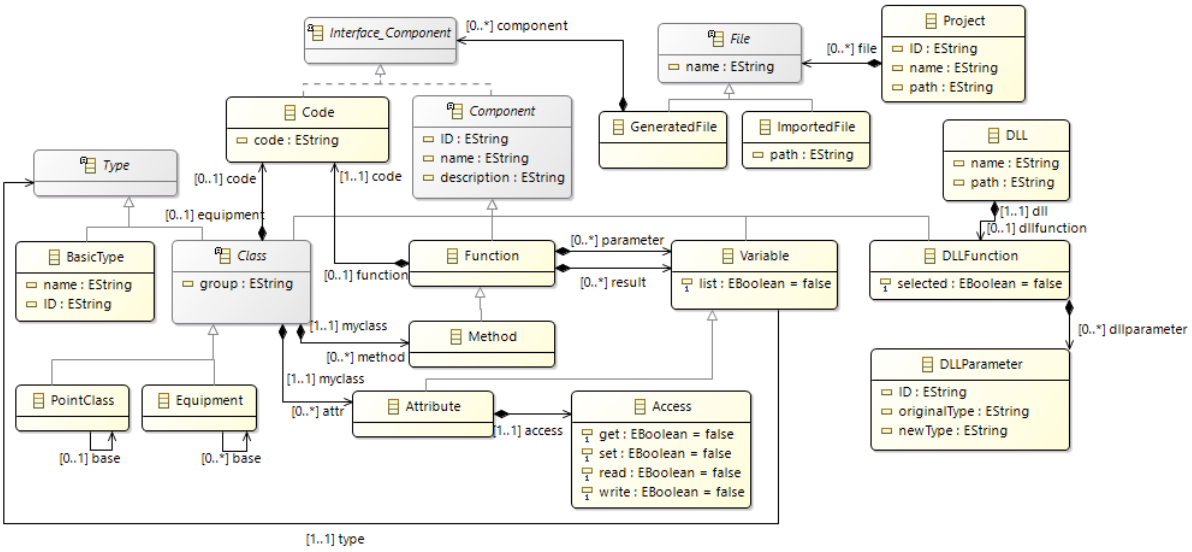

Figura 3.5: Metamodelo PSM para MPA (DAMO, 2019).

Sobre os projetos e os arquivos, o relacionamento é o mesmo explicado na seção 3.2.1. Um `GeneratedFile` pode ter, além de componentes, trechos de códigos em linguagem de programação Lua. `Code` não são exclusividade de funções, visto que em `Class` e em arquivos a serem gerados também podem ter `Code` em sua composição.

O `Component` é o equivalente a `Entity` no metamodelo M4PIA mostrado na figura 3.4. Mais ainda, um `Component` também possui atributos para identificador, nome e descrição. São componentes da pré-configuração: classes de equipamento e pontos de controle, funções auxiliares, variáveis e funções importadas de DLLs.

O componente abstrato `Class` pode tanto ser especificado como `Equipment` como, também `PointClass`. Este último é considerado na plataforma como um tipo de dado para especificar pontos de controle. As classes podem conceber parte de um grupo de classificação e a hierarquia entre elas são implementadas pelos relacionamentos bases somem com especificações do mesmo tipo. Classes são constituídas por métodos e atributos, que dispõem de relacionamento exclusivo com uma classe e representam e alteram os estados, características e comportamentos do equipamento ou ponto de controle modelado.

A classe `Method` implementa uma operação sobre equipamento e é uma especificação de `Function`, que pode ter associadas à ela inúmeras variáveis como parâmetro ou retorno. Para a plataforma MPA há bibliotecas com funções auxiliares relevantes e

amplamente empregadas nas suas aplicações como, por exemplo: aguardar o tempo passado como parâmetro; guardar dados de operação (*log*); notificar o operador; calcular interpolação, média e desvio padrão de conjunto de valores; e diversas outras.

A classe `Variable` e sua especificação `Attribute` não são abstratas, há a possibilidade de ser uma lista de dados e, necessariamente, ter um `Type` associado. Tipos são os `BasicType`, classes de pontos de controle e equipamentos. Por meio das ferramentas de modelagem EMF, as instâncias dos tipos básicos do MPA são importadas automaticamente no início de cada projeto, elas são: *Real*, *Inteiro*, *Lógico* e *Textual*. Quatro instâncias de classes de pontos de controle nativos do MPA também são importadas: *Ponto Real*, *Ponto Inteiro*, *Ponto Lógico* e *Ponto Textual*, cada uma com seus atributos.

Todo atributo de uma classe tem um `Access` associado. O qual serve para gerenciar as permissões de acesso das manobras de operação definidas nos diagramas de controle sobre o atributo. O acesso ao caminho para o ponto de controle é definido por *get/set* e ao valor diretamente no supervisório por *read/write*. Para os valores básicos são usados os campos *get/set* para definição do acesso.

Outro componente relevante para as aplicações em MPA é a função DLL. Por meio do pacote *Alien*, o MPA permite a importação de bibliotecas dinâmicas (DLLs), declarando as `DLLFunctions` disponibilizadas e também selecionadas, seus `DLLParameters`, seu nome e o caminho da biblioteca correspondente. O pacote possui tipos de dados específicos e, para a correta declaração das funções, o tipo de cada parâmetro deve ser corretamente mapeado do original (Linguagem da DLL) para o novo tipo *Alien*.

Assim como o metamodelo M4PIA, também existem restrições para a modelagem de aplicações para a plataforma MPA. A seguir são listadas as principais restrições em questão:

- todos os identificadores podem contar apenas com letras, números e símbolos "_"(*underscore*) devendo iniciar por uma letra;
- todo `File` de um `Project` precisa ter um nome único;
- O identificador dever ser único entre todos os `Component` do tipo `Class` e `Function` entre todos os `File` de um `Project`;
- o identificar necessita ser único entre todas os `Method` e `Attribute` de uma `Class`;
- o identificador deve ser único entre todas as `Variable`, parâmetros ou resultados de uma `Function`;
- Os `Access` *read* e *write* devem estar desabilitados para `Attribute` do tipo `BasicType`.

3.2.3 Metamodelo PSM para EMSO

O outro metamodelo PSM que compõe a infraestrutura M4PIA é o EMSO. Ele serve para a modelagem de aplicações para simular processos industriais. O metamodelo em questão é apresentado na figura 3.6.

Um `Project` e seus modos de relacionamentos com os arquivos foram mantidos similar ao apresentado anteriormente no metamodelo M4PIA em 3.2.1. Para além da modelagem dos equipamentos para fins de simulação, a plataforma contém outras funcionalidades tal como otimização de processos, estimação de parâmetros e criação de diagramas de equipamentos.

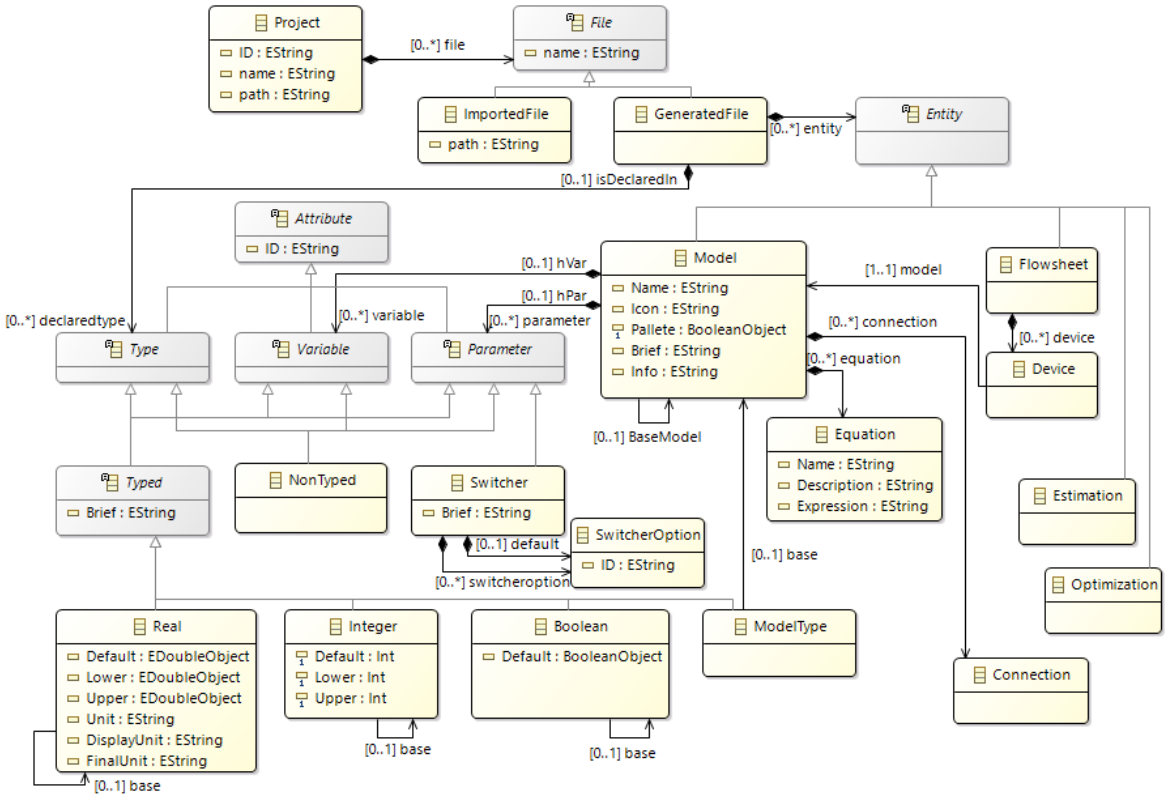

Figura 3.6: Metamodelo PSM para EMSO (DAMO, 2019).

Um arquivo da linguagem de modelagem própria do EMSO pode ser composto por uma entidade e declarações de novos tipos de dados. São considerados uma `Entity`: o modelo de um equipamento, o diagrama de equipamentos e a otimização. O domínio da infraestrutura é a definição de equipamentos para diferentes plataformas, feito no EMSO através da definição de modelos matemáticos. Portanto, o metamodelo tem composição principalmente nos conceitos relacionados a entidade `Model`. Os demais conceitos estão apenas representados como `Flowsheet` e `Devices`.

A classe `Model` terá sempre um *Name* e pode conter um ícone, uma descrição detalhada com outra resumida e se estará disponível para ser instanciado na paleta

de construção dos diagramas. As funções matemáticas que irão representar o comportamento do modelo são relacionadas sempre como uma *Equation*. Cada equação do modelo possui uma equação matemática e pode ter definido um nome e uma descrição. A expressão matemática também é descrita na linguagem própria da plataforma. Podem ser necessárias mais de uma equação EMSO para implementar um método definido em outra linguagem de programação no modelo PIM. O `Model` contém ainda parâmetros, constantes durante toda a simulação e variáveis que serão utilizadas em suas equações.

A classe abstrata `Attribute` agora pode ser especificada em `Type` de dados e atributos variáveis, `Variable`, ou constantes, `Parameter`, de um modelo. Variáveis não podem assumir um tipo textual, mas os parâmetros textuais podem ser instanciados por meio de um `Switcher`.

A hierarquia é disponibilizada para `Models` e atributos numéricos ou lógicos ao permitir a associação de um elemento de mesmo tipo como base. Um modelo ainda pode ser composto por atributos `ModelType`, representando que um processo pode ser modelado por subprocessos.

Existem restrições que devem ser respeitadas pelos modelos EMSO implementados em conformidade com o seu metamodelo. Algumas das mais relevantes são listadas a seguir:

- Todos os identificadores de `Project` e `Attribute` e o nome de `Model` podem contar apenas letras, números e símbolos "_"(*underscore*) devendo iniciar por uma letra;
- Todo `File` de um `Project` precisa ter um nome único;
- O identificador deve ser único entre todos os `Component` do tipo `Class` e `Function` entre todos os `File` de um `Project`;
- O identificar necessita ser único entre todos `Attributes`, parâmetros ou variáveis de um `Model`;
- O objeto tido como base deve ser do tipo dado declarado para dados `Real`, `Integer` ou `Boolean`;
- O valor *default* deve estar entre os limites inferior e superior para dados `Real` ou `Integer`.

3.3 TRANSFORMAÇÕES ENTRE MODELOS

O processo de desenvolvimento de transformações para uma Engenharia Reversa (RE) é uma atividade bastante complexa, e que ocorre uma única vez no processo de

criação da solução. Subsequentemente, as transformações são reutilizadas nas mais diversas aplicações do domínio que possuem a mesma plataforma alvo. Para cada plataforma que integra a infraestrutura M4PIA faz-se necessário definir as respectivas regras de transformação.

3.3.1 Transformações de Modelo para Modelo - M2M

A infraestrutura M4PIA foi desenvolvida para que as transformações M2M acontecessem entre os diferentes níveis de abstração: do modelo independente de plataforma (PIM), de mais alto grau de abstração, para o modelo específico da plataforma alvo (PSM), MPA ou EMSO. Os modelos são gerados automaticamente pela execução das regras de transformação M2M.

O desenvolvimento proposto neste trabalho visa realizar a transformação M2M no sentido reverso. Mais especificamente, a partir do modelo PSM MPA ou EMSO a transformação M2M cria um modelo mais abstrato (tipo PIM). A estrutura das transformações M2M reversa propostas no trabalho são apresentadas na figura 3.7.

Figura 3.7: Transformação reversa de modelos (M2M) para a infraestrutura M4PIA.

A implementação das novas regras de transformações M2M utiliza a ferramenta *Eclipse QVT Operational*, com a linguagem QVTo (QVT *Operational Mappings*), a qual é o padrão da OMG para transformações de modelos.

As transformações em questão acontecem de modo unidirecional e as regras de transformação definem as relações entre os conceitos do metamodelo fonte e do metamodelo alvo. Ao ter como ponto de partida o modelo fonte, o PSM específico, a execução da transformação faz o mapeamento de um mais elementos do modelo fonte em um ou mais elementos do modelo alvo, nesse caso o PIM.

3.3.1.1 Transformação M2M Reversa 1: PSM/MPA para PIM

As regras definidas para a transformação M2M para modelos MPA executam um mapeamento dos elementos do modelo específico da plataforma MPA para o modelo PIM. Os argumentos de entrada recebidos pela transformação são o metamodelo PSM MPA, o modelo PSM da aplicação conforme o metamodelo MPA e o metamodelo PIM do M4PIA. Como apresentado no trecho de código 3.1, os modelos e os respectivos metamodelos são declarados no início da transformação, sendo o modelo PIM como entrada/fonte e o modelo PSM como saída/alvo.

```
2 modeltype PSMmpa uses MPAmetamodel('http://www.das.ufsc.br/MPAmetamodel');
3 modeltype PIM uses M4PIAmetamodel('http://www.das.ufsc.br/M4PIAmetamodel');
4
5 transformation mpa2m4pia(in sourcePSM: PSMmpa, out targetPIM: PIM);
```

Trecho de Código 3.1: Declaração dos metamodelos fonte e alvo da transformação M2M MPA para o PIM M4PIA.

O método *main* da transformação inicia o mapeamento pelo objeto `Project`, instanciado no modelo fonte (PSM) de aplicação, conforme apresentado no trecho de código 3.2. Dado que as classes de projeto e arquivos correspondem entre os metamodelos, o identificador, o nome e os relacionamentos do `Project` são mapeados diretamente. O mesmo ocorre para o mapeamento *ImportedFile2ImportedFile*, como exemplificado no código 3.3. No mapeamento *GeneratedFile2GeneratedFile*, os arquivos a serem gerados, nos quais estarão os equipamentos modelados, os componentes do modelo fonte são mapeados em entidades do modelo alvo: funções e `Class` do MPA para funções e equipamentos PIM.

```
7 main() {
8   sourcePSM.rootObjects()[Project] -> map Project2Project();
9 }
```

Trecho de Código 3.2: Método *main* da transformação M2M MPA para o M4PIA.

No código mostrado em 3.4, o mapeamento de uma classe para um equipamento é dividido em um mapeamento do `PointClass` para o `Equipment` e do `Equipment` (MPA) para o `Equipment` (M4PIA). Isto se justifica porque o `Class` é abstrato. O `PointClass` e o `Equipment` são similares, contendo identificadores, nome, descrição e também atributos e métodos de cada classe por meio dos mapeamentos individuais de cada elemento (*Attribute2Attribute* e *Method2Method*). A diferença entre `PointClass` e `Equipment` no PSM é um atributo *dataype*, que quando tem um valor igual a *true* determina que é um `PointClass`, e quando contém o valor *false* indica ser um `Equipment`. Exemplifi-

```
11 mapping MPAmetamodel::Project:: Project2Project() : M4PIAmetamodel::Project {
12   result.ID := self.ID;
13   result.name := self.name;
14   result.path := self.path;
15   file += self.file->select(a | a.oclIsKindOf(ImportedFile))[ImportedFile]->map
         ImportedFile2ImportedFile();
16   file += self.file->select(a | a.oclIsKindOf(GeneratedFile))[GeneratedFile]->map
          GeneratedFile2GeneratedFile();
17 }
18
19 mapping MPAmetamodel::ImportedFile :: ImportedFile2ImportedFile() :
       M4PIAmetamodel::ImportedFile {
20   result.name := self.name;
21   result.path := self.path;
22 }
23
24 mapping MPAmetamodel::GeneratedFile :: GeneratedFile2GeneratedFile() :
       M4PIAmetamodel::GeneratedFile {
25   result.name := self.name;
26   entity += self.component -> select(a | a.oclIsKindOf(Function))[Function]->map
         Function2Function();
27   entity += self.component-> select(a | a.oclIsKindOf(Class))[Class]->map
         Class2Equipment();
28 }
```

Trecho de Código 3.3: Mapeamentos de projetos e arquivos a serem gerados e importados.

cando de outra maneira, na transformação `PointClass` e `Equipment` do modelo (PSM), transformam-se em `Equipment` no modelo PIM tendo sua distinção realizada pelo *datatype*. Também há a definição da herança no mapeamento, através do relacionamento `base`.

```
134 mapping MPAmetamodel::Class :: Class2Equipment() : M4PIAmetamodel::Equipment
135 disjuncts MPAmetamodel::PointClass :: Point2Equipment,
136       MPAmetamodel::Equipment :: Equipment2Equipment {}
137
138 mapping MPAmetamodel::PointClass :: Point2Equipment() : M4PIAmetamodel::Equipment
139 when {self.oclIsKindOf(MPAmetamodel::PointClass)}{
140   result._datatype := true;
141   result.ID := self.ID;
142   result.name := self.name;
143   result.description := self.description;
144   result.base := self.base.resolveone(M4PIAmetamodel::Equipment);
145   result.attr += self.attr->Attribute2Attribute();
146   result.method += self.method->Method2Method();
147 }
148
149 mapping MPAmetamodel::Equipment :: Equipment2Equipment() : M4PIAmetamodel::
       Equipment
150 when {self.oclIsKindOf(MPAmetamodel::Equipment)}{
151   result._datatype := false;
152   result.ID := self.ID;
153   result.name := self.name;
154   result.description := self.description;
155   result.base := self.base.resolveone(M4PIAmetamodel::Equipment);
156   result.attr += self.attr->Attribute2Attribute();
157   result.method += self.method->Method2Method();
158 }
```

Trecho de Código 3.4: Mapeamento do `Class` do PSM para o `Equipment` do modelo PIM.

O mapeamento *Function2Function* e *Method2Method* são iguais na transformação MPA para o modelo M4PIA, conforme apresentado no código 3.5. Os parâmetros e resultados são incluídos pelo mapeamento *Variable2Variable* e, caso a linguagem usada na definição do código seja a linguagem LUA, o *code* do modelo PIM recebe o *code* do objeto `Code` correspondente do modelo fonte PSM. Caso a declaração seja em outra linguagem, seu código e linguagem são incluídos na sua descrição.

```
265 mapping MPAmetamodel::Method :: Method2Method() : M4PIAmetamodel::Method {
266   result.ID := self.ID;
267   result.name := self.name;
268   result.parameter += self.parameter -> Variable2Variable();
269   result.result += self.result -> Variable2Variable();
270   if self.ID.equalsIgnoreCase("LUA") then {
271     result.description := self.description;
272   result.code := self.code.code;
273   }
274   else {
275     result.description := self.description + "\n ( In language: "+ self.ID + "\n
          Code:\n" + self.code.code + "\n)";
276   }
277   endif
278 }
```

Trecho de Código 3.5: Mapeamento de métodos de equipamentos (análogo para funções).

O mapeamento dos elementos que possuem um relacionamento de atributo (*attr*) com um equipamento ou ponto de controle do modelo PSM é similar ao das variáveis (parâmetros e resultados) de uma função, distinguindo apenas pela chamada ao mapeamento do acesso, como anteriormente ilustrado no trecho de código 3.4. Já o trecho de código 3.6 apresenta uma parte em comum dos mapeamentos *Variable2Variable* e *Attribute2Attribute*, especificamente o desmembramento dos tipos de variáveis possíveis no M4PIA. A partir do tipo da variável no PSM MPA, numéricas, lógicas ou textuais, a separação acontece para a definição dos estados e adequada associação do relacionamento de herança e parte dos mapeamentos. No trecho 3.6 é mostrada a separação *Attribute2Attribute* e a realização do mapeamento para um `BasicType` do M4PIA. Além disso, a verificação da base pelo *type* onde por meio do identificador (*Real* ou *Real_POINT*), o atributo *point* é definido como *true* ou *false*, respectivamente.

3.3.1.2 Transformação M2M Reversa 2: PSM/EMSO para PIM

A outra transformação M2M integrada à infraestrutura M4PIA suporta a generalização do modelo PSM da plataforma de simulação EMSO para o modelo PIM. No trecho de código 3.7 são declarados como argumentos de entrada o metamodelo PSM EMSO e o modelo da aplicação em nível PSM como fonte de dados, o metamodelo PIM da

```
161 mapping MPAmetamodel::Attribute :: Attribute2Attribute() : M4PIAmetamodel::
        Attribute
162 disjuncts MPAmetamodel::Attribute :: Attribute2Real,
163           MPAmetamodel::Attribute :: Attribute2Integer,
164           MPAmetamodel::Attribute :: Attribute2Boolean,
165           MPAmetamodel::Attribute :: Attribute2String,
166           MPAmetamodel::Attribute :: Attribute2EquipmentType {}
167
168 mapping MPAmetamodel::Attribute:: Attribute2Real() : M4PIAmetamodel::_Real
169 when {(self.type.oclAsType(MPAmetamodel::BasicType).ID = "REAL") or (self.type.
        oclAsType(MPAmetamodel::PointClass).ID = "REAL_POINT")} {
170   result.ID := self.ID;
171   result.name := self.name;
172   result.description := self.description;
173   result.list := self.list;
174   if self._access->notEmpty() then
175     result._access := self._access.map Access2Access()
176   endif;
177   result.base := self.type.resolveone(M4PIAmetamodel::_Real);
178   if (self.type.oclAsType(MPAmetamodel::BasicType).ID = "REAL") {
179     result.oclAsType(M4PIAmetamodel::BasicType).point := false
180   };
181   if (self.type.oclAsType(MPAmetamodel::BasicType).ID = "REAL_POINT") {
182     result.oclAsType(M4PIAmetamodel::BasicType).point := true
183   };
184 }
```

Trecho de Código 3.6: Mapeamento de atributos de equipamentos (análogo para variáveis).

plataforma M4PIA. O modelo independente de plataforma M4PIA, em nível, PIM é a saída/alvo da transformação.

```
2 modeltype PSMemso uses EMSOmetamodel('http://www.das.ufsc.br/EMSOmetamodel');
3 modeltype PIM uses M4PIAmetamodel('http://www.das.ufsc.br/M4PIAmetamodel');
4
5 transformation emso2m4pia(in sourcePSM:PSMemso, out targetPIM:PIM);
```

Trecho de Código 3.7: Declaração dos metamodelos fonte e alvo da transformação M2M EMSO para o M4PIA.

O método *main*, ilustrado no código 3.8, inicia a transformação pelo objeto `Project`, instanciado no modelo fonte da aplicação. A estrutura de projetos e arquivos é a mesma para o metamodelo PIM e os dois metamodelos PSM. Nesse sentido o mapeamento *Project2Project* é similar em ambas as transformações M2M, conforme exibido no código 3.3.

```
7 main() {
8   sourcePSM.rootObjects()[Project] -> map Project2Project();
9 }
```

Trecho de Código 3.8: Método *main* da transformação M2M EMSO para o M4PIA.

No mapeamento *GeneratedFile2GeneratedFile*, exibido no trecho de código 3.9, os estados e os relacionamentos dos arquivos a serem gerados a partir do modelo PIM são

mapeados. Cada entidade do `Model` no PSM EMSO *Model2Equipment* é mapeada para um `Equipment` no PIM. Mais ainda, cada entidade do tipo `Type` é mapeada para um *DeclaredType*. O `Type` não é considerado uma entidade, mas por meio dele, é possível dividir a transformação para os tipos numéricos e lógicos. De acordo com o metamodelo M4PIA, podem ser definidas funções auxiliares fora do contexto de um equipamento. No domínio desta plataforma específica, o conceito de função está sempre associada a um equipamento através de expressões matemáticas.

```
24 mapping EMSOmetamodel::GeneratedFile :: GeneratedFile2GeneratedFile() :
       M4PIAmetamodel::GeneratedFile {
25   result.name := self.name;
26   result.entity += self.entity->select(a | a.oclIsKindOf(Model))[Model]->map
         Model2Equipment();
27   result.entity += self.declaredtype->select(a | a.oclIsKindOf(Type))[Type]->map
         Type2DeclaredType();
28 }
```

Trecho de Código 3.9: Mapeamento das entidades dos arquivos a serem geradas a partir do modelo PIM.

O mapeamento entre declarações dos tipos é dividido em três mapeamentos, sendo dois deles numéricos e um lógico. Isso é feito com vista à definição dos estados e à correta associação do relacionamento de herança com o mesmo tipo de dado. A separação é exibida no trecho 3.10 e é ilustrada também o mapeamento do tipo `Real` no EMSO para o tipo `Real` no M4PIA no trecho 3.11.

```
30 mapping EMSOmetamodel::Type :: Type2DeclaredType() : M4PIAmetamodel::DeclaredType
       disjuncts
31   EMSOmetamodel::_Real :: Real2Real1,
32   EMSOmetamodel::_Integer :: Integer2Integer1,
33   EMSOmetamodel::_Boolean :: Boolean2Boolean1 {}
```

Trecho de Código 3.10: Mapeamento dos tipos de dados declarados.

```
115 mapping EMSOmetamodel::_Real :: Real2Real(cons:Boolean) : M4PIAmetamodel::_Real
116 when {self.oclIsKindOf(EMSOmetamodel::_Real)} {
117   result.ID := self.ID;
118   result.name := self.ID;
119   result.brief := self.Brief;
120   result._default := self.Default;
121   result.lower := self.Lower;
122   result.upper := self.Upper;
123   result.unit := self.Unit;
124   result.displayunit := self.DisplayUnit;
125   result.unit := self.FinalUnit;
126   result.base := self.base.late resolveone(M4PIAmetamodel::_Real);
127   if cons then { result.constant := true} else { result.constant := false} endif
128 }
```

Trecho de Código 3.11: Continuação do mapeamento dos tipos de dados declarados para *Real2Real*.

Cada `Model` no EMSO correspondente conforme o metamodelo EMSO é mapeado para um `Equipment` do M4PIA, de acordo com as regras apresentadas no código 3.12. O parâmetro do modelo matemático no EMSO é mapeado como um atributo do equipamento do tipo constante. Se for uma variável, o mapeamento do atributo do equipamento é tido como não constante. Cada `Equation` no modelo matemático de um `Model` é transformado em um `Method` do `Equipment` pelo mapeamento *Equation2Method*. Este trecho de código que apresenta o mapeamento do modelo matemático do PSM EMSO para o método PIM é apresentado no trecho de código 3.13. Além do mais, se a linguagem de modelagem EMSO tiver sido utilizada na definição da equação PSM, o código do método corresponde ao *Expression* da equação do modelo fonte, e caso a equação tenha sido declarada em outra linguagem, o código e a linguagem são incluídos na descrição da equação.

```
70 mapping EMSOmetamodel::Model :: Model2Equipment() : M4PIAmetamodel::Equipment{
71   result.ID := self.Name;
72   result.name := self.Name;
73   result.description := self.Info;
74   result.base += self.BaseModel.late resolveone(M4PIAmetamodel::Equipment);
75   result.method += self.equation->Equation2Method();
76   result.brief:= self.Brief ;
77   result.icon := self.Icon;
78   result.public := self.Pallete;s
79   result.attr +=  self.parameter -> map Parameter2Attribute(true);
80   result.attr +=  self.variable -> map Variable2Attribute(false);
81 }
```

Trecho de Código 3.12: Mapeamento de `Models` do PSM para `Equipment` do PIM.

```
167   result.name := self.Name;
168   if self.Name.equalsIgnoreCase("EMSO") then {
169     result.description := self.Description;
170     result.code := self.Expression;
171   }
172   else {
173     result.description:= self.Description  + "\n ( In language: "+ self.Name + "\
          n Code:\n" + self.Expression + "\n)";
174   }
175   endif
176 }
```

Trecho de Código 3.13: Mapeamento de equações de modelos para metódos do PIM.

Como discutido anteriormente, no trecho de código 3.10, o mapeamento dos atributos numéricos e lógicos seguem esta mesma estrutura para: *Real2Real*, *Integer2Integer* e *Boolean2Boolean*, conforme o trecho 3.14. Além do mais, o mapeamento dos parâmetros e variáveis para atributos são análogos. A distinção entre ambos se dá no parâmetro de entrada do tipo *Boolean* atribuído na transformação *Model2Equipment* (código 3.12) que confere se o parâmetro *Boolean* é ou não verdadeiro. Se for, o atributo constante do

equipamento M4PIA será verdadeiro. Caso contrário, então é o atributo constante do equipamento M4PIA será falso.

```
83 mapping EMSOmetamodel::Variable :: Variable2Attribute(cons:Boolean) :
       M4PIAmetamodel::Attribute disjuncts
84   EMSOmetamodel::NonTyped :: NonTyped2NonTyped,
85   EMSOmetamodel::_Real :: Real2Real,
86   EMSOmetamodel::_Integer :: Integer2Integer,
87   EMSOmetamodel::_Boolean :: Boolean2Boolean,
88   EMSOmetamodel::ModelType :: ModelType2EquipmentType{}
89
90
91 mapping EMSOmetamodel::Parameter :: Parameter2Attribute(cons:Boolean) :
       M4PIAmetamodel::Attribute disjuncts
92   EMSOmetamodel::NonTyped :: NonTyped2NonTyped,
93   EMSOmetamodel::_Real :: Real2Real,
94   EMSOmetamodel::_Integer :: Integer2Integer,
95   EMSOmetamodel::_Boolean :: Boolean2Boolean,
96   EMSOmetamodel::Switcher :: Switcher2String,
97   EMSOmetamodel::ModelType :: ModelType2EquipmentType{}
```

Trecho de Código 3.14: Separação dos mapeamento de parâmetros e variáveis do modelo matemático EMSO para atributos do PIM.

.

Também são definidos os mapeamentos de variáveis e parâmetros não tipados e do tipo modelo *ModelType2EquipmentType*. Este pode ser visto no trecho 3.15 e é incumbido por dar a característica de composição para a classe: um modelo pode ser composto por outros submodelos, definidos como atributos. Os parâmetros do PSM podem ser do tipo textual e conter opções de escolhas, sendo mapeados para `String` no PIM, como apresentado no trecho de código 3.16.

```
100 mapping EMSOmetamodel::ModelType :: ModelType2EquipmentType(cons:Boolean) :
        M4PIAmetamodel::EquipmentType
101 when {self.oclIsTypeOf(EMSOmetamodel::ModelType)}{
102   result.ID :=self.ID;
103   result.name := self.ID;
104   result.brief := self.Brief;
105   result.base := self.base.late resolveone(M4PIAmetamodel::Equipment);
106   if cons then { result.constant := true} else { result.constant := false} endif
107 }
```

Trecho de Código 3.15: Mapiamento da composição de modelos por outros modelos definidos como seus atributos.

.

.

3.3.2 Transformações de Texto para Modelo PSM

As transformações T2M (*Text-to-Model* - Texto para Modelo) devem ser capazes de analisar um arquivo de texto (código fonte) da plataforma específica e extrair da sua

```
152 mapping EMSOmetamodel::Switcher :: Switcher2String(cons:Boolean) : M4PIAmetamodel
        ::_String
153 when {self.oclIsTypeOf(EMSOmetamodel::Switcher)}{
154   log('Switcher2String \t\t' +self.ID);
155   result.ID :=self.ID;
156   result.name := self.ID;
157   result.brief:= self.Brief ;
158   result.stringoption += self.switcheroption->SwitcherOption2StringOption();
159   if cons then { result.constant := true} else { result.constant := false} endif
160 }
161
162 mapping EMSOmetamodel::SwitcherOption :: SwitcherOption2StringOption() :
        M4PIAmetamodel::StringOption {
163   result.value :=self.ID;
164 }
```

Trecho de Código 3.16: Mapeamento de dados do tipo textual e sua opções de escolha.

estrutura dados relevantes como, por exemplo, objetos lógicos de um modelo de dados conceitual. Neste caso isso significa identificar os elementos principais que compõem as plataformas específicas MPA e EMSO e obter seus dados e popular adequadamente um objeto do modelo com essas informações. As figuras 3.8 e 3.9 ilustram a estrutura da transformação da plataforma MPA e EMSO, respectivamente.

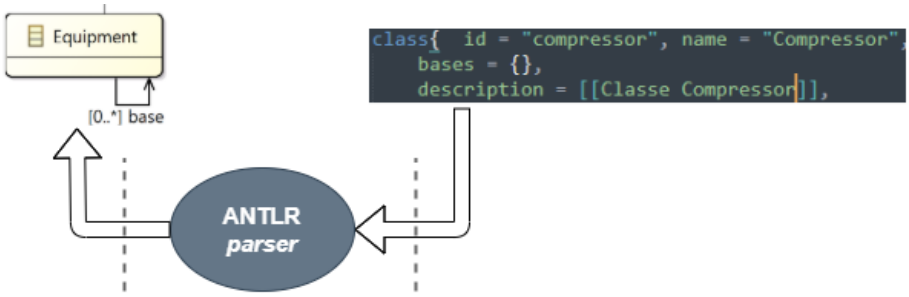

Figura 3.8: Estrutura da Transformação T2M (*Text-to-Model* - Texto para Modelo) da plataforma MPA.

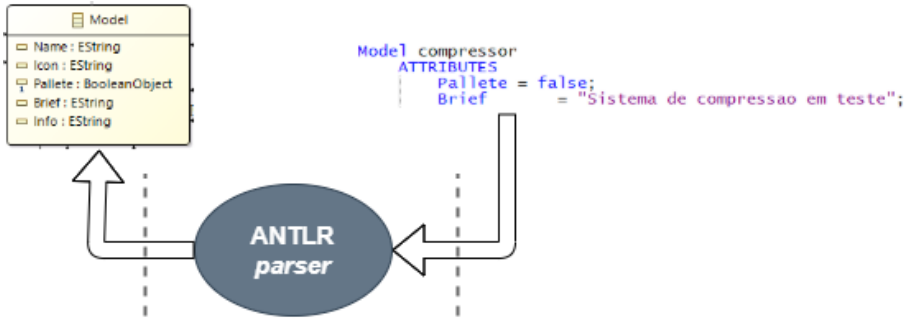

Figura 3.9: Estrutura da Transformação T2M (*Text-to-Model* - Texto para Modelo) da plataforma EMSO.

3.3.2.1 Transformação T2M Reversa:Código-fonte para PSM/MPA

Para a criação dessas transformações utilizou-se a ferramenta ANTLR, descrita em 2.3.2. Uma transformação de leitura e análise de um arquivo fonte deve ser capaz de identificar e separar cada um dos seus elementos sintaticamente. Sendo assim, as gramáticas a serem definidas utilizando o ANTLR precisam incorporar esse conhecimento.

No fluxo de desenvolvimento de transformações por meio do ANTLR, é necessário definir as gramáticas de *lexer* e *parser* que abrangem a linguagem a ser compreendida pela transformação. A gramática de *lexer* é responsável por agrupar os caracteres em *tokens* identificáveis, enquanto a gramática de *parser* identifica as regras da linguagem com base na entrada tokenizada e monta uma estrutura em árvore AST com as informações presentes no arquivo. Para o sistema MPA, foi desenvolvida uma gramática de *parser* condizente com a linguagem em que fosse possível extrair as informações relevantes da linguagem. A figura 3.10 apresenta um trecho da programação de um *lexer* para a plataforma MPA. A figura 3.11 apresenta um trecho da programação de um *parser* para a plataforma MPA.

```
1  lexer grammar MPALexer;
2  @header{ package m4pia.antlr4.mpa; }
3
4  ALTFNC      : 'func{'  -> pushMode(TRY);
5  ALTCLASS    : 'class{' -> pushMode(TRY);
6
7  FNOPEN      : OPKY1 EQ* OPKY1;
8  FNCLOSE     : CLKY1 EQ* CLKY1 -> popMode;
9  OPKY1       : '[';
10 CLKY1       : ']';
11
12 ALTCLCOL    : '}';
13 ALTOPCOL    : '{';
14 ALTSTRCODE  : 'code';
15 ALTWS       : [ \t\r\n ]+ -> skip ;
16 ALTEQ       : '=';
17
18 STRASSERT   : 'assert';
19 QTALIEN     : '"alien"';
20 STRALIEN    : 'alien';
21 STRREQ      : 'require';
22 ALTOPPAR    : '(' -> pushMode(INSIDE);
23 ALTCLPAR    : ')';
24
25 TD          : ':';
26 ALTDT       : '.';
27 STRTYPES    : 'types';
28 STRINC      : 'include';
29
30 ALTCMA      : ',' ->pushMode(TRY);
31 ALTID       : [a-zA-Z0-9_+]+;
```

Figura 3.10: Trecho do código do *lexer* para a plataforma MPA.

A partir da gramática do *parser* e *lexer*, origina-se uma outra classe de implementação em linguagem de programação Java chamada de `MPAFileParser.java` e uma outra

```
1    parser grammar MPAParser;
2  @header{package m4pia.antlr4.mpa;}
3
4  options { tokenVocab = MPALexer; }
5
6  doc : ( mpafunction | dllblock | equipment | include | code )* EOF;
7
8  mpafunction : FUNC (id | name | description | code | parameters | results | EMPTY)+ (CLCOL | ALTCLCOL);
9  equipment   : (CLASS | ALTCLASS) (id | name | description | group | code | base | isPoint | simVal | attributes | EMPTY|  methods)+  (ALTCLCOL| CLCOL) ;
10 include     : STRINC ALTOPPAR QTEXT1 ICLPAR;
11
12 methods     : STRMETH EQ OPCOL  method* CLCOL ALTCMA;
13 method      : OPCOL (id | description | name | parameters | results | code)+ (CLCOL (ALTCMA)* | CLOSEANDJUMP) ;
14
15 paramPair   : OPCOL (name | type)+ (deft | simVal)* ( CLOSEANDJUMP | CLCOL ) ;
16 parameters  : (STRPRM | STRATTR) EQ OPCOL (paramPair (CMA| ALTCMA)*)* CLCOL (CMA| ALTCMA) | CLOSEANDJUMP;
17 results     : STRRES EQ OPCOL (paramPair (CMA| ALTCMA))* CLCOL (CMA| ALTCMA) | CLOSEANDJUMP;
18 attributes  : STRATTR EQ OPCOL (OPCOL (name | type | id | description | access | isPoint | deft | simVal)* ( CLOSEANDJUMP | CLCOL ) (CMA| ALTCMA)*)* CLCOL (CMA| ALTCMA) | CLOSEANDJUMP;
```

Figura 3.11: Trecho do código do *parser* para a plataforma MPA.

classe auxiliar chamada de `MPAFileParserBaseListiner.java`. A classe de implementação do *parser* recebe como entrada um arquivo que virá *tokerizado* pela etapa anterior, e, sendo assim, possível a geração de uma *Abstract Syntax Tree* (AST) de acordo com as regras definidas pela gramática. Cada uma das regras definidas no *parser* torna-se em um nó da árvore AST ao final da transformação de leitura do arquivo.

Após a geração da AST, é possível navegar através da árvore completa, executando funções a cada nó ou folha de interesse. Na classe `MPAFileParserBaseListiner.java` são declarados automaticamente cabeçalhos dessas funções de entrada e saída para cada regra presente a linguagem. Assim, definiu-se uma nova classe que sobrescreve as funções geradas no `BaseListener`, com o objetivo de extrair as informações sobre equipamentos e funções do código-fonte de pré-configuração da plataforma de operação, automação e controle MPA. A figura 3.12 apresenta um trecho de código com a extração de informações das classes de equipamentos da plataforma MPA.

```
 99    @Override public void enterEquipment(MPAParser.EquipmentContext ctx) {
100        Equipment equipments = factory.createEquipment();
101        archive.getComponent().add(equipments);
102
103            if(ctx.id(0) != null)
104                equipments.setID(remQuote(ctx.id(0).QTEXT().getText()));
105            if(ctx.name(0) != null)
106                equipments.setName(remQuote(ctx.name(0).QTEXT().getText()));
107            if(ctx.group(0) != null)
108                equipments.setGroup(remQuote(ctx.group(0).QTEXT().getText()));
109            if(ctx.description(0) != null)
110                equipments.setDescription(remQuote(ctx.description(0).QTEXT().getText()));
111    }
```

Figura 3.12: Trecho do código de extração de informações da classe de equipamentos para a plataforma MPA.

Uma outra classe foi desenvolvida, em linguagem Java devido ao suporte da ANTLR, para executar as transformações, chamando o `BaseListener` de modo que é possível percorrer a árvore, chamando as funções de entrada e saída de nós. Ao final desta etapa, obtém-se listas contendo as informações coletadas a partir da varredura na árvore que gera a como saída um modelo populado com as informações extraídas por meio desse processo.

3.3.2.2 Transformação T2M Reversa:Código-fonte para PSM/EMSO

De modo similar ao desenvolvimento da transformação T2M para a plataforma MPA, ocorre o processo de desenvolvimento da transformação T2M para a plataforma EMSO. Utilizou-se para criação dessa transformação a ferramenta ANTLR, descrita em 2.3.2. Uma transformação de leitura e análise de um código-fonte precisa ter a capacidade de identificar e separar cada um dos componentes presentes no código de maneira sintática. Sendo assim, as gramáticas a serem definidas utilizando o ANTLR precisam incorporar esse conhecimento.

No fluxo de desenvolvimento de transformações por meio do ANTLR, é necessário definir as gramáticas de *lexer* e *parser* que abrangem a linguagem a ser compreendida pela transformação. A gramática de *lexer* é responsável por agrupar os caracteres em *tokens* identificáveis. Por sua vez, a gramática de *parser* identifica as regras da linguagem com base na entrada tokenizada e monta uma estrutura em árvore AST com as informações presentes no arquivo. Para o sistema EMSO, desenvolveu-se uma gramática de *parser* de modo adequado com a linguagem para seja possível extrair as informações relevantes da linguagem. A figura 3.13 apresenta um trecho da programação de um *lexer* para a plataforma EMSO. A figura 3.14 apresenta um trecho da programação de um *parser* para a plataforma EMSO.

```
2⊕ * Define a grammar called EMSO
4  grammar EMSO;
5
6⊕ @header {
9
10 //tokens {
11 FLOWSHEET       : 'FlowSheet' ;
12 MODEL           : 'Model' ;
13 ESTIMATION      : 'Estimation' ;
14 OPTIMIZATION    : 'Optimization' ;
15 RECONCILIATION  : 'Reconciliation' ;
16 CASE_STUDY      : 'CaseStudy';
17 SENSITIVITY     : 'Sensitivity';
18 UNCERTAINTY     : 'Uncertainty';
19
20 TIME       : 'time';
21 USING      : 'using' ;
22 END        : 'end' ;
23 AS         : 'as' ;
24 IN_        : 'in';
25 OUT_       : 'out';
26 OUTER      : 'outer';
27 TO         : 'to';
```

Figura 3.13: Trecho do código do *lexer* para a plataforma EMSO.

A partir da gramática do *parser* e *lexer*, origina-se uma outra classe de implementação

```
177 parse : (using_ | type | model | flowsheet | optimization | estimation | reconciliation | case_study | sensitivity | uncertainty)* EOF ;
178
179 using_ : USING STRING (COMMA STRING)* SEMI ;
180
181 model  : MODEL ID (inheritance )?
182     (attributes | variables | parameters | set | specify | equations | initial
183     | guess | connections )*
184     END ;
```

Figura 3.14: Trecho do código do *parser* para a plataforma EMSO.

em linguagem de programação Java chamada de `EMSOParser.java` gerado pelas ferramentas do ANTLR, e uma outra classe auxiliar chamada de `EMSOBaseListiner.java`. A classe de implementação do *parser* recebe como entrada um arquivo *tokerizado* pela etapa anterior, e, sendo assim, possível a geração de uma *Abstract Syntax Tree* (AST) de acordo com as regras definidas pela gramática. Cada uma das regras definidas no *parser* torna-se, consequentemente, em um nó da árvore AST ao final da transformação de leitura do arquivo.

Após a criação da AST, pode-se navegar por meio da árvore completa, executando funções a cada nó ou folha de interesse. Na classe `EMSOBaseListener.java` são declarados automaticamente cabeçalhos dessas funções de entrada e saída para cada regra presente a linguagem. Assim, definiu-se uma nova classe que sobrescreve as funções geradas no `BaseListener`, com o objetivo de extrair as informações sobre equipamentos e funções do código-fonte da plataforma de simulação EMSO. A figura 3.15 apresenta um trecho de código com a extração de informações das classes de modelos da plataforma EMSO.

```
146     @Override public void enterModel(EMSOParser.ModelContext ctx) {
147
148         Model tempModel = factory.createModel();
149         archive.getEntity().add(tempModel);
150
151         if(ctx.ID() != null)
152             tempModel.setName(ctx.ID().getText());
153
154         int i = 0;
155         while(ctx.attributes(0).derivation(0).exprString() != null) {
156         if(ctx.attributes(0).derivation(0).exprString().expr().logicalExpr().rangeExpr().addExpr(0).multExpr(0).
157             tempModel.setPallete(true);
158         } else if (ctx.attributes(0).derivation(i).exprString().expr().logicalExpr().rangeExpr().addExpr(0).multi
159             tempModel.setPallete(false);
160         }
161         if(ctx.attributes(0).derivation(i).getText().contains("Brief"))
162             tempModel.setBrief(ctx.attributes(0).derivation(i).exprString().getText());
163
164         if(ctx.attributes(0).derivation(i).getText().contains("Info"))
165             tempModel.setInfo(ctx.attributes(0).derivation(i).exprString().getText());
166
167         if(ctx.attributes(0).derivation(i).getText().contains("Icon"))
168             tempModel.setIcon(ctx.attributes(0).derivation(i).exprString().getText());
169         }
170     }
```

Figura 3.15: Trecho do código de extração de informações da classe de modelos para a plataforma EMSO.

Posteriormente, desenvolveu-se uma classe, também, em linguagem Java, para executar as transformações, chamando o `BaseListener` de modo que é possível percorrer a árvore, chamando as funções de entrada e saída de nós. Ao final, obtém-se listas

contendo as informações coletadas a partir da varredura na árvore que gera a como saída um modelo populado com as informações extraídas por meio desse processo.

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentada a proposta desenvolvida para integrar a RE na infraestrutura do M4PIA no domínio de definição de classes de equipamentos para aplicações de simulação, operação e controle de plantas na indústria petroquímica. Foram descritos os metamodelos e transformações que compõem a solução e o processo de concepção e utilização da infraestrutura.

4 AVALIAÇÃO DA PROPOSTA

A apresentação do capítulo presente se dará por um exemplo de aplicação da infraestrutura M4PIA através de uma prova conceito. Partindo deste estudo, a solução proposta neste trabalho é analisada por meio de uma discussão dos aspectos desejados para a infraestrutura e artefatos que fazem parte de sua composição.

Será apresentada uma aplicação da RE em uma planta simplificada de compressão de gás de plataforma de petróleo como ilustração e validação da utilização da infraestrutura M4PIA. Está aplicação foi utilizada nas interações de criação das transformações, e na análise inicial da solução.

4.1 PROVA DE CONCEITO: SISTEMA DE COMPRESSÃO DE GÁS

Nas plantas *off-shore* de produção de petróleo e gás, os sistemas de compressão de gás são responsáveis por fornecer energia ao gás, aumentado sua pressão e proporcionando vazão de gás a uma pressão específica, dependendo do ponto de operação requerido e as especificações dos sistemas que receberão o gás comprimido. Após ser comprimido pelo sistema, o gás pode ser exportado para terra por intermédio de gasodutos ou navios, sendo usado para geração de energia elétrica na plataforma, reinjetado no reservatório para manutenção da pressão ou usando no método de elevação artificial de petróleo mediante a injeção de *gas lift*.

O sistema de compressão necessita processar a vazão de gás fornecida pelo vaso de entrada. Se o compressor vir a falhar, o gás irá acumular e aumentar a pressão dentro do vaso. Para que a pressão não supere os limites de segurança, uma parte do gás deve ser queimada (*flare*). Uma vazão de gás na entrada vaso abaixo dos limites especificados pode levar o compressor a uma região de operação instável (*surge*). Para evitar a parada do compressor, cada estágio de compressão tem uma linha de reciclo com uma válvula anti-surge.

Sendo um sistema de grande importância para o devido funcionamento de toda a produção, o controle das variáveis de um sistema de compressão em uma plataforma é objeto de muitos estudos na área de controle e otimização de processos e inclui a definição de inúmeras relações entre vazões, pressões, temperatura, aberturas de válvulas, entre diversos outros estados e atributos os sistema e seus componentes.

Para esta prova conceito do trabalho, o sistema simplificado de compressão de gás utilizado é composto por um vaso de surge com uma válvula de flare (alívio de pressão) na entrada, dois compressores de estágio em paralelo e um header de saída. Cada estágio de compressão possui um trocador de calor, um vaso de sucção, um elemento

compressor e uma válvula de reciclo (anti-surge). O header de saída possui três válvulas de controle: uma válvula de exportação e duas de saída para injeção de *gas lift*. A figura 4.1 ilustra o sistema de compressão de gás descrito.

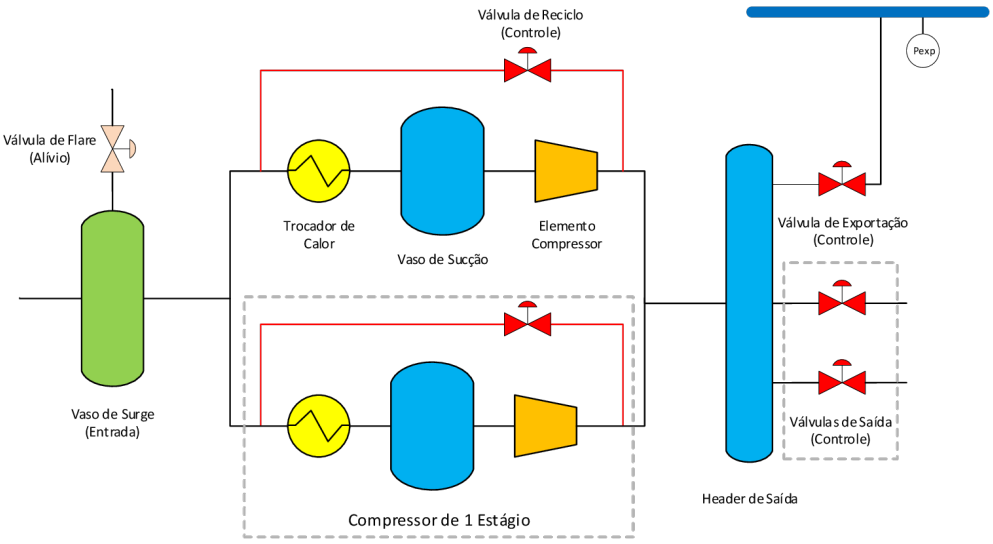

Figura 4.1: Sistema simplificado de compressão de gás (DAMO, 2019).

4.2 MODELAGEM DO SISTEMA DE COMPRESSÃO

Para a modelagem do sistema, a partir do paradigma orientado a objetos, é necessário a análise do domínio da aplicação para definir os equipamentos de interesse, os atributos e os métodos. As classes de equipamentos que fazem parte da aplicação da prova conceito pode ser modeladas como componentes de um projeto único do sistema de compressão simplificado. Uma boa prática para modelagem dos sistemas é procurar pela generalização, um exemplo, é o sistema de compressão pode ter um ou mais compressores e um compressor pode ter um ou mais estágios. Também usando o conceito de hierarquia, como uma válvula que pode ser do tipo válvula de alívio ou válvula de controle. Afim de testar a engenharia reversa desenvolvida, as modelagens do sistema de compressão simplificado foram realizadas manualmente nos códigos-fontes, ou seja, no mais baixo nível da infraestrutura M4PIA, em ambas as plataformas: MPA em linguagem LUA e na plataforma EMSO em linguagem EMSO. Ambas as modelagem buscaram atender as especificidades de cada plataforma e seguir a mesma estrutura.

Para o sistema escolhido como prova conceito, o equipamento `Válvula` tem como atributos valores reais de `Abertura`, `Vazão`, `Temperatura` e `Pressão a Montante`. `Válvula de Controle` é uma especialização de `Válvula` com os atributos reais `KC` da válvula e `Pressão a Jusante`.

Um equipamento Vaso possui Pressão e Temperatura como atributos com valores reais e pode ser estendido para o equipamento Vaso de Surge com atributos reais Vazões de Entrada e Vazões de Saída e uma Válvula de Flare do tipo Válvula de Alívio. O equipamento Header de Saída também é um Vaso e tem como atributos um valor real de Vazão de Entrada, uma Válvula de Exportação e uma Lista de Válvulas de Saída do tipo Válvulas de Controle.

Um equipamento Trocador de Calor tem por atributos os valores reais Pressões de Sucção e Descarga e Temperatura de Sucção e Descarga. O equipamento Elemento de Compressão possui os atributos reais Vazão, Pressões de Sucção e Descarga e Temperatura de Sucção e Descarga.

Um equipamento Estágio de Compressão é composto por: um valor real de Potência, um Trocador de Calor, um Vaso de Sucção do tipo Vaso, um Elemento de Compressão e uma Válvula de Reciclo do tipo Válvula de Controle.

Um equipamento Compressor possui como atributos uma Lista de Estágios de Compressão, valores reais para a Potência Real e a Potência Nominal. A Potência Real do Compressor é calculada por um método que soma a Potência de todos os Estágios do Compressor e divide pela Potência Nominal.

O equipamento Sistema de Compressão possui como atributos: o valor real da Potência, um Vaso de Entrada do tipo Vaso de Surge, um Header de Saída e uma Lista de Compressores. A Potência é calculada por um método que soma a Potência de todos os Compressores do Sistema de Compressão e divide pelo número de compressores.

Foram modeladas manualmente ao todo 11 classes de equipamentos, com seus 36 atributos e 2 métodos para a plataforma MPA. Para a plataforma EMSO foram modelados, também de modo manual, 11 modelos, 2 métodos, 22 variáveis e 16 parâmetros. A modelagem ocorreu para cada plataforma contemplada na infraestrutura M4PIA, tanto o MPA como a plataforma EMSO, de acordo com cada especificidade da linguagem de cada plataforma afim de exemplificar um sistema legado de uma planta simplificada. Os códigos-fonte completos são encontrados nos apêndices A e B. A seguir, são ilustrados o trecho de código 4.1 da classe compressor na plataforma MPA e o trecho de código 4.2 da classe compressor para a plataforma EMSO.

4.3 CENÁRIOS DE AVALIAÇÃO DA PROVA CONCEITO

Para testar as transformações desenvolvidas neste trabalho, foi proposta uma avaliação empírica com quatro cenários diferentes na cadeia de transformações como

```
203 class{ id = "compressor", name = "Compressor", group = "Compressor",
204   bases = {},
205   description = [[Compressor equipment and its stages]],
206   attributes = {
207      {  id = "power", name = "Power", type = "REAL",
208       access = "grw",
209       description = [[Compressor total power W]],
210     },
211     { id = "nominal_power", name = "Nominal power" type = "REAL",
212       access = "g",
213       description = [[Compressor nominal power W]],
214     },
215     { id = "list_stages", name = "List of the compression stages" type = "
          compression_stage[]",
216       access = "g",
217       description = [[List of compression stages]],
218     },
219   },
220   codes = [[]],
221   methods = {
222     { id = "calculate_power", name = "Calculate the compressor normalized power",
223       description = [[Summing the power of all stages]],
224       parameters = {},
225       results = {
226         { name = "Normalized power of the compressor W", type = "REAL" },
227       },
228       code = [===[
229             function()
230               local power_compressor = 0
231               local power_nominal = self.nominal_power
232               for r2, ref2 in ipairs(self.list_stages) do
233                 power_compressor + ref2.power:read()
234               end
235               power_compressor = power_compressor/power_nominal
236               self.power:write(power_compressor)
237               return power_compressor
238             end
239           ]===],
240     },
241   },
242 }
```

Trecho de Código 4.1: Trecho de código da classe Compressor modelada no MPA.

apresentado na figura 4.2 a partir da modelagem do sistema de compressão simplificado apresentado na seção 4.1. O objetivo é analisar a engenharia reversa implementada na infraestrutura cobrindo todas as transformações possíveis na cadeia.

O primeiro cenário de transformações tem como ponto de partida o código fonte MPA e vai até o modelo PIM M4PIA, e depois o caminho inverso é realizado novamente até o código-fonte MPA. No segundo, o código-fonte EMSO é a origem e vai até o modelo PIM M4PIA e depois o retorno das transformações até o código-fonte EMSO. O terceiro cenário, parte do código fonte MPA para o modelo PIM M4PIA e depois vai para o código-fonte EMSO. Por fim, o quarto cenário parte do código-fonte EMSO e vai para o modelo PIM M4PIA, em seguida, para o código-fonte MPA.

A validação busca detectar empiricamente as possíveis perdas de informações e as informações trafegadas com sucesso nas transformações entre uma mesma plataforma

```
120 Model compressor
121
122   ATTRIBUTES
123     Pallete = false;
124     Info = "Compressor equipment and its stages";
125
126   PARAMETERS
127     nominal_power as Real (Brief="Compressor total power W");
128     list_stages(n_compression_stage) as compression_stage (Brief="List of
          compression stages");
129     n_compression_stage as Integer (Brief= "Number of elements of
          compression_stage");
130
131   VARIABLES
132     power as Real (Brief="Compressor total power W");
133
134   EQUATIONS
135     # Summing the power of all satages
136     "Calculate the compressor normalized power"
137     power = sum(list_stages.power)/nominal_power;
138 end
```

Trecho de Código 4.2: Trecho de código da classe Compressor modelada no EMSO.

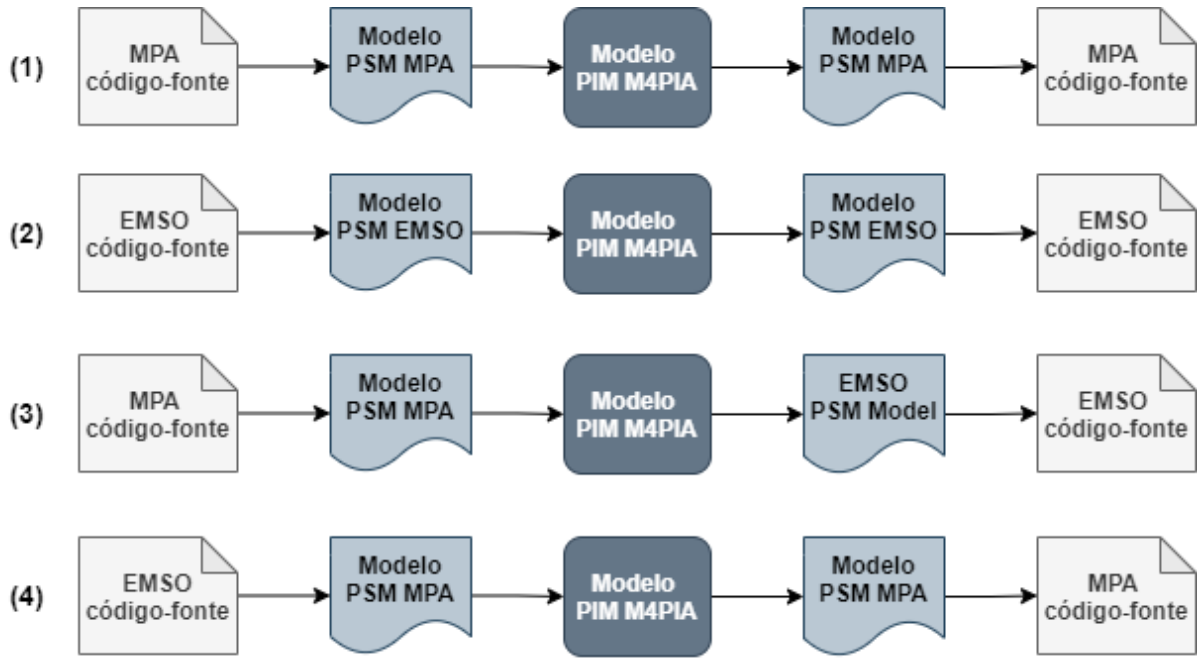

Figura 4.2: Cenários para validação da prova conceito.

específica e, também, de uma plataforma específica a outra por meio do modelo de mais alto nível PIM M4PIA. Nestes cenários, não está incluído nenhum refinamento em nenhuma das etapas das transformações.

De acordo com Fenton and Bieman (2019), ma métrica de qualidade largamente usada nas avaliações da Engenharia de Software é o número de linhas de código *Lines of Code* (LOC) por representar o esforço de desenvolvimento e manutenção, sendo associado aos custos e tempo de desenvolvimento, produtividade dos desenvolvedores, qualidade do código (taxa de erros detetados/LOC) e complexidade de manutenção.

As medições e avaliação pelas LOC são úteis, entretanto possuem limitações como, por exemplo, a diferença de complexidade de programação entre as linhas de código (linhas em branco, comentários, declarações de variáveis ou linhas de instruções) e da estruturação de um mesmo código por cada desenvolvedor.

A definição mais aceita na literatura para contar linhas de código das aplicações inclui somente linhas de códigos efetivas. Os comentários e linhas em branco não são contabilizados (FENTON; BIEMAN, 2019). Vale ressaltar que não há intensão de desconsiderar a relevância dos comentários inseridos pelos desenvolvedores. O estilo de escrita do desenvolvedor não necessariamente corresponde à estrutura definida nos *templates*, podendo as diferenças entre as quebras de linhas serem significativas neste caso.

4.3.1 Cenário 1: Código-fonte MPA - PIM M4PIA - Código-fonte MPA

No primeiro cenário da prova conceito, tem-se o código-fonte MPA, contendo a modelagem do sistema de compressão simplificado, como fonte para as transformações. A primeira transformação realizada é a T2M (*Text-to-Model* - Texto para Modelo), em que por meio dos *parsers* desenvolvidos para a linguagem LUA usada pela plataforma MPA, gerou-se uma AST (*Abstract Syntax Tree*) para extrair os textos contidos em partes chaves do código. A partir disto, é possível obter o modelo PSM da plataforma MPA populando o modelo. Nessa primeira etapa do cenário nota-se que algumas informações do código-fonte foram perdidas na transformação T2M devido a não ser possível implementar essas informações na transformação. Por exemplo, o modelo PSM não armazena nenhum comentário contido no código-fonte por não conter essa informação no modelo PSM. Outra informação perdida na transformação T2M da plataforma MPA são os acessos conferidos para cada atributo das classes, automaticamente os acessos são setados para falso pela transformação. Os grupos das classes do código-fonte MPA, também são perdidos pela transformação ficando com o campo nulo. Essas informações não foram implementadas na transformação T2M. A figura 4.3 apresenta a classe compressor do modelo PSM MPA gerado automaticamente.

A partir do PSM MPA obtido pela transformação T2M, realiza-se a transformação M2M do modelo PSM fonte para o modelo PIM. Essa transformação é realizada por meio da linguagem QVTo, conforme detalhado na seção 3.3.1. A transformação M2M não obteve perdas, as informações contidas no modelo PSM popularam de maneira esperada o modelo PIM. A figura 4.4 ilustra a modelagem da classe compressor no modelo PIM.

Após as execuções da transformação T2M do código-fonte LUA para o PSM MPA,

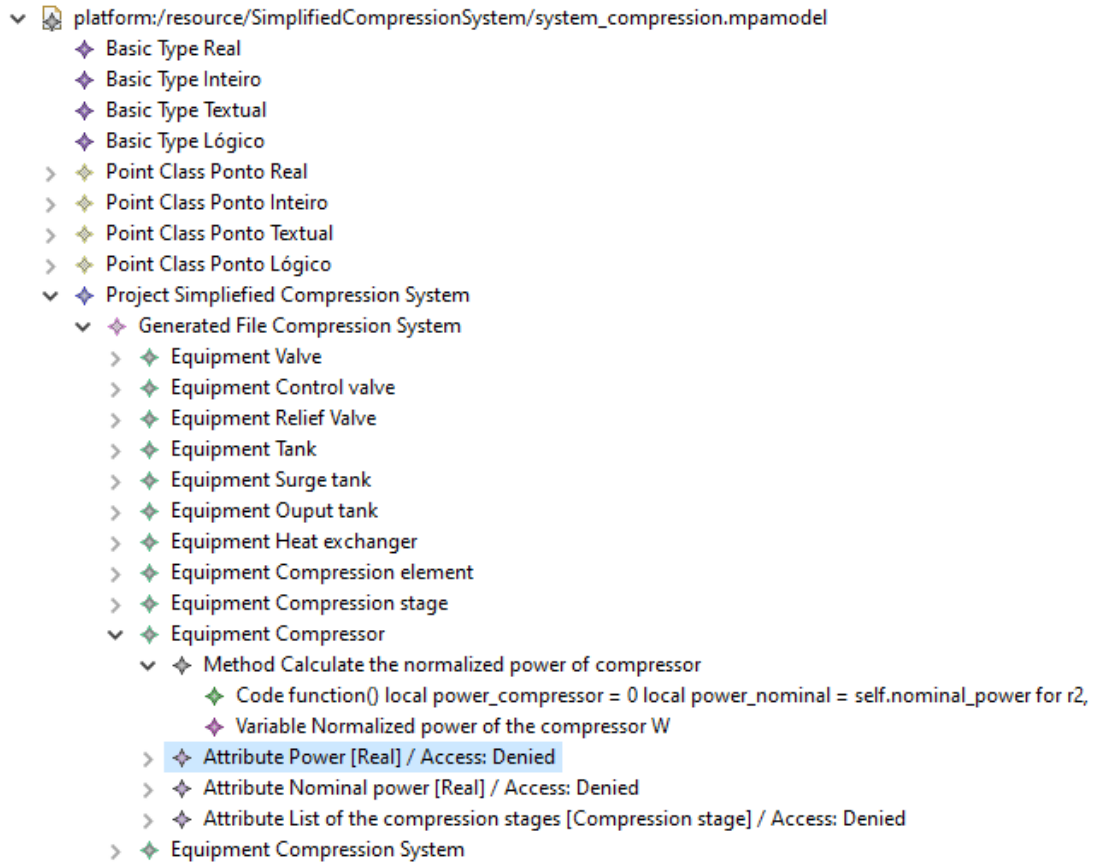

Figura 4.3: Classe compressor no modelo PSM obtida pela transformação T2M do código-fonte MPA.

e, a transformação M2M do PSM MPA para PIM, afim de obter o modelo de maior alto nível de abstração, realizou-se as transformações do mais alto nível de abstração até o mais baixo. Neste caso, a primeira transformação é a M2M do PIM até o modelo PSM da plataforma MPA. A transformação M2M em ambos os sentidos não apresentou perda de informações, neste cenário. Entretanto, a transformação do modelo PIM para o PSM MPA adicionou `Basic Types` e `Point Classes` que caracterizam o modelo MPA.

Em seguida, a transformação M2T gerou automaticamente o código-fonte na linguagem LUA para a plataforma MPA. O código-fonte gerado automaticamente mostrou-se com algumas informações perdidas em comparação com o código-fonte modelado manualmente que iniciou o cenário. As principais perdas estão em algumas especificidades dentro das classes de equipamentos, como grupo e as informações de acesso dos atributos e comentários dentro dos códigos. Segue um trecho de código 4.3 da classe compressor gerada automaticamente pelo primeiro cenário. A tabela 4.1 apresenta uma comparação quantitativa do número de equipamentos, métodos, atributos e variáveis existentes no código-fonte MPA modelado manualmente e no código-fonte MPA gerado automaticamente.

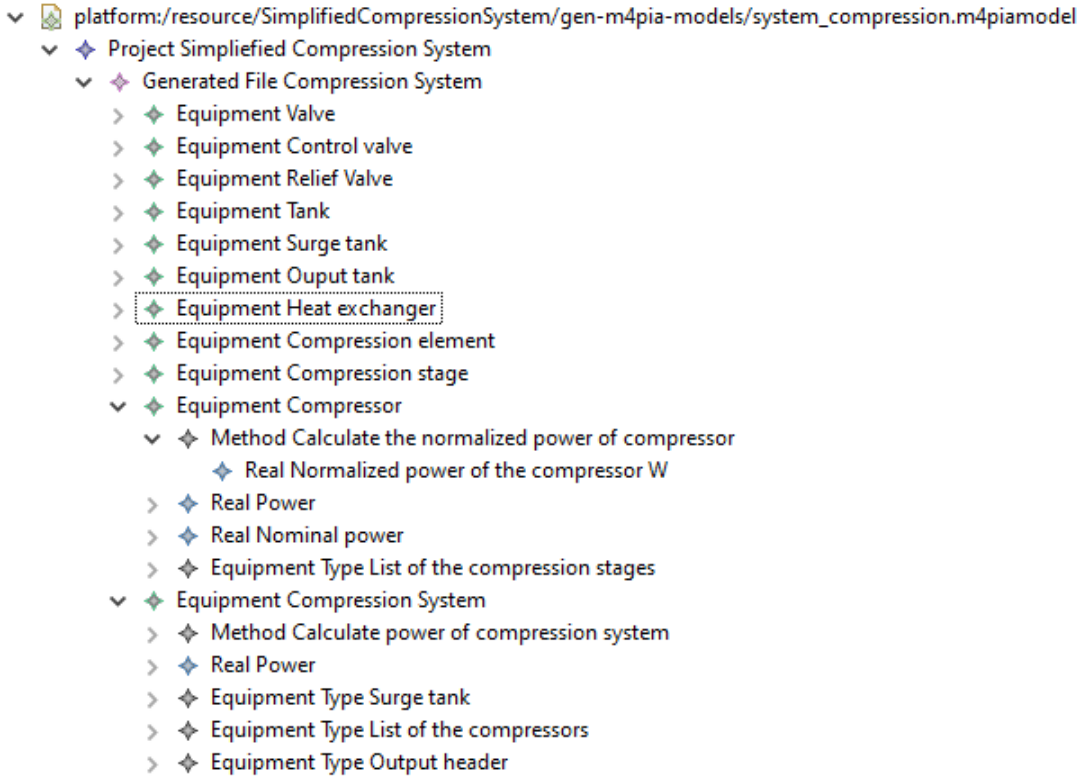

Figura 4.4: Classe compressor no modelo PIM obtida pela transformação M2M do PSM MPA.

Tabela 4.1: Cenário 1: Transformação MPA para MPA: código-fonte "Manual"vs. Gerado automaticamente.

	MPA "Manual"	MPA Gerado
Núm. de Equipamentos	11	11
Núm. de Métodos	2	2
Núm. de Atributos	36	36
Núm. de Variáveis	2	2
Lines of Code (LOC)	288	244

4.3.2 Cenário 2: Código-fonte EMSO - PIM M4PIA - Código-fonte EMSO

Similar ao cenário 1 da seção 4.3.1, o cenário 2 propõe as transformações do código-fonte EMSO até o modelo PIM M4PIA de mais alto nível de abstração. Em seguida, a partir do modelo PIM gerado automaticamente pela engenharia reversa, realiza-se a transformação até o código-fonte EMSO.

A transformação T2M do código-fonte EMSO para o modelo PSM EMSO tem o mesmo princípio de funcionamento do cenário 1 com o desenvolvimento de um *parser* para a linguagem EMSO, a partir disso, gera uma AST que atua varrendo o código-fonte com o objetivo de coletar as informações do código para popular o modelo PSM. Nesta

```
178 class{  id = "compressor", name = "Compressor", group = "",
179   bases = {},
180   description = [[Compressor equipment and its stages]],
181   attributes = {
182     { id = "power", name = "Power", type = "REAL", access = "",
183       description = [[Compressor's total power W]],
184     },
185     { id = "nominal_power", name = "Nominal power", type = "REAL", access = "",
186       description = [[Compressor's nominal power W]],
187     },
188     { id = "list_stages", name = "List of the compression stages", type = "
          compression_stage[]", access = "",
189       description = [[List of compression stages]],
190     }
191   },
192   code = [[ ]],
193   methods = {
194     { id = "compressor_calculate_power", name = "Calculate the normalized power
          of compressor",
195       description = [[]],
196       parameters = {},
197       results = {
198         {name = "Normalized power of the compressor W", type = "REAL" },
199       },
200       code = [===[ function()
201         local power_compressor = 0
202         local power_nominal = self.nominal_power
203         for r2, ref2 in ipairs(self.list_stages) do
204           power_compressor + ref2.power:read()
205         end
206         power_compressor = power_compressor/power_nominal
207         self.power:write(power_compressor)
208         return power_compressor
209       end ]===]
210     }
211   }
212 }
```

Trecho de Código 4.3: Trecho de código da classe Compressor gerado automaticamente pelo cenário 1.

transformação, nota-se que as informações não foram perdidas. As classes especializadas mantiveram suas classes pai, atributos e parâmetros foram alocados com as mesmas características para o modelo PSM. Devido a restrições definidas pelas linguagem OCL estabelecidas na infraestrutura M4PIA, algumas informações do modelo PSM foram preenchidas automaticamente. Neste caso, a regra define que o limite inferior de um dado real ou inteiro deve ser menor ou igual ao limite superior. Para isso, os campos de limites foram preenchidos automaticamente com os valores inferior e superior iguais a 0. Os comentários do código também não foram implementados na transformação T2M. A figura 4.5 ilustra o `Modelo` compressor gerado automaticamente pelas transformação T2M.

Para a transformação M2M do modelo PSM EMSO para o modelo PIM M4PIA a execução também tem o princípio de realização similar ao do cenário 1, com as regras de transformação desenvolvidas pela linguagem QVTo. A transformação não apresentou

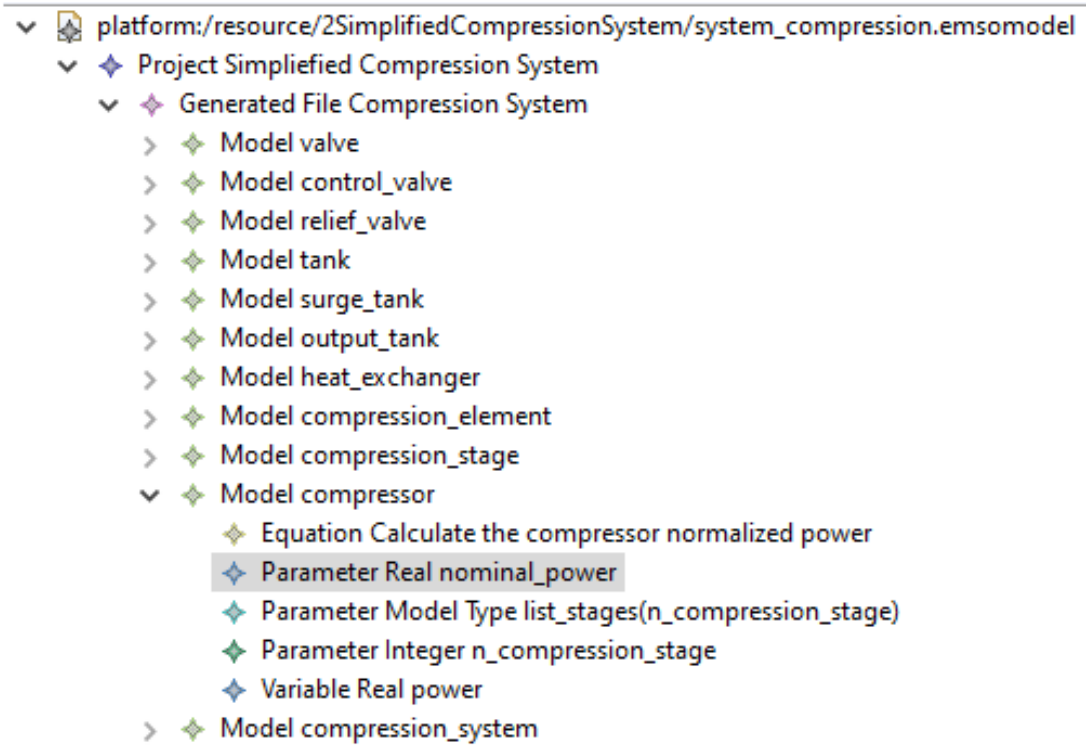

Figura 4.5: Classe compressor no modelo PSM obtida pela transformação T2M do código-fonte EMSO.

perda de informações do PSM para o modelo PIM. A figura 4.6 ilustra esta transformação com classe compressor gerada automaticamente.

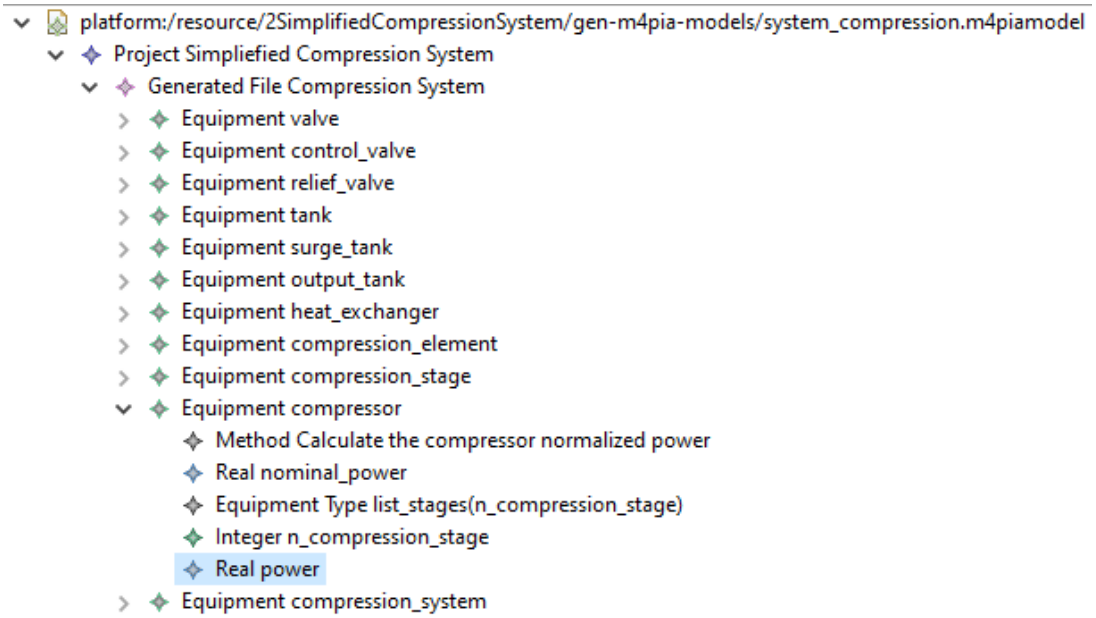

Figura 4.6: Classe compressor no modelo PIM obtida pela transformação M2M do PSM EMSO.

Após a transformação automática do código-fonte EMSO para o PSM EMSO, e, da transformação M2M PSM EMSO para PIM, realizou-se a transformação M2M

até o modelo PSM da plataforma EMSO. Observa-se que a transformação M2M não apresentou perdas de informações em ambos os sentidos. Porém, no sentido PIM para o PSM, informações foram adicionadas ao modelo PSM como os limites inferiores e superiores.

Em seguida, a transformação M2T gerou automaticamente o código-fonte na linguagem EMSO para a plataforma EMSO. O código-fonte gerado automaticamente mostra as adições de limites no parâmetro do tipo `Inteiro`, o que não havia no código-fonte modelado manualmente. Segue um trecho de código 4.4 da classe compressor gerada automaticamente pelo cenário 2. A tabela 4.2 apresenta uma comparação quantitativa do número de modelos, equações, variáveis e parâmetros existentes no código-fonte EMSO modelado manualmente e no código-fonte EMSO gerado automaticamente.

```
129 Model compressor
130
131 ATTRIBUTES
132   Pallete = false;
133   Info  = "Compressor equipment and its stages";
134 PARAMETERS
135   nominal_power as Real(Brief="Compressor total power W");
136   list_stages(n_compression_stage) as compression_stageList of compression stages
        ;
137   n_compression_stage as Integer(Brief="Number of elements of compression_stage",
        Lower=0,Upper=0);
138 VARIABLES
139   power as Real(Brief="Compressor total power W");
140 EQUATIONS
141   #Summing the power of all satages
142   "Calculate the compressor normalized power"
143   power = sum(list_stages.power)/nominal_power;
144 end
```

Trecho de Código 4.4: Trecho de código da classe Compressor gerado automaticamente pelo cenário 2.

Tabela 4.2: Cenário 2: Transformação EMSO para EMSO: código-fonte "Manual"vs. Gerado automaticamente.

	EMSO "Manual"	EMSO Gerado
Núm. de Modelos	11	11
Núm. de Equações	2	2
Núm. de Variáveis	22	22
Núm. de Parâmetros	16	16
Lines of Code (LOC)	159	154

4.3.3 Cenário 3: Código-fonte MPA - PIM M4PIA - Código-fonte EMSO

O cenário 3 tem por início o código-fonte MPA até o modelo PIM M4PIA e, em seguida, executou-se a transformação do modelo de mais alto nível para o código-fonte EMSO. O intuito principal é notar como se comporta as transformações de um código legado de uma plataforma específica para outra. A primeira parte da transformação do código-fonte MPA até o modelo de mais alto nível PIM ocorre de modo similar ao cenário 1 apresentado na seção 4.3.1. A figura 4.7 apresenta a classe sistema de compressão gerada automaticamente pela engenharia reversa de transformações.

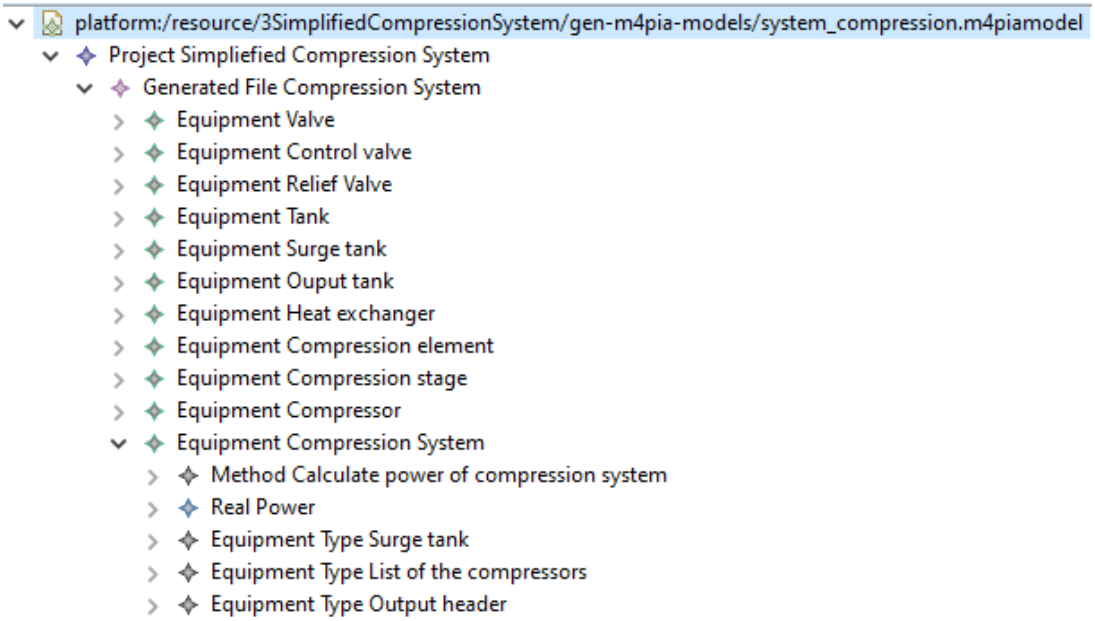

Figura 4.7: Classe sistema de compressão no modelo PIM obtida pela transformação M2M do PSM MPA.

Idealmente, é nesta etapa que o refinamento do sistema deve ocorrer, no mais alto nível de abstração. O modelo PIM foi desenvolvido para contemplar ambas as plataformas MPA e EMSO abrangendo suas particularidades e semelhanças. No entanto, para uma transformação sem refinamento, o sistema certamente apresentará perda de informações devido as particularidades de cada plataforma.

As classes no modelo MPA, contém dados de `grupo`, `ID`, `Name`, `Descrição` e `base`. A plataforma EMSO além das citadas para a plataforma MPA, o EMSO contém `Info`, `DataType` e mais o `Brief` similar a descrição, mas modelado separadamente. A transformação vindo do código-fonte MPA não contém as características das informações presentes na plataforma EMSO, resultando que estas citadas na plataforma EMSO tenham seus campos vazios.

Os `Métodos` na plataforma MPA, são equivalentes às `Equações` no modelo EMSO, ambos centralizam em `Método` na plataforma PIM. Em `Método` há opção de especi-

ficar qual linguagem dentro de `Código` será usada. O PIM gerado automaticamente vindo do código-fonte MPA contém o atributo `Linguagem` do `Código` adicionados pela transformação M2M na linguagem LUA. Isso significa que ocorrerá uma modelagem equivocada no alto nível de abstração com uma codificação não reconhecida pela plataforma EMSO no ato da transformação M2M do PIM para PSM EMSO. A figura 4.8 mostra esse ponto apresentando as informações contidas dentro do método da classe sistema de compressão transformada automaticamente.

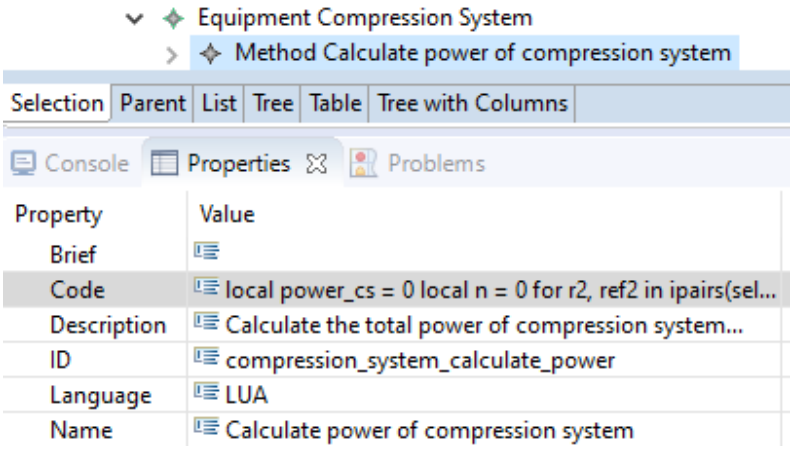

Figura 4.8: Método da classe sistema de compressão no modelo PIM obtida pela transformação M2M do PSM MPA.

Na próxima parte do cenário, realiza-se a transformação M2M do modelo PIM para o PSM EMSO e nota-se também que a transformação manteve todas as classes de equipamentos para o PSM EMSO. Os atributos das classes, agora chamados de `Modelo` no PSM EMSO, contém problemas de conceituação na transformação. Na plataforma EMSO existem os atributos chamados de `Variável` para valores que são dinâmicos e atributos chamados de `Parâmetros` para valores estáticos. Na plataforma MPA, essa conceituação não existe, independente de o atributo ser estático ou dinâmico, o que resulta que o modelo PIM, proveniente da transformação MPA não contenha esta informação. Com isso, automaticamente a transformação M2M gerou todos os atributos na categoria de `Variável`. A figura 4.9 apresenta alguns atributos do modelo PSM EMSO obtidos automaticamente.

Por fim, o código-fonte EMSO gerado automaticamente pela transformação M2T é obtido com diversos problemas da modelagem. Os atributos não contém `Brief` que atua como uma descrição textual. Os códigos contido nos métodos das classes de equipamentos foram perdidos na transformação M2M, portanto não se encontram no código. Na plataforma MPA e no EMSO existe a possibilidade de declarar um atributo sendo do tipo lista. No MPA, o atributo ser do tipo lista é preciso apenas adicionar um

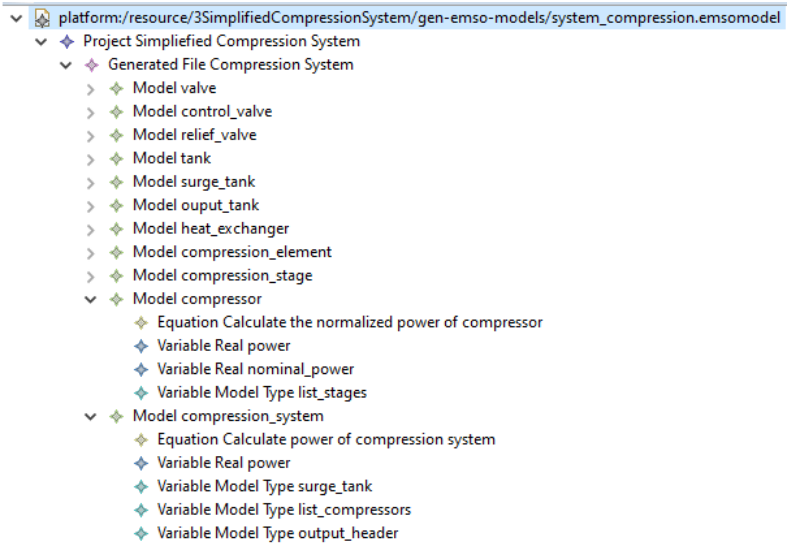

Figura 4.9: Variáveis do sistema de compressão no modelo PSM EMSO obtida pela transformação M2M do PIM M4PIA.

par de chaves junto ao nome declarado dentro do `Type`, sendo assim, necessário apenas um atributo. Entretanto para o EMSO é necessário que hajam dois parâmetros, um do tipo lista com um parêntese junto ao seu nome e o outro to tipo `Inteiro` definindo o tamanho da lista. Listas no MPA não têm um tamanho definido, mas no EMSO sim. Essa distinção entre plataformas faz com que o código gerado automaticamente do EMSO não tenha o tamanho de dois atributos do tipo lista definidos, resultando na ausência de dois atributos nas variáveis e também nenhum parâmetro, devido ao modelo PIM provido da plataforma MPA não reconhecer essa conversão e transformando todos em variáveis causando uma modelagem equivocada.

O trecho de código 4.5 apresenta a classe sistema de compressão gerada automaticamente ao fim da cadeia de transformações do cenário 3. Também é apresentada uma análise quantitativa pela tabela 4.3 onde é feita uma comparação com o código-fonte EMSO modelado manualmente, com o código fonte EMSO gerado automaticamente pela cadeia de transformações do MPA até o EMSO.

4.3.4 Cenário 4: Código-fonte EMSO - PIM M4PIA - Código-fonte MPA

O cenário 4, inicia os ciclos de transformações pelo código-fonte MPA, realiza a transformação T2M para o modelo PSM EMSO até o modelo PIM M4PIA, similar ao cenário 2 exposto na seção 4.3.2. A figura 4.10 mostra, de exemplo, a classe sistema de compressão do modelo PIM M4PIA gerada automaticamente pelas transformações T2M e M2M.

Similar ao cenário 3 da seção 4.3.3, os métodos das classes que contém o `Código`

```
136 Model compression_system
137
138 ATTRIBUTES
139   Pallete = false;
140   Info  = "Compression system equipment";
141
142 VARIABLES
143   power as Real;
144   surge_tank as surge_tank;
145   list_compressors as compressor;
146   output_header as ouput_tank;
147
148 EQUATIONS
149   #"Calculate power of compression system"
150 end
```

Trecho de Código 4.5: Trecho de código da classe Compressor gerado automaticamente pelo cenário 3.

Tabela 4.3: Cenário 3: Transformação MPA para EMSO: código-fonte "Manual"vs. Gerado automaticamente.

	EMSO "Manual"	EMSO Gerado
Núm. de Modelos	11	11
Núm. de Equações	2	2
Núm. de Variáveis	22	36
Núm. de Parâmetros	16	0
Lines of Code (LOC)	159	140

são transformados para modelo PIM com a linguagem da plataforma fonte. Neste cenário, a linguagem selecionada é a linguagem EMSO. Ao se executar a transformação M2M, do modelo PIM para o modelo PSM MPA, o `Código` setado no método será perdido na transformação devido a regra criada nesta etapa da transformação.

Ao realizar a transformação M2M do PIM, proveniente de um modelo EMSO, para o PSM MPA, todos os atributos dos tipos variáveis ou do tipo parâmetros no modelo EMSO serão transformados em atributos `Variáveis` no modelo PSM MPA.

Como destacado no cenário 3, em ambos códigos-fonte modelados nas plataformas MPA e EMSO, existem dois atributos do tipo lista, um na classe compressor e outro na classe sistema de compressão. Para modelar uma lista na plataforma MPA, é necessário colocar junto ao `Type` do atributo um conjunto de chaves vazias, neste exemplo, o atributo "`list_compressors`"tem o seu `Type` setado como "compressors[]". Para a plataforma EMSO, a maneira de setar um atributo como lista, por meio de uma boa pratica de programação, configura-se setando no `ID` do atributo o seu `ID` acompanhado um parêntese, com o `ID` de outro parâmetro do tipo `Inteiro`. Nesta modelagem, o atributo da classe sistema de compressão chamado "list_compressors(n_compression_-

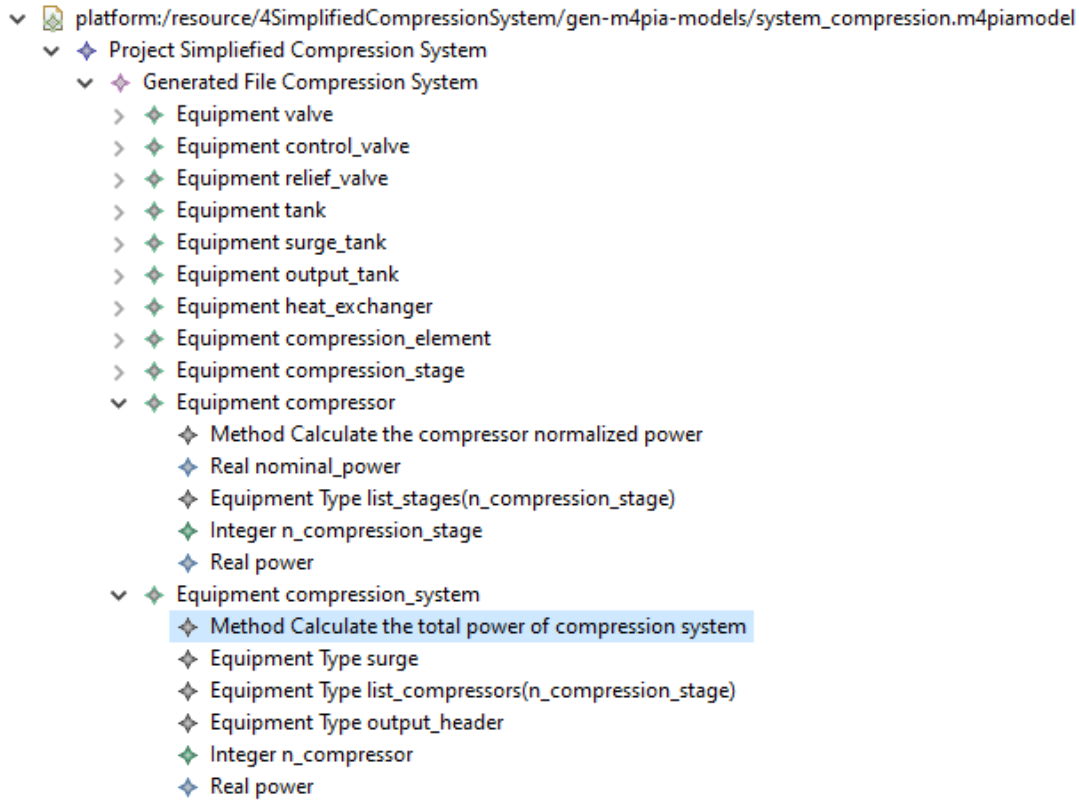

Figura 4.10: Classe sistema de compressão no modelo PIM obtida pela transformação M2M do PSM EMSO.

stage)"é uma lista, onde o parâmetro "n_compression_stage"determina o tamanho da lista. A figura 4.11 mostra essa modelagem no modelo PSM MPA. Isso significa que ao transformar a modelagem feita na plataforma EMSO para a plataforma MPA, estes dois atributos da classe a mais que surgirão por meio da transformação serão entendidos como propriedades da classe. Em um sistema real, isso pode acarretar problemas de modelagem e custo maior, onde existiram pontos a serem controlados sem, de fato, terem utilidade.

Realizando a transformação M2T do modelo PSM MPA para o código-fonte MPA obtém-se o fim do ciclo de transformações do cenário 4. O código gerado automaticamente contém todas as classes de equipamentos existentes no código EMSO que iniciou o ciclo de transformações e, também, todos as variáveis e parâmetros. Entretanto, a informação `Grupo` da classe, o `Acesso` do atributo, e o `Código` do método perderam-se no código gerado automaticamente. O trecho de código 4.6 ilustra a classe sistema de compressão do código MPA gerado automaticamente pelo cenário 4. A tabela 4.4 apresenta uma comparação quantitativa do código MPA modelado manualmente, com o código gerado automaticamente para a mesma planta.

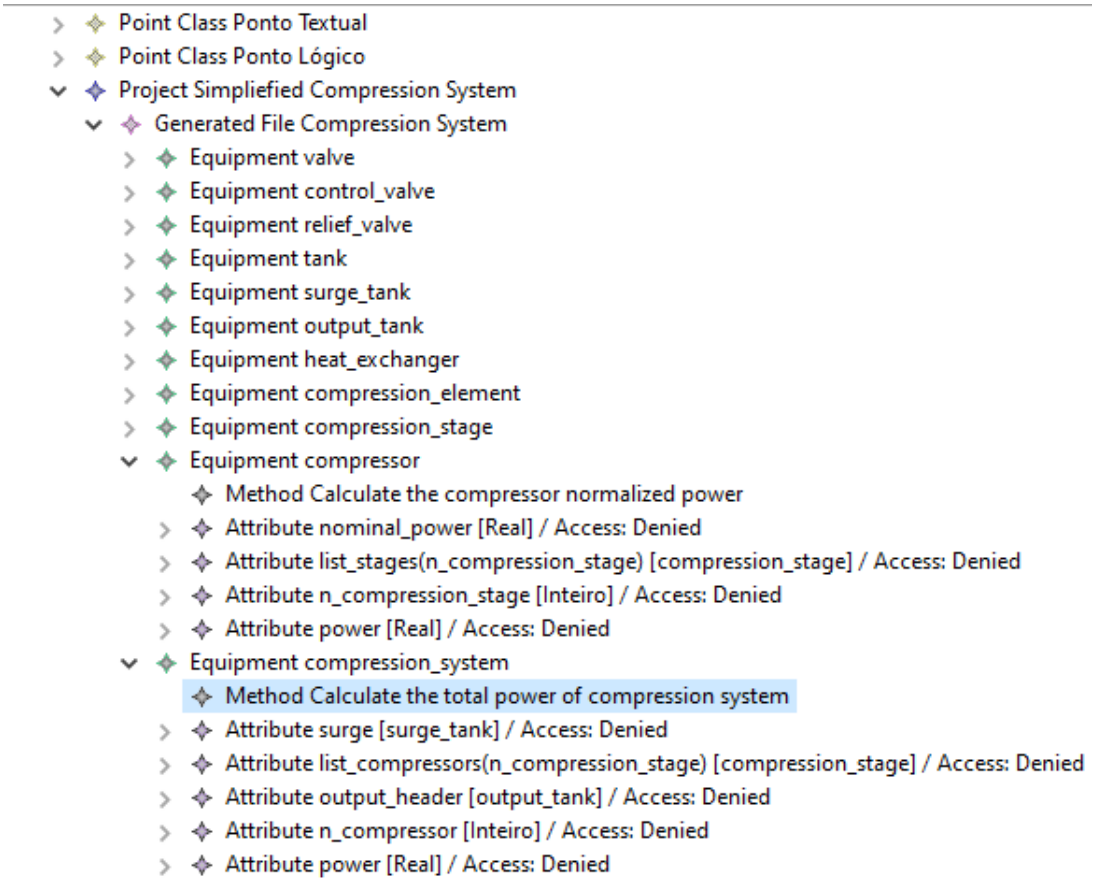

Figura 4.11: Classe sistema de compressão no modelo PSM MPA obtida pela transformação M2M do PIM M4PIA.

Tabela 4.4: Cenário 4: Transformação EMSO para MPA: código-fonte "Manual"vs. Gerado automaticamente.

	MPA "Manual"	MPA Gerado
Núm. de Equipamentos	11	11
Núm. de Métodos	2	2
Núm. de Atributos	36	38
Núm. de Variáveis	2	0
Lines of Code (LOC)	288	235

4.4 DISCUSSÕES

A prova conceito realizada por meio da planta petroquímica do sistema de compressão simplificado pretendeu analisar a aplicabilidade da infraestrutura M4PIA no domínio de definição de classes de equipamentos para aplicações de controle, operação e simulação na indústria petroquímica, visando, mais ainda, a eficácia da engenharia reversa dos códigos-fontes das plataformas suportadas pela infraestrutura através das transformações que geram os códigos de baixo nível até o mais alto nível de abstração

```
204 class{  id = "compression_system", name = "compression_system", group = "",
205   bases = {},
206   description = [[Compression system equipment]],
207   attributes = {
208     { id = "surge", name = "surge", type = "surge_tank", access = "",
209       description = [[]],
210     },
211     { id = "list_compressors(n_compression_stage)", name = "list_compressors(
          n_compression_stage)", type = "compression_stage", access = "",
212       description = [[]],
213     },
214     { id = "output_header", name = "output_header", type = "output_tank", access
          = "",
215       description = [[]],
216     },
217     { id = "n_compressor", name = "n_compressor", type = "INTEGER", access = "",
218       description = [[]],
219     },
220     { id = "power", name = "power", type = "REAL", access = "",
221       description = [[]],
222     }
223   },
224   code = [[ ]],
225   methods = {
226     { id = "", name = "Calculate the total power of compression system",
227       description = [[]],
228       parameters = {},
229       results = {},
230       code = [===[ function()
231         invalid
232       end ]===]
233     }
234   }
235 }
```

Trecho de Código 4.6: Trecho de código da classe sistema de compressão gerado automaticamente pelo cenário 4.

da infraestrutura. Possibilitando auxiliar na reengenharia de sistemas legados das plataformas MPA e EMSO.

O sistema de compressão usado para aplicação da prova conceito neste trabalho pode ser considerado um sistema pequeno em comparação as aplicações reais da indústria petroquímica. É plausível que, em um sistema real da industrial petroquímico existam, aproximadamente, centenas de equipamentos e métodos com milhares de atributos e métodos.

O MPA, como destacado na seção 2.5 é uma plataforma de controle, operação e automação de processos industriais, por outro lado, o EMSO, apresentado na seção 2.6 é uma plataforma de simulação, otimização e controle genérico de processos de plantas industriais. É esperado que as transformações não tivessem a completa cobertura das transformações devido as suas particularidades, todavia, a cadeia de transformações permite o refinamento da modelagem em todas as partes do processo viabilizando que as transformações ocorram com maior precisão e com um menor esforço de conhecimento das especificidades das plataformas de desenvolvimento.

O modelo PIM é a etapa de maior abstração de toda a cadeia de transformações, isso significa o suporte tanto da plataforma MPA, como a plataforma EMSO. É desejável que a infraestrutura M4PIA seja o ponto de partida da modelagem de um sistema da indústria petroquímica. Por tanto, a engenharia reversa, aspirando trazer os códigos-fonte legados dos sistemas, permite que a modelagem e o refinamento de um sistema ocorra com maior facilidade, reduzindo o custo de desenvolvimento e reaproveitando as modelagem legada.

Todas as classes, métodos e atributos contidos no código-fonte MPA e EMSO do sistema de compressão simplificado obtiveram sucesso nas transformações T2M e M2M até o modelo PIM. As classes e atributos que eram especializadas de outros equipamentos foram transformados tendo suas classes de maior hierarquia mantidas. Os métodos e atributos de cada sistema apresentaram perdas de informações importantes para a melhor caracterização da modelagem do sistema.

A engenharia reversa aplicada neste trabalho ainda conta com muitas limitações ligadamente com a infraestrutura M4PIA. A transformação T2M tem por gargalo o modelo PSM de cada plataforma específica. Isso significa que a abstração dos modelos especializados não suportam todas as aplicabilidades dos códigos-fonte das plataformas. Um exemplo que é utilizado constantemente nos sistemas é a importação de bibliotecas no código-fonte. Essa especifidade não está contemplada na transformação T2M.

O levantamento de métricas precisas e coleta de dados detalhados de um processo de engenharia reversa baseado em modelos podem demandar um longo tempo de acompanhamento da implantação e uso das infraestruturas, o que torna essa análise mais apropriada para ferramentas consolidadas (MOHAGHEGHI; DEHLEN, 2009). A utilização da proposta apresentada neste trabalho ainda está na sua fase inicial. Poucos projetistas tiveram contato com a infraestrutura M4PIA, sendo todos esses parte da equipe do projeto no qual esta ferramenta é desenvolvida. Dessa maneira, ainda não é possível obter dados significativos para realizar uma análise completa dos ganhos de produtividade e demais benefícios almejados com o uso do desenvolvimento baseado em modelos e geração automática dos códigos-fontes para o modelo M4PIA de alto nível de abstração e geração automática de códigos com fim de validar a solução proposta.

Mesmo não conseguindo analisar com métricas precisas a infraestrutura, a análise realizada deste trabalho, por meio de todos os possíveis cenários de transformações da infraestrutura, apresenta que o suporte da engenharia reversa na infraestrutura M4PIA tem potencial para auxiliar no desenvolvimento de projetos de plantas petroquímicas e, mais ainda, atuar com a reengenharia de sistemas legados, muitas vezes, difíceis de trabalhar.

4.5 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentada uma prova de conceito através um sistema simplificado de compressão de gás que permitiu ilustrar o desenvolvimento de uma aplicação utilizando a infraestrutura M4PIA e como se dá a engenharia reversa implementada em companhia da geração automática de códigos-fonte. Através de um sistema de compressão simplificado foram modelados dois códigos-fontes: um para a plataforma MPA e outro para a plataforma EMSO. Por meio destas modelagens, foi realizado quatro cenários de avaliação da prova conceito que cobriu todas as possibilidades de transformações da infraestrutura M4PIA, foram realizadas análises empíricas, contendo discussões sobre cada cenários e apresentando as limitações da engenharia reversa e da infraestrutura M4PIA. A solução atua de maneira a auxiliar a reengenharia de sistemas legados, entretanto ainda precisa evoluir em algumas propriedades. No próximo capítulo serão apresentadas as conclusões, contribuições, limitações e perspectivas futuras obtidas com o desenvolvimento deste trabalho.

5 CONCLUSÕES

Este trabalho apresenta a proposta de integração de suporte a engenharia reversa baseada em modelos a infraestrutura M4PIA para o desenvolvimento de aplicações de simulação, supervisão, operação e controle de plantas industriais, mais especificamente da indústria petroquímica. A infraestrutura M4PIA (*Model-Driven Engineering for Petrochemical Industry Automation*) contempla o suporte e a automação de projetos e desenvolvimento de classes de equipamentos e processos industriais. Fazem parte da composição da infraestrutura os metamodelos PIM (*Platform Independent Model*) e PSM (*Platform Specific Model*) e as transformações M2M (*Model-to-Model* - Modelo para Modelo) e M2T (*Model-to-Text* - Modelo para Texto) com a capacidade de geração automática de códigos para a implementação das aplicações nas plataformas específicas suportadas a partir da especificação do sistema modelado em alto nível de abstração. Também a compõem a infraestrutura as transformações T2M (*Text-to-Model* - Texto para Modelo) e M2M (*Model-to-Model* - Modelo para Modelo) para a realização da engenharia reversa capaz de migrar códigos de sistemas legados e importação de códigos, de/para o mais alto nível de abstração. É possível implementar a geração e a recuperação arquitetural automática de códigos para os códigos de pré-configuração para a plataforma de operação e controle MPA (Módulo de Procedimentos Automatizados) e, igualmente, a geração e recuperação arquitetural automática de códigos de modelos para simulação de processos em linguagem de modelagem EMSO (*Environment for Modeling, Simulation and Optimization*).

Como prova conceito deste trabalho, um sistema de compressão de gás simplificado de uma plataforma de petróleo e gás foi usado para ilustração da aplicabilidade da solução no domínio dentro de quatro cenários possíveis cobrindo todo o espectro de transformações suportados pela infraestrutura, possibilidade de escalabilidade dos ganhos considerando as dimensões do sistema simplificado e, do mesmo modo, a verificação das características e recursos suportadas pela infraestrutura. Das abordagens de desenvolvimento dirigidas por modelos são almejados ganhos na manutenibilidade das aplicações, a interoperabilidade e rastreabilidade entre as distintas plataformas de *software*, e, juntamente, a redução do tempo de desenvolvimento e erros de compatibilidade devido a diferentes modelagens dos mesmos conceitos. Importante ressaltar que devido a usabilidade da infraestrutura ainda está em etapas inciais e não consolidado, foi possível analisar apenas os cenários desenhados para verificar a recuperação arquitetural e geração de códigos automaticamente.

Serão apresentadas, a seguir, as principais contribuições e limitações deste traba-

lho, bem como as questões para a evolução e desenvolvimento de trabalhos futuros.

5.1 CONTRIBUIÇÕES

Como apresentado no capítulo de introdução deste trabalho, a solução almejava definir o processo de desenvolvimento de engenharia reversa para ser suportada pela infraestrutura, especificar as transformações entre os modelos de nível específico para o de nível independente de plataforma, definir as transformações entre os códigos-fontes para os modelos de nível específico, analisar o desempenho das transformações por meio de uma prova conceito e discussões sobre os cenários desenvolvidos para testes.

O trabalho tem como elementos principais de contribuição científica a infraestrutura MDRE que possibilita a reengenharia de sistemas legados para as plataformas contempladas pela infraestrutura M4PIA tanto a plataforma MPA, quanto a EMSO. Além disso, o desenvolvimento das transformações T2M e M2M para ambas as plataformas MPA e EMSO permitindo recuperar a arquitetura de sistemas de mais baixo nível até o mais alto nível de abstração independente de plataforma. Também é uma contribuição deste trabalho a apresentação de uma prova conceito com quatro distintos cenários desenvolvidos para observar todas as possíveis transformações permitidas pela infraestrutura M4PIA e uma análise empírica dos resultados obtidos pelos cenários.

5.2 LIMITAÇÕES

As limitações presentes neste trabalho também merecem destaque no que se refere à necessidade de melhoria no desenvolvimento das aplicações e das avaliações dos benefícios e problemas decorrentes do uso da infraestrutura na aplicação do domínio.

A análise carece de dados que permitam uma análise não subjetiva da ferramenta e que indique os prós obtidos com a utilização da infraestrutura proposta, destaque para a análise de produtividade no desenvolvimento das aplicações esperada como resultado da engenharia reversa de geração automática de um modelo independente de plataforma a partir dos códigos-fontes legados dos sistemas, reduzindo os erros sintáticos comuns na codificação, avanços na interoperabilidade e reutilização entre as plataformas.

Outra limitação é falta de uma avaliação do impacto ocorrido na metamodelo PIM com o acréscimo de uma nova plataforma de *software* e suas respectivas transformações. Vale destacar a ausência de um corretor de códigos dos métodos, funções e demais perdas ocorrido entre as transformados automaticamente entre as plataformas.

5.3 PUBLICAÇÃO

Este trabalho gerou a publicação de um artigo científico desenvolvido por Cruz, Damo and Becker (2020). A artigo foi intitulado "*Round-Trip Engineering for Petrochemical Industry Automation*"publicado no "*21th IFAC World Congress*"marcado para ocorrer em Berlim, Alemanha. Entretanto, devido a pandemia causada pela COVID-19, o congresso aconteceu em caráter virtual com a apresentação via vídeo gravado, disponíveis entre os dias 11 e 17 de Julho de 2020.

5.4 TRABALHOS FUTUROS

Os pontos destacados para trabalhos futuros são uma análise da possibilidade de implementação de linguagens gráficas de modelagem para facilitar mais ainda a definição de classes de equipamentos, reengenharia de sistemas legados e permitir uma visualização melhor dos sistemas modelados e as relações entre seus elementos.

Outro trabalho futuro pertinente é a definição de um método para extensão dos artefatos que já compõem a infraestrutura, como, novas plataformas de *software* de supervisão, operação e controle de plantas industriais, inclusão de novos elementos do domínio das aplicações, e ajustes em todas as transformações possíveis contempladas na infraestrutura para garantir a correta propagação de informações.

Uma característica interessante para tornar a infraestrutura mais robusta é a adição de novas plataformas de *software* suportadas, com plataformas comerciais populares no mercado para indústria do petróleo e gás, e uma avaliação de seus impactos.

Análise da viabilidade de um desenvolvimento que tenha a capacidade de fazer uma recuperação arquitetural na implementação das operações pela análise semântica do código .

A aplicação de uma análise com experimentos formais e estudo de caso com desenvolvedores de aplicações para comparação entre o desenvolvimento usando o paradigma MDE para geração automática de código e MDRE para reengenharia de sistemas legados e o método de programação manual tradicional e coleta e análise de dados. Simultaneamente a isso, o levantamento de métricas bem definidas para análise da dados quantitativos para validação da proposta.

REFERÊNCIAS

AMARAL, V.; MERNIK, M. Special issue on quality in model-driven engineering. *Software Quality Journal*, Springer, v. 24, n. 3, p. 597–599, 2016.

ANGYAL, L.; LENGYEL, L.; CHARAF, H. A synchronizing technique for syntactic model-code round-trip engineering. In: IEEE. *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*. [S.l.], 2008. p. 463–472.

BARBIER, G. et al. Modisco, a model-driven platform to support real legacy modernization use cases. In: *Information Systems Transformation*. [S.l.]: Elsevier, 2010. p. 365–400.

BECKER, L.; JR, W. P.; PEREIRA, C. Proposal of an integrated object-oriented environment for the design of supervisory software for real-time industrial automation systems. In: *Fourth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'99)*. [S.l.: s.n.], 1999. p. 249–249.

BECKER, L. B. et al. Projeto de sistemas distribuídos e de tempo real para automação. EDUFBA, p. 193–224, 2018.

BRAHIM, B.; TAOUFIQ, G. et al. Comparative study between clustering and model driven reverse engineering approaches. *Lecture Notes on Software Engineering*, IACSIT Press, v. 1, n. 2, p. 135, 2013.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, Morgan & Claypool Publishers, v. 3, n. 1, p. 1–207, 2017.

BRUNELIERE, H. et al. Modisco: a generic and extensible framework for model driven reverse engineering. In: ACM. *Proceedings of the IEEE/ACM international conference on Automated software engineering*. [S.l.], 2010. p. 173–174.

BRUNELIèRE, H. et al. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, v. 56, n. 8, p. 1012–1032, 2014. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584914000883>.

CAMPOS, M. et al. Controle avançado e otimização na indústria do petróleo. *Rio de Janeiro: Editora Interciência*, 2013.

COSENTINO, V. et al. A model driven reverse engineering framework for extracting business rules out of a java application. In: BIKAKIS, A.; GIURCA, A. (Ed.). *Rules on the Web: Research and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 17–31. ISBN 978-3-642-32689-9.

CRUZ, M. V. S.; DAMO, T. P.; BECKER, L. B. Round-trip engineering for petrochemical industry automation. *IFAC-PapersOnLine*, v. 53, n. 2, p. 11824–11829, 2020. ISSN 2405-8963. 21th IFAC World Congress. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405896320310065>.

DAMO, T. P. *Engenharia Baseada em Modelos para a Aplicação de Simulação, Controle e Operação de Plantas na Indústria Petroquímica*. [S.l.]: Universidade Federal de Santa Catarina, 2019.

FAVRE, J.-M. Foundations of model (driven)(reverse) engineering: models–episode i: stories of the fidus papyrus and of the solarus. In: SCHLOSS DAGSTUHL-LEIBNIZ-ZENTRUM FÜR INFORMATIK. *Dagstuhl Seminar Proceedings*. [S.l.], 2005.

FAVRE, L.; MARTINEZ, L.; PEREIRA, C. Mda-based reverse engineering of object oriented code. In: *Enterprise, business-process and information systems modeling*. [S.l.]: Springer, 2009. p. 251–263.

FAVRE, L.; MARTINEZ, L.; PEREIRA, C. Reverse engineering of object-oriented code: An adm approach. In: *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications*. [S.l.]: IGI Global, 2018. p. 1479–1502.

FENTON, N.; BIEMAN, J. *Software metrics: a rigorous and practical approach*. [S.l.]: CRC press, 2019.

FINKLER, T. F.; SOARES, R. P. *EMSO-OPC Link - User Manual*. [S.l.: s.n.], 2007.

FLEUREY, F. et al. Model-driven engineering for software migration in a large industrial context. In: SPRINGER. *International Conference on Model Driven Engineering Languages and Systems*. [S.l.], 2007. p. 482–497.

GUISASOLA, T.; MAIA, R. *MPA um Sistema de Controle de Plantas Industriais*. 2009. Disponível em: <https://www.lua.org/wshop09/mpa.pdf>. Acesso em: 12 nov. 2020.

INSTITUTO TECGRAF/PUC-RIO. *Instituto Tecgraf/PUC-Rio*. 2020. Disponível em: <https://www.tecgraf.puc-rio.br/>. Acesso em: 09 nov. 2020.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 9506: Industrial automation systems – Manufacturing Message Specification.* [S.l.: s.n.], 2003. 2. ed.

IZQUIERDO, J. L. C.; MOLINA, J. G. Extracting models from source code in software modernization. *Software & Systems Modeling*, Springer, v. 13, n. 2, p. 713–734, 2014.

KENT, S. Model driven engineering. In: SPRINGER. *International Conference on Integrated Formal Methods*. [S.l.], 2002. p. 286–298.

KLATT, K.-U.; MARQUARDT, W. Perspectives for process systems engineering—personal views from academia and industry. *Computers & Chemical Engineering*, Elsevier, v. 33, n. 3, p. 536–550, 2009.

KURTEV, I. *Adaptability of model transformations.* [S.l.]: Citeseer, 2005.

MARAH, H. M.; CHALLENGER, M.; KARDAS, G. Re4tinyos: A reverse engineering methodology for the mde of tinyos applications. In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. [S.l.: s.n.], 2020. p. 741–750.

MASCARENHAS, F. *Alien - Pure Lua extensions*. 2020. Disponível em: <http://mascarenhas.github.io/alien/>. Acesso em: 12 nov. 2020.

MOHAGHEGHI, P.; DEHLEN, V. Existing model metrics and relations to model quality. In: IEEE. *2009 ICSE Workshop on Software Quality*. [S.l.], 2009. p. 39–45.

NIRUMAND, A.; ZAMANI, B.; LADANI, B. T. Vandroid: A framework for vulnerability analysis of android applications using a model-driven reverse engineering technique. *Software: Practice and Experience*, Wiley Online Library, v. 49, n. 1, p. 70–99, 2019.

OBJECT MANAGEMENT GROUP. *Model Driven Architecture (MDA)*. [S.l.: s.n.], 2001. OMG Document: ormsc/2001-07-01.

OBJECT MANAGEMENT GROUP. *MOF Model to Text Transformation Language, v1.0*. [S.l.: s.n.], 2008. OMG Document: formal/2008-01-16.

OBJECT MANAGEMENT GROUP. *Object Constraint Language*. [S.l.: s.n.], 2014. OMG Document: formal/2014-02-03.

OBJECT MANAGEMENT GROUP. *XML Metadata Interchange (XMI) Specification, Version 2.5.1*. [S.l.: s.n.], 2015. OMG Document: formal/2015-06-07.

OBJECT MANAGEMENT GROUP. *MOF Query/View/Transformation*. [S.l.: s.n.], 2016a. OMG Document: formal/2016-06-03.

OBJECT MANAGEMENT GROUP. *OMG Meta Object Facility (MOF) Core Specifcation*. [S.l.: s.n.], 2016b. OMG Document: formal/2019-10-01.

PARR, T. *The definitive ANTLR 4 reference*. [S.l.]: Pragmatic Bookshelf, 2013.

PEREZ-CASTILLO, R. Marble: Modernization approach for recovering business processes from legacy information systems. In: IEEE. *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. [S.l.], 2012. p. 671–676.

PÉREZ-CASTILLO, R. et al. Marble. a business process archeology tool. In: IEEE. *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. [S.l.], 2011. p. 578–581.

PÉREZ-CASTILLO, R.; GUZMÁN, I. G.-R. de; PIATTINI, M. Business process archeology using marble. *Information and Software Technology*, Elsevier, v. 53, n. 10, p. 1023–1044, 2011.

RAIBULET, C.; FONTANA, F. A.; ZANONI, M. Model-driven reverse engineering approaches: A systematic literature review. *IEEE Access*, IEEE, v. 5, p. 14516–14542, 2017.

RAMÓN, Ó. S.; CUADRADO, J. S.; MOLINA, J. G. Model-driven reverse engineering of legacy graphical user interfaces. *Automated Software Engineering*, Springer, v. 21, n. 2, p. 147–186, 2014.

RAMÓN, Ó. S.; VANDERDONCKT, J.; MOLINA, J. G. Re-engineering graphical user interfaces from their resource files with usiresourcer. In: IEEE. *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*. [S.l.], 2013. p. 1–12.

RUGABER, S.; STIREWALT, K. Model-driven reverse engineering. *IEEE software*, IEEE, v. 21, n. 4, p. 45–53, 2004.

SABIR, U. et al. A model driven reverse engineering framework for generating high level uml models from java source code. *IEEE Access*, IEEE, v. 7, p. 158931–158950, 2019.

SATUF, E.; PINTO, S. e quaresma, b.,(2009). *Sistema automático de alinhamento para a Plataforma de rebombeio autônomo PRA-1*, 2009.

SCHMIDT, D. C. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, Citeseer, v. 39, n. 2, p. 25, 2006.

SEBORG, D. E. et al. *Process dynamics and control*. [S.l.]: John Wiley & Sons, 2010.

SELIC, B. The pragmatics of model-driven development. *IEEE software*, IEEE, v. 20, n. 5, p. 19–25, 2003.

SOARES, R. d. P. Desenvolvimento de um simulador genérico de processos dinâmicos. Universidade Federal do Rio Grande do Sul, 2003.

SOARES, R. d. P.; SECCHI, A. Emso: A new environment for modelling, simulation and optimisation. In: *Computer Aided Chemical Engineering*. [S.l.]: Elsevier, 2003. v. 14, p. 947–952.

SOLEY, R. et al. Model driven architecture. *OMG white paper*, v. 308, n. 308, p. 5, 2000.

SOMMERVILLE, I. Software engineering 9th edition. *ISBN-10*, v. 137035152, 2011.

STEINBERG, D. et al. *EMF: eclipse modeling framework*. [S.l.]: Pearson Education, 2008.

THE MATHWORKS INCORPORATED. *MATLAB - MathWorks*. 2021. Disponível em: <https://www.mathworks.com/products/matlab.html>. Acesso em: 07 jun. 2021.

UML, O.; MOF, I. *The Unified Modeling Language UML*. [S.l.]: ed, 2011.

VÖLTER, M. et al. *Model-driven software development: technology, engineering, management*. [S.l.]: John Wiley & Sons, 2013.

YANG, Z. et al. C2aadl_reverse: A model-driven reverse engineering approach for development and verification of safety-critical software. *Journal of Systems Architecture*, Elsevier, p. 102202, 2021.

YUSUF, L.; CHESSEL, M.; GARDNER, T. Implement model-driven development to increase the business value of your it system. *Retrieved January*, v. 29, p. 2008, 2006.

```
2 ]=]
3
4  class{ id = "valve", name = "Valve", group = "Valves",
5   isPoint = false,
6   bases = {},
7   description = [[Basic Valve]],
8   attributes = {
9      {  id = "phi", name = "Opening", type = "REAL",
10       access = "gsrw",
11       description = [[Real which informs valve of opening]],
12     },
13     { id = "flow",  name = "Flow",  type = "REAL",
14       access = "gsr",
15       description = [[Real which informs the control of valve massic flow]],
16     },
17     { id = "temperature", name = "Teperature",  type = "REAL",
18       access = "gsr",
19       description = [[Kelvin unit]],
20     },
21     { id = "upstream_pressure", name = "Upstream Pressure", type = "REAL",
22       access = "gsr",
23       description = [[Real which informs the control's valve pressure]],
24     },
25   },
26   codes = [[]],
27   methods = {}
28 }
29
30 class{ id = "control_valve", name = "Control valve", group = "Valves",
31   bases = {"valve"},
32   description = [[Control valve, extends basic valve]],
33   attributes = {
34      {  id = "downstream_pressure", name = "Downstream Pressure", type = "REAL",
35       access = "gsw",
36       description = [[Real which informs the control of valve massic flow]],
37     },
38     { id = "kc",  name = "KC",  type = "REAL",
39       access = "gs",
40       description = [[Valve's coefficient - gain]],
41     },
42   },
43   codes = [[]],
44   methods = {}
45 }
46
47 class{ id = "relief_valve", name = "Relief valve", group = "Valves",
48   bases = {"valve"},
49   despription = [[Pressure relief valve, extends basic valve]],
50   attributes = {
51      {  id = "maximum_pressure", name = "Maximum Pressure", type = "REAL",
52       access = "gs",
53       description = [[Maximum pressure to relief]],
54     },
```

/

```
55   },
56   codes = [[]],
57   methods = {}
58 }
59
60 class{ id = "tank", name = "Tank", group = "Compression System",
61   bases = {},
62   description = [[Basic tank]],
63   attributes = {
64      {  id = "pressure", name = "Suction Pressure", type = "REAL",
65       access = "gsw",
66       description = [[Bar unit]],
67     },
68     { id = "temperature", name = "Suction Temperature", type = "REAL",
69       access = "grw",
70       description = [[Kelvin unit]],
71     },
72   },
73   codes = [[]],
74   methods = {}
75 }
76
77 class{ id = "surge_tank", name = "Surge tank", group = "Compression System",
78   bases = {"tank"},
79   description = [[Tank to avoid surge with a flare valve]],
80   attributes = {
81      {  id = "input_flow", name = "Input flow", type = "REAL",
82       access = "gsr",
83       description = [[Real which informs the surge of tank input massic flow]],
84     },
85     { id = "output_flow", name = "Output flow", type = "REAL",
86       access = "gsr",
87       description = [[Real which informs the surge of tank output massic flow]],
88     },
89     { id = "flare_valve", name = "Flare valve", type = "relief_valve",
90       access = "g",
91       description = [[Surge tank flare valve]],
92     },
93   },
94   codes = [[]],
95   methods = {}
96 }
97
98
99 class{ id = "ouput_tank", name = "Output Header", group = "Compression System",
100   bases = {"tank"},
101   description = [[Output tank with valves to exportation line and gas liftt
        output]],
102   attributes = {
103      {  id = "input_flow", name = "Input flow", type = "REAL",
104       access = "gsr",
105       description = [[Real point which informs the input massic flow of the
            output tank]],
```

```
106     },
107     { id = "export_valve", name = "Export valve", type = "control_valve",
108       access = "g",
109       description = [[Exportation line valve]],
110     },
111     { id = "list_vs_output", name = "Output valve", type = "control_valve[]",
112       access = "g",
113       description = [[List of output valves]],
114     },
115   },
116   codes = [[]],
117   methods = {}
118 }
119
120 class{ id = "heat_exchanger", name = "Heat Exchanger", group = "Compression
      System",
121   bases = {},
122   description = [[Basic heat exchanger]],
123   attributes = {
124      {  id = "input_pressure", name = "Input pressure", type = "REAL",
125       access = "grw",
126       description = [[Bar unit]],
127     },
128     { id = "output_pressure", name = "Output pressure", type = "REAL",
129       access = "grw",
130       description = [[Bar unit]],
131     },
132     { id = "input_temperature", name = "Input temperature", type = "REAL",
133       access = "grw",
134       description = [[Kelvin unit]],
135     },
136     { id = "output_temperature", name = "Output temperature", type = "REAL",
137       access = "grw",
138       description = [[Kelvin unit]],
139     },
140   },
141   codes = [[]],
142   methods = {}
143 }
144
145 class{ id = "compression_element", name = "Compression element", group = "
      Compressor",
146   bases = {},
147   description = [[Compressor element]],
148   attributes = {
149      {  id = "flow", name = "Compressor flow", type = "REAL",
150       access = "gsr",
151       description = [[m^3/s unit]],
152     },
153     { id = "suction_pressure", name = "Suction pressure", type = "REAL",
154       access = "grs",
155       description = [[Bar unit]],
156     },
```

```
157     { id = "suction_temperature", name = "Suction temperature", type = "REAL",
158       access = "gsr",
159       description = [[Kelvin unit]],
160     },
161     { id = "discharge_pressure", name = "Discharge pressure", type = "REAL",
162       access = "grs",
163       description = [[Bar unit]],
164     },
165     { id = "discharge_temperature", name = "Discharge temperature", type = "REAL"
            ,
166       access = "grw",
167       description = [[Kelvin unit]],
168     },
169   },
170   codes = [[]],
171   methods = {}
172 }
173
174 class{ id = "compression_stage", name = "Compression stage", group = "Compressor"
       ,
175   bases = {},
176   description = [[Equipments of compression system stage]],
177   attributes = {
178      {  id = "power", name = "Power", type = "REAL",
179       access = "grw",
180       description = [[Stage power W]],
181     },
182     { id = "exchanger", name = "Heat exchanger", type = "heat_exchanger",
183       access = "g",
184       description = [[Heat exchanger of this stage]],
185     },
186     { id = "suction_tank", name = "Suction tank", type = "tank",
187       access = "g",
188       description = [[Suction tank of this stage]],
189     },
190     { id = "compression", name = "Compression element", type = "
            compression_element",
191       access = "g",
192       description = [[Compression element of this stage]],
193     },
194     { id = "recycle_valve", name = "Recycle valve", type = "control_valve",
195       access = "g",
196       description = [[Recycle valve of this stage]],
197     },
198   },
199   codes = [[]],
200   methods = {}
201 }
202
203 class{ id = "compressor", name = "Compressor", group = "Compressor",
204   bases = {},
205   description = [[Compressor equipment and its stages]],
206   attributes = {
207      {  id = "power", name = "Power", type = "REAL",
```

```
208       access = "grw",
209       description = [[Compressor total power W]],
210     },
211     { id = "nominal_power", name = "Nominal power" type = "REAL",
212       access = "g",
213       description = [[Compressor nominal power W]],
214     },
215     { id = "list_stages", name = "List of the compression stages" type = "
            compression_stage[]",
216       access = "g",
217       description = [[List of compression stages]],
218     },
219   },
220   codes = [[]],
221   methods = {
222     { id = "calculate_power", name = "Calculate the compressor normalized power",
223       description = [[Summing the power of all stages]],
224       parameters = {},
225       results = {
226         { name = "Normalized power of the compressor W", type = "REAL" },
227       },
228       code = [===[
229             function()
230               local power_compressor = 0
231               local power_nominal = self.nominal_power
232               for r2, ref2 in ipairs(self.list_stages) do
233                 power_compressor + ref2.power:read()
234               end
235               power_compressor = power_compressor/power_nominal
236               self.power:write(power_compressor)
237               return power_compressor
238             end
239           ]===],
240     },
241   },
242 }
243
244 class{ id = "compression_system", name = "Compression system", group = "
       Compression System",
245   bases = {},
246   description = [[Compression system equipment]],
247   attributes = {
248      {  id = "power", name = "Power", type = "REAL",
249       access = "rw",
250       description = [[Power of compression system W]],
251     },
252     { id = "surge_tank", name = "Surge Tank" type = "surge_tank",
253       access = "g",
254       description = [[Surge tank associated with compression system]],
255     },
256     { id = "list_compressors", name = "List of the compressors" type = "
            compressor[]",
257       access = "g",
258       description = [[List of compression stages]],
```

```
259     },
260     { id = "output_header", name = "Output header" type = "ouput_tank",
261       access = "g",
262       description = [[Output header of compression system]],
263     },
264   },
265   codes = [[]],
266   methods = {
267     { id = "calculate_power", name = "Calculate power of compression system",
268       description = [[Calculate the total power of compression system]],
269       parameters = {},
270       results = {
271         { name = "Power W", type = "REAL" },
272       },
273       code = [===[
274             function()
275               local power_cs = 0
276               local n = 0
277               for r2, ref2 in ipairs(self.list_compressors) do
278                 power_cs + ref2.power:read()
279                 n = n + 1
280               end
281               power_cs = power_cs/n
282               self.power:write(power_cs)
283               return power_cs
284             end
285           ]===],
286     },
287   },
288 }
```

B SISTEMA DE COMPRESSÃO DE GÁS PARA EMSO

```
 2
 3   ATTRIBUTES
 4     Pallete = false;
 5     Info = "Basic valve";
 6
 7   VARIABLES
 8     phi as Real (Brief= "Real which informs valve opening");
 9     flow as Real (Brief= "Real which informs the control valve massic flow");
10     temperature as Real (Brief= "Kelvin unit");
11     upstream_pressure as Real (Brief= "Real which informs the controls valve
           pressure");
12 end
13
14 Model control_valve as valve
15
16   ATTRIBUTES
17     Pallete = false;
18     Info = "Control valve, extends basic valve";
19
20   PARAMETERS
21     kc as Real (Brief= "Valves coefficient - gain");
22
23   VARIABLES
24     downstream_pressure as Real (Brief= "Real which informs the controls valve
           massic flow");
25 end
26
27 Model relief_valve as  valve
28
29   ATTRIBUTES
30     Pallete = false;
31     Info = "Pressure relief valve, extends basic valve";
32   PARAMETERS
33     maximum_pressure as Real (Brief= "Maximum pressure to relief");
34 end
35
36 Model tank
37
38   ATTRIBUTES
39     Pallete = false;
40     Info = "Basic tank";
41
42   VARIABLES
43     pressure as Real (Brief= "Bar unit");
44     temperature as Real (Brief= "Kelvin unit");
45 end
46
47 Model surge_tank as tank
48
49   ATTRIBUTES
50     Pallete = false;
```

```
51     Info = "Tank to avoid surge with a flare valve";
52
53   PARAMETERS
54     flare_valve as relief_valve (Brief= "Surge tank flare valve");
55
56   VARIABLES
57     input_flow as Real (Brief= "Real which informs the surges tank input massic
           flow");
58     output_flow as Real (Brief= "Real which informs the surges tank output massic
           flow");
59 end
60
61 Model output_tank as tank
62
63   ATTRIBUTES
64     Pallete = false;
65     Info = "Output tank with valves to exportation line and gas-liftt output";
66
67   PARAMETERS
68     export_valve as control_valve (Brief= "Exportation line valve");
69     n_control_valve as Integer (Brief="Number of elements of control_valve");
70
71   VARIABLES
72     input_flow as Real (Brief= "Real point which informs the input massic flow of
           the output tank");
73
74 end
75
76 Model heat_exchanger
77
78   ATTRIBUTES
79     Pallete = false;
80     Info = "Basic heat exchanger";
81
82   VARIABLES
83     input_pressure as Real (Brief= "Bar unit");
84     output_pressure as Real (Brief= "Bar unit");
85     input_temperature as Real (Brief= "Kelvin unit");
86     output_temperature as Real (Brief= "Kelvin unit");
87 end
88
89 Model compression_element
90
91   ATTRIBUTES
92     Pallete = false;
93     Info = "Compressor element";
94
95   VARIABLES
96     flow as Real (Brief="m^3/s unit");
97     suction_pressure as Real (Brief= "Bar unit");
98     suction_temperature as Real (Brief= "Kelvin unit");
99     discharge_pressure as Real (Brief= "Bar unit");
100    discharge_temperature as Real (Brief= "Kelvin unit");
```

```
101 end
102
103 Model compression_stage
104
105   ATTRIBUTES
106     Pallete = false;
107     Info = "Equipments of compression system stage";
108
109   PARAMETERS
110     exchanger as heat_exchanger (Brief="Heat exchanger of this stage");
111     suction_tank as tank (Brief="Suction tank of this stage");
112     compression as compression_element (Brief= "Compression element of this stage
          ");
113     recycle_valve as control_valve (Brief= "Recycle valve of this stage");
114
115   VARIABLES
116     power as Real (Brief="Stage power W");
117
118 end
119
120 Model compressor
121
122   ATTRIBUTES
123     Pallete = false;
124     Info = "Compressor equipment and its stages";
125
126   PARAMETERS
127     nominal_power as Real (Brief="Compressor total power W");
128     list_stages(n_compression_stage) as compression_stage (Brief="List of
          compression stages");
129     n_compression_stage as Integer (Brief= "Number of elements of
          compression_stage");
130
131   VARIABLES
132     power as Real (Brief="Compressor total power W");
133
134   EQUATIONS
135     # Summing the power of all satages
136     "Calculate the compressor normalized power"
137     power = sum(list_stages.power)/nominal_power;
138 end
139
140 Model compressor_system
```

```
141
142   ATTRIBUTES
143     Pallete = false;
144     Info = "Compression system equipment";
145
146   PARAMETERS
147     surge as surge_tank (Brief="Surge tank associated with compression system");
148     list_compressors(n_compression_stage) as compression_stage (Brief="List of
            the compressors");
149     output_header as output_tank (Brief="Output header of compression system");
150     n_compressor as Integer (Brief= "Number of elements of list_compressors");
151
152   VARIABLES
153     power as Real (Brief="Power of compression system W");
154
155   EQUATIONS
156     # Calculate power of compression system
157     "Calculate the total power of compression system"
158     power = sum(list_compressors.power)/n_compressor;
159 end
```