UNIVERSIDADE FEDERAL DO MARANHÃO
Centro de Ciências Exatas e Tecnologia
Programa de Pós-Graduação em Engenharia Elétrica

Eder Matheus Silveira Felix

UM FRAMEWORK BASEADO EM ENGENHARIA DIRIGIDA POR MODELOS E *WEAVING* DE MODELOS PARA SUPORTAR A ATIVIDADE DE DESENVOLVIMENTO DE *SOFTWARE* E A INTEGRAÇÃO DE APLICAÇÕES EM *SMART GRIDS*

São Luís - MA
2021

Eder Matheus Silveira Felix

UM FRAMEWORK BASEADO EM ENGENHARIA DIRIGIDA POR MODELOS E *WEAVING* DE MODELOS PARA SUPORTAR A ATIVIDADE DE DESENVOLVIMENTO DE *SOFTWARE* E A INTEGRAÇÃO DE APLICAÇÕES EM *SMART GRIDS*

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Maranhão.

Orientador: Prof. Dr. Denivaldo Cicero Pavão Lopes

São Luís - MA

2021

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Felix, Eder Matheus Silveira.
Um framework baseado em Engenharia Dirigida por Modelos e Weaving de modelos para suportar a atividade de desenvolvimento de software e a integração de aplicações em Smart Grids / Eder Matheus Silveira Felix. - 2021.
175 f.

Orientador(a): Denivaldo Cicero Pavão Lopes.
Dissertação (Mestrado) - Programa de Pós-graduação em Engenharia Elétrica/ccet, Universidade Federal do Maranhão, São Luís, MA, 2021.

1. Engenharia dirigida por modelos. 2. Smart Grids. 3. Weaving de modelos. I. Lopes, Denivaldo Cicero Pavão. II. Título.

Eder Matheus Silveira Felix

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Maranhão.

Aprovada em 27 de julho de 2021:

Prof. Dr. Denivaldo Cicero Pavão Lopes
Orientador
Universidade Federal do Maranhão
* * *
Prof. Dr. Francisco José da Silva e Silva
Universidade Federal do Maranhão
* * *
Prof. Dr. Shigeaki Leite Lima
Universidade Federal do Maranhão
* * *
Prof. Dr. Marcos Didonet Del Fabro
Examinador Externo
Universidade Federal do Paraná
* * *
Prof. Dr. Cleonilson Protásio de Souza
Examinador Externo
Universidade Federal da Paraíba
* * *
Prof. Dr. Slimane Hammoudi
Examinador Externo
Grande Ecole d'Ingénieurs Généralistes - ESEO (França)

Dedicatória

À memória do meu avô, Miguel Félix,
do meu primo, Leonardo Silveira,
e da minha avó, Laurinda Silveira.

Agradecimentos

Agradeço a Deus pela possibilidade de estar vivo, pela possibilidade de estar terminando mais esta etapa da minha vida e por todas as outras infinitas possibilidades.

Aos meus pais, Luziana Silveira e Zaqueu Félix, pelo apoio sempre presente, mesmo à distância, e pela confiança depositada em mim.

À minha querida amiga Kamilla Karem, pela amizade de sempre e por ouvir todos os meus desabafos e lamentações ao longo desses anos.

Aos meus irmãos da fé, da Comunidade Batista da Ilha, pelo apoio e pelas orações.

Aos meus amigos e colegas que conheci no LESERC, Lady Débora, Matheus Ferreira, Rayanne Oliveira, Pedro Cutrim, Yann Ferreira e Raimundo Lima por terem feito dos meus dias no laboratório mais divertidos e leves. Agradeço pela amizade de cada um.

À todos os meu amigos e familiares que não citei diretamente aqui, pelo apoio, carinho, amizade e companheirismo.

Aos professores do PPGEE e colegas com quem convivi durante este mestrado. Os aprendizados que eu tive nesses dois anos irei levar para sempre comigo.

Ao Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal do Maranhão pelo suporte e pela possibilidade de realizar esta pesquisa.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo financiamento desta pesquisa.

À Universidade Federal do Maranhão, pela estrutura física e equipamentos fornecidos durante o desenvolvimento deste trabalho.

Por fim, ao meu orientador, Denivaldo Lopes, pela paciência, pelos conselhos, pela sua contribuição direta nesta pesquisa e por ter se colocado sempre à disposição em todos os momentos durante o desenvolvimento deste trabalho.

"[...] a tribulação produz perseverança;
a perseverança, um caráter aprovado;
e o caráter aprovado, esperança."

(Apóstolo Paulo em Romanos 5:3,4.)

Resumo

As *Smart Grids* combinam sensoriamento e instrumentação dos sistemas elétricos de potência com um complexo sistema de comunicação e informação para processamento de dados. A heterogeneidade dos dados nos sistemas de energia elétrica impõe grandes desafios para os desenvolvedores no que diz respeito à interoperabilidade entre as soluções de *software* para *Smart Grids*. Desta forma, métodos da Engenharia de *Software* são necessários para lidar com esta complexidade e para facilitar a interoperabilidade. A Engenharia Dirigida por Modelos (MDE) tem sido proposta na literatura para gerenciar a complexidade de desenvolvimento de *software*. A MDE é uma abordagem que visa definir e gerenciar os artefatos de *software* como modelos de alto nível durante todo o processo de desenvolvimento. Neste trabalho, um *framework* baseado em MDE chamado FMDE4SGRID é proposto para auxiliar a atividade de desenvolvimento de *software* e a integração de aplicações para *Smart Grids*. A técnica de *weaving* de modelos é incorporada ao *framework* proposto para permitir a separação entre o desenvolvimento da lógica de negócio das aplicações e o desenvolvimento do modelo da rede elétrica. O FMDE4SGRID foi implementado no ambiente Eclipse e três aplicações de *Smart Grids* foram desenvolvidas para validar a abordagem proposta. As aplicações implementadas utilizam serviços de *middleware* em uma arquitetura do tipo *Enterprise Service Bus* (ESB) para compartilhar informações da rede elétrica. Os resultados mostram que o FMDE4SGRID auxilia na análise, projeto e codificação das aplicações. O FMDE4SGRID auxilia também na configuração do *middleware* utilizado para a integração das aplicações.

Palavras-chave: Smart Grids, Engenharia Dirigida por Modelos, *weaving* de modelos.

Abstract

Smart Grids combine the sensing and instrumentalization of electrical power systems with a complex communication and information system for data processing. The heterogeneity of data in electric power systems poses big challenges for developers regarding interoperability between software solutions for Smart Grids. Therefore, Software Engineering methods are necessary to deal with this complexity and to facilitate interoperability. Model-Driven Engineering (MDE) has been proposed in the literature to manage the complexity of software development. MDE is an approach that aims to define and manage software artifacts as high-level models throughout the development process. In this work, an MDE-based framework called FMDE4SGRID is proposed to support the software development activity and the integration of applications for Smart Grids. The model weaving technique is used within the proposed framework to allow to allow the separation between the development of the business logic of the applications and the development of the electric network model. FMDE4SGRID was implemented in the Eclipse platform and three Smart Grids applications were developed to validate the proposed approach. The implemented applications use middleware services in an Enterprise Service Bus (ESB) type architecture to share information from the electrical network. The results show that FMDE4SGRID assists in the analysis, design and coding of applications. FMDE4SGRID also supports the configuration of the middleware used for application integration.

Keywords: Smart Grids, Model-Driven Engineering, model weaving.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACM	*Association for Computing Machinery*
AMI	*Advanced Metering Infrastructure*
AMS	*Asset Management System*
API	*Aplication Programing Interface*
ATL	*Atlas Transformation Language*
CDMS	*Customer Data Management System*
CIM	*Common Information Model*
CORBA	*Common Object Request Broker Architecture*
CSV	*Comma-Separeted Values*
CWM	*Common Warehouse Metamodel*
DCPS	*Data-Centric Publish-Subscribe*
DDS	*Data Distribution Service*
DER	*Distributed Energy Resources*
DMS	*Distribution Management System*
DSL	*Domain Specific Language*
DSO	*Distribution System Operator*
EBNF	*Extended Backus-Naur Form*
EMF	*Eclipse Modeling Framework*
EMS	*Energy Management System*
EPE	*Empresa de Pesquisa Energética*
ESB	*Enterprise Service Bus*
FMDE4SGRID	*Framework Based on MDE For Smart Grids*
GIS	*Geographical Information System*

ICT	*Information and Communication Technology*
IDE	*Integrated Development Environment*
IDL	*Interface Definition Language*
IEA	*International Energy Agency*
IEC	*International Electrotechnical Commission*
IED	*Inteligent Electronic Device*
IEEE	*Institute of Electrical and Electronic Engineers*
IP	*Internet Protocol*
JNI	*Java Native Interface*
JSON	*JavaScript Object Notation*
M2T	*MDA 2 Tracks*
MDA	*Model Driven Architecture*
MDE	*Model Driven Engineering*
MDMS	*Meter Data Management System*
MOF	*Meta Object Facility*
MPC	*Make Project Creator*
NIST	*National Institute of Standards and Technology*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
PDM	*Platform Description Model*
PIM	*Platform Independent Model*
PSAL	*Power System Automation Language*
PSM	*Platform Specific Model*
QoS	*Quality of Service*
QVT	*Query*
RDFS	*Resource Description Framework Schema*

SCADA	*Supervisory Control and Data Acquisition*
SGAM	*Smart Grid Architecture Model*
SPMS	*Synchrophasor Measurement System*
TCP	*Transmission Control Protocol*
TSO	*Transmission System Operator*
UML	*Unified Modeling Language*
UUID	*Universally Unique Identifier*
WSDL	*Web Services Description Language*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*

Sumário

1 INTRODUÇÃO

Este capítulo contém o contexto, a problemática, a motivação, a hipótese, os objetivos, a justificativa, a solução proposta, a metodologia de pesquisa e a apresentação dos demais capítulos.

1.1 Contexto

A energia elétrica está estabelecida como um recurso básico para a vida humana em sociedade. De fato, a presença e a utilização da energia elétrica se tornou tão corriqueira que, ao longo do dia, não se nota mais a sua presença como uma "novidade" tecnológica. Por outro lado, a falta de energia elétrica causada, por exemplo, por um *blackout*, causa grandes transtornos, prejuízos e até mesmo danos à saúde das pessoas, já que os hospitais e os aparelhos médicos dependem da energia elétrica para funcionar. Portanto, é necessário que o fornecimento e a qualidade da energia elétrica se mantenham em níveis satisfatórios. Segundo Pinheiro (2015), a produção de parâmetros adequados de qualidade no fornecimento de energia elétrica é uma condição essencial à satisfação dos clientes comerciais e industriais, à preservação de seus equipamentos elétricos e ao desenvolvimento do mercado econômico.

No Brasil, a energia elétrica utilizada para alimentar os computadores, as lâmpadas, os sistemas de refrigeração, a bateria dos dispositivos móveis, os eletrodomésticos e outros aparelhos que utilizamos no dia-a-dia é, normalmente, gerada a milhares de quilômetros de distância de onde essa energia é consumida. Como explica Arnold (2011), a rede elétrica, em sua topologia tradicional, é projetada para transmitir a energia elétrica gerada em grande quantidade (em grande parte, a partir de usinas que utilizam combustíveis fósseis) através de linhas de transmissão e sistemas de distribuição e de forma unidirecional até os consumidores.

Geralmente, a geração de energia elétrica em grande quantidade requer a utilização de fontes de energia que causam grandes impactos ambientais, como a queima de combustíveis fósseis. Países altamente indutrializados, como os Estados Unidos e a China, possuem uma matriz energética formada, em sua maior parte, por fontes de energia não-renováveis e que causam poluição do meio ambiente. Segundo a *U.S. Energy Information Administration*[1], cerca de 63% da geração de eletricidade nos Estados Unidos em 2019 foi proveniente de

[1] A *U.S. Energy Information Administration* é uma instituição estadunidense subordinada ao Departamento de Energia dos Estados Unidos que coleta, analisa e dissemina informações independentes e imparciais sobre energia com o objetivo de promover políticas públicas, mercados eficientes e um entendimento público da energia e a sua interação com a economia e o meio ambiente. Site da EIA: <https://www.eia.gov/>. Acesso em 11/12/2020.

combustíveis fósseis. Na China, no mesmo período, esse número é de aproximadamente 68%, segundo o *website China Energy Portal*[2].

Nas últimas duas décadas, com a preocupação crescente em relação ao aquecimento global, a limitação dos recursos naturais do planeta e a necessidade de diminuírem a dependência dos países produtores de energia (por exemplo, petróleo, gás, carvão natural etc), os governos têm demonstrado cada vez mais interesse em alterar a matriz energética de seus respectivos países para incluir fontes renováveis de energia elétrica. Desta forma, o investimento dos países em fontes alternativas de energia elétrica como os painéis fotovoltaicos e turbinas eólicas têm crescido significativamente (DUPONT et al., 2015; IEA, 2019; AMBROSE, 2020).

No entanto, as fontes alternativas de energia elétrica possuem características fundamentalmente distintas das grandes plantas de geração tradicionais. As fontes alternativas têm capacidade de geração significativamente menor que as plantas de geração tradicionais. E a geração de energia elétrica em fontes alternativas, como a energia solar e a energia eólica, varia no tempo, em geral, de forma aleatória (WAN et al., 2015; REHMANI et al., 2018). A primeira diferença implica que existe a tendência de que o sistema de energia elétrica do futuro será de geração distribuída e bidirecional (geração da concessionária para o consumidor e do consumidor para a concessionária); a segunda diferença implica que o sistema elétrico deverá ter alta capacidade de sensoriamento e automação avançada para suportar o caráter variável das fontes distribuídas (STRASSER et al., 2014). Nesse contexto, a modernização da rede elétrica se mostra um passo necessário, e já é prioridade nacional para muitos países em todo o mundo (ARNOLD, 2011).

Smart Grid ("Rede Inteligente", em português) é o termo que é utilizado para descrever a modernização e digitalização do sistema de energia elétrica, que tem como objetivo torná-lo mais eficiente, integrado, flexível, limpo, resiliente e seguro (GHARAVI; GHAFURIAN, 2011). Existem várias definições diferentes de *Smart Grid* na literatura. Gharavi e Ghafurian (2011) definem *Smart Grid* como "*um sistema elétrico que utiliza fluxo de informação em dois sentidos, tecnologias de comunicação seguras e inteligência computacional de forma integrada ao longo de todo o espectro do sistema de energia elétrica, da geração ao ponto de entrega onde a energia elétrica é consumida.*". *Smart Grid* diz respeito, portanto, a um sistema que tem como objetivo digitalizar toda a rede elétrica. Entre as principais vantagens que a *Smart Grid* traz em relação à rede elétrica tradicional, Fang et al. (2011) cita as seguintes: melhora na qualidade e confiabilidade da entrega de energia elétrica; aumento da eficiência das redes elétricas existentes; manutenção preditiva e auto recuperação à falhas; facilita a instalação de fontes renováveis de energia; automatiza

[2] O *China Energy Portal* publica, em língua inglesa, informações sobre a produção de energia elétrica da China. Estas informações são publicadas originalmente por *sites* do governo chinês. O *China Energy Portal* está disponível no endereço https://chinaenergyportal.org/. Data do último acesso: 07 de dezembro de 2020.

a manutenção e automação; permite a expansão do mercado com novos produtos e serviços.

As tecnologias de informação e comunicação têm um papel essencial no contexto de *Smart Grids*, pois são essas tecnologias que permitem que sensores, dispositivos, subestações, sistemas de gerenciamento e usuários troquem dados e produzam informações. Frequentemente, as *Smart Grids* são referidas como "sistemas de sistemas" na literatura, pois envolvem, necessariamente, a integração de vários subsistemas, como SCADA (do inglês, *Supervisory Control and Data Acquisition*), SPMS (do inglês, *Synchrophasor Measurement System*) (FELIX, 2018; LIMA, 2015; FERREIRA, 2017), AMI (do inglês, *Advanced Metering Infrastructure*) (MOHAMMAD, 2018), EMS (do inglês, *Energy Management System*) (STANTON et al., 2007), entre outros. Além disso, dada a dimensão do sistema de energia elétrica, um grande volume de dados provenientes de medidores inteligentes (*Smart Meters*), sensores instalados na rede elétrica, medição fasorial, monitoramento de *status* de equipamentos e dados dos consumidores deverão ser integrados e processados para permitir a tomada de decisão de gestores e a operação do sistema (KEZUNOVIC, 2011).

Para suportar tal integração de aplicações e subsistemas, é necessário prover interoperabilidade ao longo de todo o panorama dos sistemas de energia elétrica (IVANOV et al., 2015). Segundo Kim et al. (2017), existem dois tipos de interoperabilidade que devemos considerar no contexto de *Smart Grids*: interoperabilidade de dados, que demanda que os sistemas representem a informação da mesma forma, ou de maneira semanticamente equivalente; e a interoperabilidade de comunicação, que demanda que os sistemas utilizem os mesmos protocolos de comunicação para estabelecer a troca de dados. Segundo a discussão levantada por Ivanov et al. (2015), a utilização de padrões e normas internacionais para a representação de dados é a forma mais efetiva de se atingir compatibilidade e interoperabilidade entre aplicações, modelos de negócios, concessionárias e operadores do sistema de energia elétrica. Nesse contexto, metodologias para o projeto e manutenção de modelos de dados baseados em normas para sistemas de *software* de *Smart Grids* se tornam necessárias para melhorar a interoperabilidade.

1.2 Problemática

A interoperabilidade em *Smart Grid* tem sido uma preocupação recorrente (KIM et al., 2017; IVANOV et al., 2015). A necessidade de padrões abertos para a representação de dados no domínio dos sistemas de energia elétrica levou orgãos internacionais de padronização do setor elétrico como IEEE e IEC a desenvolverem normas para a modelagem de dados para os sistemas de potência (IVANOV et al., 2015), como o conjunto de normas IEC 61970 (IEC, 2016) e IEC 61968 (IEC, 2019), que definem o que é chamado de *Common Information Model* (CIM) (GUNGOR et al., 2011); a norma IEC 61850 para subestações

(IEC, 2018); e a norma IEC 61499 para a automação de sistemas de potência (IEC, 2012), entre outras. No entanto, no domínio de *Smart Grids*, a heterogeneidade das normas, equipamentos, dispositivos e aplicações se configura em um desafio para os desenvolvedores de *software* no que diz respeito à interoperabilidade (IVANOV et al., 2015).

Diferentes organizações e grupos de trabalho têm desenvolvido *frameworks* ou arquiteturas que visam fornecer interoperabilidade para *Smart Grids* de uma maneira mais abrangente, não apenas através de formas de representação da informação. Um exemplo relevante é o *framework GridWise*, fruto do trabalho desenvolvido pelo *GridWise Architecture Council*[3]. Segundo o *GridWise*, a interoperabilidade entre sistemas é alcançada somente quando há um acordo em várias camadas de preocupação: "*Essas camadas representam os detalhes tecnológicos envolvidos na conexão entre os sistemas, o entendimento da informação trocada, os processos de negócio e os objetivos organizacionais de acordo com os negócios, economia e políticas regulatórias.*" (GWAC, 2008). Desta forma, fica claro que a utilização das normas para representação da informação nos sistemas elétricos está inserida dentro de um contexto mais amplo de interoperabilidade, no qual os sistemas devem concordar não apenas em como representam a informação, mas em tecnologias de comunicação e modelos de negócios.

A própria natureza do sistema de energia elétrica o torna um sistema extremamente robusto, complexo e com muitas subdivisões (ANDRÉN et al., 2015). Geralmente, os sistemas elétricos de potência são sistemas que: são compostos por subsistemas que são operados por empresas diferentes (por exemplo, as empresas que operam os sistemas de geração, transmissão e distribuição) que utilizam soluções diferentes para gerenciamento da informação; possuem uma quantidade imensa de dispositivos heterogêneos que geram uma grande quantidade de dados; as informações são geradas em vários formatos diferentes e existem muitos padrões disponíveis para a representação da informação. Portanto, existe a necessidade de se aplicar abordagens de desenvolvimento de sistemas de *software* que sejam pautadas pela utilização de normas e padrões da indústria, com o objetivo final de fornecer interoperabilidade em um ambiente tão desafiador como o que é prospectado para *Smart Grids*.

Diante dos desafios apresentados, as seguintes questões podem ser levantadas: como permitir a integração das informações geradas pelas aplicações de *software* no sistema de energia elétrica? Como realizar, dentro de uma abordagem de desenvolvimento de *software*, o geranciamento de padrões, normas e arquiteturas de sistemas de *software* para o domínio de *Smart Grids*?

[3] O *GridWise Architecture Council* é uma equipe de líderes da indústria que tem como missão dar forma a uma arquitetura para sistemas de energia elétrica altamente inteligente e interativa. Esta arquitetura fornecerá as diretrizes para a interação entre os participantes e interoperabilidade entre tecnologias e sistemas. Mais informações no site: <https://www.gridwiseac.org/>. Acesso em: 11/12/2020.

1.3 Motivação

Na literatura, a interoperabilidade e a integração de aplicações para *Smart Grids* têm sido temas recorrentes (SOUVENT et al., 2019; ALI et al., 2016; KIM et al., 2017; IVANOV et al., 2015; WIDERGREN et al., 2010). Para permitir que haja interoperabilidade entre as aplicações de *Smart Grids* é necessário que estas aplicações sejam desenvolvidas com base em padrões e normas e que esses padrões sejam harmonizados e gerenciados dentro de uma abordagem de desenvolvimento. Uma alternativa viável para solucionar esse desafio é considear que os padrões e normas da indústria são modelos computacionais bem definidos e utilizar as ferramentas computacionais apropriadas para manipular esses modelos (ANDRÉN et al., 2014).

A Engenharia Dirigida por Modelos (MDE - *Model Driven Engineering*) é uma abordagem de desenvolvimento de *software* que tem sido proposta na literatura para gerenciar a crescente complexidade dos sistemas de *software*. Em MDE, o modelo é o principal artefato do processo de desenvolvimento de *software* (BÉZIVIN, 2005). A MDE tem como objetivo abstrair os conceitos relevantes de domínio e a lógica de negócio para proteger os sistemas de *software* das mudanças e complexidade das plataformas de implementação e, além disso, representar os requisitos dos sistemas de maneira mais efetiva, através das linguagens específicas de domínio ou DSLs (*Domain Specific Languages*) (SCHMIDT, 2006).

No início dos anos 2000, a OMG padronizou a Arquitetura Dirigida por Modelos (MDA - *Model Driven Architecture*), que é um *framework* baseado nos princípios da MDE (SIEGEL, 2014). A MDA surgiu com o objetivo de integrar os diversos padrões e *middlewares* que estavam surgindo em profusão na época, como WSDL, .NET e CORBA (SOLEY, 2000). A MDA, em essência, consiste em dois mecanismos principais: centralizar o gerenciamento dos modelos, utilizando uma forma de representação comum, geralmente baseada em MOF (*Meta Object Facility*) e UML (*Unified Modeling Language*); e permitir a troca de dados e persistência dos modelos utilizando um formato comum, que é o XMI (XML *Metadata Interchange*) (KLEPPE et al., 2003).

Desde então, a MDA tem evoluído e várias tecnologias relacionadas à sua implementação foram desenvolvidas, como a separação entre os modelos independentes de plataforma (PIM), modelos específicos de plataforma (PSM) e a implementação do *software*; representação dos modelos em vários níveis de abstração (modelos, metamodelos e metametamodelos); e as linguagens de transformação de modelos (KLEPPE et al., 2003).

Outra operação utilizada no contexto de MDA é a técnica de *weaving* de modelos (FABRO et al., 2005; FABRO et al., 2006). O *weaving* de modelos contribui para a separação de preocupações, pois permite que modelos que representam diferentes aspectos de *software* (por exemplo, aspectos de segurança, tempo real ou interoperabilidade) desenvolvidos

separadamente sejam integrados ao PIM para então gerar os PSMs (FABRO et al., 2005). Isso permite que decisões sobre determinados aspectos do *software* sejam formalizadas em modelos separados dentro do processo de desenvolvimento, o que aumenta a possibilidade de reuso dos modelos e a preservação dos investimentos.

A MDE tem sido utilizada com sucesso em vários domínios de aplicação, como *Big Data* (JUNIOR, 2017), Aplicações Móveis (STEFANELLO, 2017) e Computação em Nuvem (MATOS, 2015). Em *Smart Grids*, a MDE tem sido utilizada por vários autores (NEIS et al., 2019; ANDRÉN et al., 2015; ANDRÉN et al., 2017; ANDRÉN et al., 2016; KAITOVIC; LUKOVIC, 2011; EBEID et al., 2016) para suportar o desenvolvimento de aplicações a partir de arquiteturas computacionais construídas especificamente para os sistemas de energia elétrica, como o SGAM (CEN-CENELEC-ETSI, 2012). Em geral, os resultados desses estudos têm mostrado que existe uma boa perspectiva acerca da evolução de metodologias baseadas em MDE para o desenvolvimento de *software* no domínio de *Smart Grids* (NEIS et al., 2019).

1.4 Hipótese

As informações manipuladas em aplicações de *Smart Grids* têm características bem específicas do domínio dos Sistemas de Energia Elétrica. Isto é, estas informações contêm dados como: características de equipamentos, como transformadores, chaves e linhas; medições de tensão, corrente, frequência, temperatura etc.; descrição de consumidores e produtores de energia elétrica; e a representação da topologia da rede elétrica. Existem normas e padrões que foram desenvolvidos para representar essas informações específicas de domínio, como o padrão IEC 61970/61968, que define o CIM.

Devido à relevância destas normas, como o CIM, a prática mais comum, em abordagens para o desenvolvimento de *software* para *Smart Grids* encontradas na literatura (ANDRÉN et al., 2016; LOPEZ et al., 2010; ANDRÉN et al., 2014; KIM et al., 2017; ANDRÉN et al., 2017; ANDRÉN et al., 2015; FISCHINGER et al., 2019; KIM et al., 2014), é fazer a separação entre o modelo de dados da rede elétrica, o qual alguns autores definem como modelo semântico de domínio, e o modelo que descreve a aplicação de *Smart Grids*, que corresponde ao modelo da lógica de negócio do sistema. O *framework* SGAM, por exemplo, faz uma clara separação entre a camada de Negócios (*Business Layer*), a camada de Funções (*Function Layer*) e a camada de Informações (*Information Layer*). O SGAM, no entanto, não fornece qualquer recomendação sobre como estas camadas serão "conectadas".

Em MDE, os trabalhos de Fabro et al. (2005) e Belangour et al. (2006) lançam luz sobre o desenvolvimento baseado em Y, em que diferentes aspectos de *software*, representados por modelos de descrição de plataformas - PDMs (do inglês, *Platform*

Description Models), são desenvolvidos independentemente do PIM. Um PDM pode ser um modelo de aspectos de segurança, um modelo de execução em tempo real, um modelo de interoperabilidade, um conjunto de *datatypes* etc. O PIM é a lógica de negócio da aplicação, ou seja, descreve as funcionalidades e objetivos do *software*. PIM e PDM são entrelaçados através da operação de *weaving*, e o PSM é gerado a partir deste entrelaçamento.

Portanto, a hipótese deste trabalho é que a utilização de uma abordagem baseada em MDE e *weaving* de modelos para o desenvolvimento de software para *Smart Grids* permite separar o desenvolvimento do modelo de dados da rede elétrica (PDM) do modelo da lógica de negócio da aplicação (PIM), permitindo o seu reúso. A operação de *weaving* permite registrar como os elementos da lógica de negócio das aplicações de *Smart Grids* se relacionam com os elementos do modelo de dados da rede elétrica, que é baseado em uma norma da indústria, como o CIM. Esse tipo de abordagem permite que o mesmo modelo de dados da rede elétrica seja utilizado por uma infinidade de aplicações dentro do domínio de *Smart Grids*, facilitando a integração de aplicações.

1.5 Solução Proposta

Neste trabalho, o FMDE4SGRID (*Framework* Baseado em MDE para *Smart Grids*) é proposto. O FMDE4SGRID auxilia o desenvolvimento de aplicações de análise de dados e gerenciamento da informação para *Smart Grids*. A separação de preocupações é suportada através da utilização da técnica de *weaving* de modelos. O FMDE4SGRID suporta a definição de um PIM, que é conforme um metamodelo de *Smart Grids*, e que contém a lógica de negócio da aplicação. A definição de um ou mais PDMs também é suportada. O PDM contém o modelo de dados do Sistema de Potência baseado em norma CIM ou, possivelmente, em outra norma. PIM e PDM são modelos independentes entre si e independentes de plataforma. Um modelo de *weaving* é utilizado para registrar as relações entre os *datatypes* do PIM e os *datatypes* do PDM.

PIM, PDM e *weaving* são produzidos manualmente e são os modelos de entrada do FMDE4SGRID. Um motor de transformação entre modelos, com base em uma definição de transformação, gera automaticamente um PSM Abstrato a partir dos modelos de entrada, contendo a descrição das funcionalidades da aplicação desenvolvida dentro de uma plataforma específica. Um motor de transformação também é utilizado para gerar automaticamente um PSM Concreto a partir do PSM Abstrato. O PSM Concreto contém, em geral, a implementação da aplicação em uma linguagem de programação específica, como o Java. Por fim, o FMDE4SGRID suporta a geração de código fonte a partir do PSM Concreto.

A plataforma específica escolhida para implementar as aplicações de *Smart Grids* desenvolvidas neste trabalho é o *middleware OpenDDS* (OCI, 2020), que é uma

implementação em código aberto da especificação DDS (*Data Distribution Service*) da OMG (OMG, 2015). O DDS é um *middleware* distribuído centrado em dados que permite a troca de dados entre aplicações através de um mecanismo de publicador-subscritor (OMG, 2021). Uma arquitetura de comunicação do tipo *Enterprise Service Bus* com o *middleware* DDS é implementada neste trabalho para realizar a integração de aplicações de *Smart Grids*.

É importante ressaltar que, apesar da geração automática dos modelos específicos de plataforma e do código fonte, geralmente a intervenção manual do usuário na edição dos modelos é necessária. Além disso, o código fonte gerado contém apenas o esqueleto do código, com a definição das classes e interfaces com os seus atributos, métodos e parâmetros. Ou seja, o comportamento dos métodos não é gerado e precisa ser escrito manualmente. Desta forma, pode-se afirmar que o FMDE4SGRID auxilia no desenvolvimento de *software* de maneira semiautomática e não automática.

1.6 Justificativa

Smart Grids é um tema muito abrangente e multidisciplinar, que gera desafios em diversas áreas, como a Eletrônica de Potência (YU et al., 2011), a Inteligência Artificial (RAMCHURN et al., 2012), o Controle (ANDERSON et al., 2011), o *Big Data* (MARLEN et al., 2019), a *Internet of Things* (IoT) (REKA; DRAGICEVIC, 2018), as Redes Definidas por *Software* (REHMANI et al., 2019), entre outras. Dentro da área de Engenharia de *Software*, garantir a interoperabilidade entre soluções de *software* para *Smart Grids* é um desafio relevante e difícil, dada a complexidade dos sistemas de energia elétrica e a heterogeneidade dos elementos que o compõem (KIM et al., 2017).

Segundo Ivanov et al. (2015), um fator de sucesso para *Smart Grids* e que facilita a interoperabilidade é a utilização de padrões abertos, como o CIM, para a representação da informação em Sistemas Elétricos de Potência. Desta forma, as abordagens da Engenharia de *Software* para *Smart Grids*, inclusive aquelas baseadas em MDE, como as abordagens apresentadas por Dänekas et al. (2014), Andrén et al. (2016) e Fischinger et al. (2019), utilizam algum padrão internacional - geralmente o CIM ou o IEC 61850 - para a representação de dados dos sistemas de potência. Além disso, estes autores prepõem que haja uma separação entre a modelagem da lógica de negócio do sistema e a modelagem de dados do sistema. No entanto, não há, entre os trabalhos analisados, um método bem definido e baseado em modelos para especificar como os elementos dos modelos de lógica de negócio se relacionam com os elementos dos modelos de dados.

Pretende-se, nesta pesquisa, investigar a utilização do conceito de M2T (do inglês, *MDA 2 Tracks*, que significa "MDA em dois caminhos/ramos") também conhecido como *desenvolvimento baseado em Y*, apresentado por Belangour et al. (2006), e da operação

de *weaving* de modelos (FABRO et al., 2005) para especificar, de maneira formal, a relação entre a lógica de negócio das aplicações de *Smart Grids* e os modelos de dados dos Sistemas de Potência, que são conforme o CIM. Um *framework* baseado em MDE e *weaving* é fornecido e implementado para dar suporte à abordagem proposta e auxiliar o desenvolvimento e integração de aplicações de *Smart Grids*.

Outros trabalhos da literatura recente utilizam os conceitos de MDE, M2T e *weaving* para suportar o desenvolvimento de *software* em outros domínios. Estes trabalhos atestam a viabilidade da utilização de *weaving* de modelos para representar a relação entre a lógica de negócio da aplicação de *software* e outros aspectos do *software*, como é exemplificado a seguir. Matos (2015) propõe um *framework* para suportar o desenvolvimento de *software* para plataformas de Computação em Nuvem considerando aspectos de segurança. Matos (2015) separa os aspectos de segurança, que são representados em um modelo de segurança (PDM), da lógica de negócio da aplicação (PIM). Estes modelos são entrelaçados através da operação de *weaving* para gerar os modelos específicos de plataforma (PSMs) e, posteriormente, gerar o código fonte de aplicações que seguem o modelo de *Software as a Service* (SaaS). Junior (2017) também utiliza os conceitos de M2T e *weaving* de modelos em um *framework* baseado em MDE para suportar o desenvolvimento de *software* para plataformas de *Big Data*. Neste caso, há a separação entre a lógica de negócio (PIM) e aspectos de *Map Reduce*, representados em um modelo de *Map Reduce* (PDM). Estes dois aspectos são entrelaçados através do *weaving*.

Junior (2017) afirma que a separação de preocupações pode ser estendida através da operação de *weaving*, e que vários PDMs podem ser adicionados e entrelaçados ao PIM para incluir outros aspectos do *software*. Stefanello (2017) reafirma e demonstra a inclusão de vários aspectos de *software* no processo de desenvolvimento através da operação de *weaving* e o desenvolvimento baseado em Y.

Há, portanto, com a realização do presente trabalho, a perspectiva de se utilizar a operação de *weaving* e o conceito de desenvolvimento baseado em Y no domínio de *Smart Grids* para garantir que o modelo da lógica de negócio das aplicações utilize os tipos de dados definidos em um modelo de dados que é conforme um padrão internacional do IEC ou IEEE, facilitando a interoperabilidade. Além disso, este trabalho abre possibilidades de continuidade da pesquisa no sentido de incluir outros aspectos de *software* ao processo de desenvolvimento de aplicações para *Smart Grids*, como segurança e operação em tempo real, preservando a separação de preocupações.

1.7 Objetivos

Os objetivos deste trabalhos de pesquisa são divididos em Objetivo Geral e Objetivos Específicos, como segue nas próximas subseções.

1.7.1 Objetivo Geral

O objetivo geral deste trabalho é propor, desenvolver e implementar um *framework* baseado nos conceitos de MDE e *weaving* de modelos para produzir aplicações de *Smart Grids* que se comuniquem em um ambiente de compartilhamento de dados conforme o padrão CIM para sistemas elétricos.

1.7.2 Objetivos Específicos

Este trabalho possui os seguintes objetivos específicos:

- Criar um *framework* baseado em MDE e *weaving* que dê suporte ao desenvolvimento de *software* para *Smart Grids*;
- Criar uma metodologia para servir como guia para a utilização do *framework* proposto;
- Criar um metamodelo de *Smart Grids* para a definição de modelos de lógica de negócio para aplicações no domínio de *Smart Grids*;
- Criar um metamodelo de DDS para a definição de modelos de aplicações de plataformas baseadas em DDS;
- Desenvolver um metamodelo de *weaving* que atenda às necessidades de separação de preocupações dentro do domínio de *Smart Grids*, principalmente a separação entre o CIM da rede elétrica e o PIM da aplicação;
- Desenvolver uma arquitetura de *middleware* baseada em DDS que dê suporte à integração de aplicações para *Smart Grids*;
- Criar uma prototipação do *framework* proposto em ambiente Eclipse;
- Projetar e implementar, com auxílio do *framework* proposto, aplicações para análise de dados em *Smart Grids*.

1.8 Metodologia de Pesquisa

A metodologia de pesquisa utilizada neste trabalho é descrita como segue:

1. Pesquisa bibliográfica em livros, artigos de conferências, artigos de períodicos, páginas da Web e normas com o objetivo de levantar conceitos, fundamentos e o estado da arte dos temas:

- Engenharia Dirigida por Modelos (MDE), incluindo a MDA, *weaving* e desenvolvimento baseado em Y;
- *Smart Grids*;
- Interoperabilidade entre sistemas em *Smart Grids*;
- Normas para padronizar a comunicação e representação da informação em *Smart Grids*;
- Metodologias e abordagens para o desenvolvimento de *software* para *Smart Grids*;
- Aplicação de MDE em *Smart Grids*.

2. Proposta de um *framework* para auxiliar o desenvolvimento de *software* e a integração de aplicações de *Smart Grids*. Esta etapa pode ser dividida em etapas menores, como segue:

   a) Definição da arquitetura do *framework*. O *framework* deve suportar: a definição de um modelo independente de plataforma (PIM - *Platform Independent Model*) que representa a lógica de negócio da aplicação com os conceitos de domínio; a definição de um modelo contendo a informação a ser compartilhada entre as aplicações, ou seja, um modelo semântico comum que seja compatível com uma norma do IEC; a geração automática de um modelo específico de plataforma abstrato, o PSM abstrato (*Platform Specific Model*), contendo os conceitos e artefatos necessários para a implementação da aplicação em DDS; a geração automática ou semiautomática de um PSM concreto a partir do PSM abstrato, contento os detalhes de implementação da aplicação em uma linguagem de programação específica; geração automática do código fonte;

   b) Definição e desenvolvimento dos metamodelos que servirão para a definição dos modelos utilizados no *framework* proposto. Os metamodelos necessários são listados a seguir:

      - Metamodelo de *Smart Grids*, para a definição do PIM;
      - Metamodelo de *weaving*, para a definição de um modelo que irá entrelaçar os elementos do PIM e do PDM;
      - Metamodelo de DDS, para a definição do PSM abstrato;
      - Metamodelo da linguagem Java, para a definição do PSM concreto.

   c) Criação das definições de transformação de modelo-a-modelo e modelo-a-texto. O *framework* proposto deve suportar as seguintes transformações de modelo-a-modelo:

      - Transformação entre os modelos de entrada (PIM, PDM e *weaving*) e o PSM abstrato;

- Transformação entre o PSM abstrato e o PSM concreto.

d) Criação de definições de transformação para a geração de código a partir do PSM concreto.

3. Implementação do *framework* proposto no IDE Eclipse. O *Eclipse Modeling Framework* (EMF) é utilizado para implementar os metamodelos e os *plugins* para edição dos modelos. A linguagem QVT é utilizada para especificar as definições de transformação dentro do ambiente Eclipse;

4. Avaliação da abordagem proposta por meio do desenvolvimento de exemplos ilustrativos e comparações com outros trabalhos encontrados na literatura.

1.9 Apresentação da Dissertação

Este manuscrito é composto por oito capítulos no total, dos quais os sete capítulos restantes desta dissertação são descritos como segue:

- O Capítulo 2 contém os principais conceitos e tecnologias utilizadas no desenvolvimento desta dissertação;
- O Capítulo 3 contém o estado da arte, onde os trabalhos sobre metodologias de desenvolvimento de *software* visando a melhora da interoperabilidade de sistemas de *software* para *Smart Grids* são descritos;
- No Capítulo 4, o *framework* proposto para suportar o desenvolvimento de *software* para sistemas de *Smart Grids* é descrito;
- O Capítulo 5 contém um protótipo do *framework* proposto, desenvolvido no ambiente Eclipse;
- No Capítulo 6, o desenvolvimento de aplicações para *Smart Grids* utilizando o *framework* proposto é demonstrado;
- O Capítulo 7 contém os resultados obtidos durante os experimentos com o *framework* proposto e mostra uma comparação com os resultados que os trabalhos descritos no Capítulo 3 obtiveram;
- Por fim, o Capítulo 8 contém as conclusões, contendo os objetivos alcançados, as principais limitações e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo contém os principais conceitos e tecnologias exploradas durante o desenvolvimento desta dissertação.

O conceito de *Smart Grid* é apresentado a partir de uma perspectiva sobre como a sociedade tem produzido e consumido energia elétrica. Os desafios envolvendo a implementação de *Smart Grids* também são discutidos. A especificação do *middleware* DDS (*Data Distribution Service*) também é abordada neste capítulo, além das suas principais implementações e aplicações. Por fim, uma discussão sobre MDE, MDA, *Model Weaving* e EMF é apresentada, não sem antes posicionar o leitor acerca de um contexto histórico sobre a evolução e os desafios da Engenharia de *Software*.

2.1 *Smart Grids*

Esta seção traz uma discussão sobre *Smart Grids*, termo utilizado para descrever a integração dos sistemas de informação e comunicação ao sistema de energia elétrica com o objetivo de tornar o sistema de energia elétrica mais eficiente e mais sustentável. Começamos esta seção problematizando a questão energética e mostrando o surgimento da necessidade de se pensar em formas alternativas de geração de energia elétrica e o desafio de integrar as fontes distribuídas de energia ao sistema elétrico. O conceito de *Smart Grid* é então apresentado, e as principais implicações e desafios que este conceito traz são discutidos.

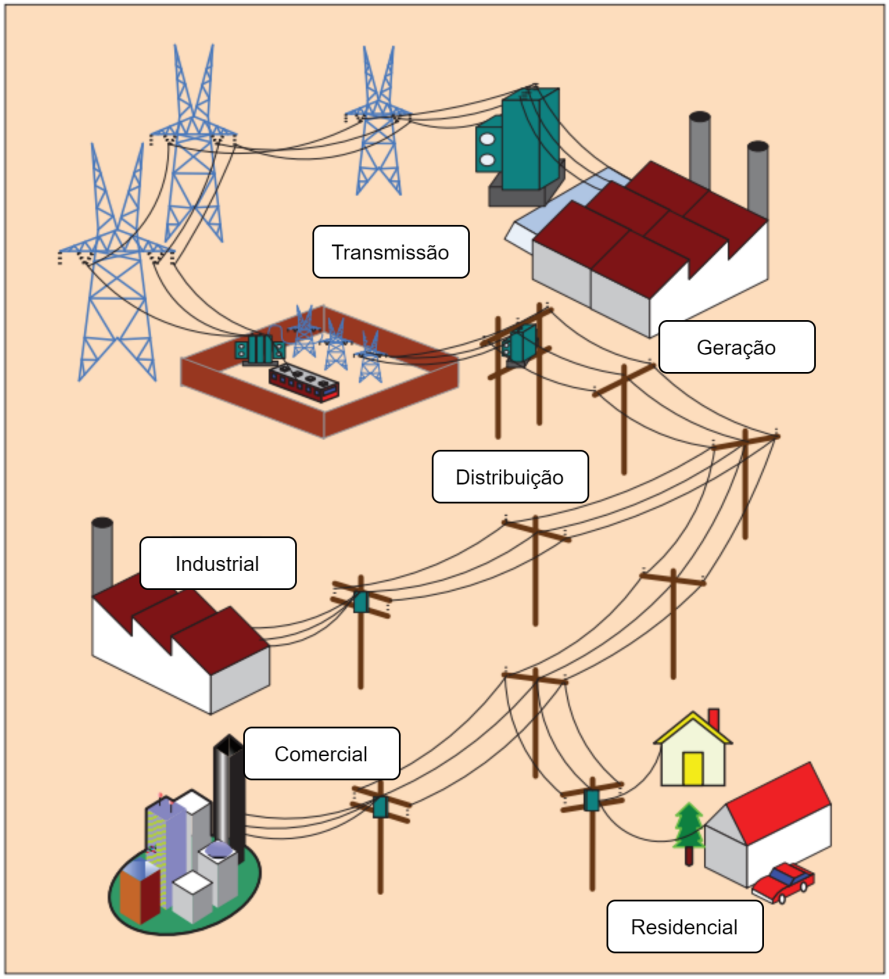
2.1.1 A questão da sustentabilidade da matriz elétrica

O sistema de energia elétrica ou sistema de potência (*power system*, em inglês) é responsável por produzir a energia elétrica a partir de alguma fonte de geração (fonte primária) — usina hidrelétrica, usina termoelétrica, usina nuclear etc. — e transmitir a energia elétrica produzida até o consumidor, onde os equipamentos elétricos são alimentados. No século XX, os trabalhos e descobertas de Thomas Edison e Nikola Tesla formaram as bases para a construção das redes elétricas modernas, compostas de geração, transmissão e distribuição, como ilustrado no esquema da Figura 2.1 (YU et al., 2011). Desde então, a eletricidade tornou-se essencial para as atividades econômicas e para a sociedade como um todo, e a demanda por energia elétrica continua a crescer. Uma estimativa da *World Energy Outlook*[1] mostra que a demanda mundial por energia elétrica aumentará em 25% até 2040

[1] O *World Energy Outlook* é uma publicação anual da Agência Internacional de Energia (IEA, do inglês, "*International Energy Agency*"), que tem o objetivo de fornecer uma visão sobre como o sistema de energia global pode se desenvolver nas próximas décadas.

(IEA, 2018). Além disso, com a utilização pervasiva de eletrônicos e microprocessadores e a dependência de energia elétrica para o funcionamento de serviços essenciais, a qualidade e a confiabilidade da energia elétrica entregue ao consumidor tem se tornado cada vez mais importante (ARNOLD, 2011).

Figura 2.1 – Esquema da rede elétrica tradicional.



Fonte: (YU et al., 2011).

No mundo, cerca de 75% da geração de energia elétrica é baseada em fontes de energia não-renováveis, sendo a maior parte constituída por combustíveis fósseis como o carvão mineral (38%) e o gás natural (23%)[2]. Nestes casos, a energia elétrica é obtida em grande quantidade a partir da queima dos combustíveis fósseis nas termoelétricas e é transmitida por longas distâncias pelos sistemas de transmissão. Outro exemplo relevante de fonte não-renovável de energia é a energia nuclear.

As fontes de energia não-renováveis possuem dois problemas principais. O primeiro e mais óbvio é a escassez dos recursos naturais para a produção de energia. O gás natural,

[2] Fonte: <https://www.epe.gov.br/pt/abcdenergia/matriz-energetica-e-eletrica>. Último acesso em 13/05/2021.

o petróleo e seus derivados e o carvão mineral têm disponibilidade limitada no planeta. Estima-se que, se o ritmo de exploração dos recursos naturais continuar o mesmo, o planeta Terra tenha reserva de fontes não-renováveis de energia por apenas mais algumas décadas (YU et al., 2011; GOLDEMBERG; LUCON, 2007).

O segundo problema relacionado às fontes não-renováveis é a poluição e os impactos ambientais causados, principalmente, pela queima de combustíveis fósseis. Os impactos ambientais podem ocorrer em nível local, causando a poluição urbana e aumentando as doenças respiratórias, cardiovasculares e câncer (MOLINA; MOLINA, 2004); em nível regional, provocando as chuvas ácidas que atingem diretamente ecossistemas, plantações, edifícios históricos, estruturas etc.; e em nível global, causando o efeito estufa e as mudanças climáticas (GOLDEMBERG; LUCON, 2007).

No Brasil, diferentemente do resto do mundo, a maior parte da matriz elétrica e energética é composta por energias renováveis, pois a maior parte da geração de energia elétrica no Brasil provém das hidrelétricas. Segundo dados da EPE (Empresa de Pesquisa Energética), somando a energia hidráulica (64,9%), a biomassa (8,4%), a energia eólica (8,6%) e a solar (1%), o Brasil possui cerca de 83% de sua matriz elétrica composta de fontes renováveis.

No entanto, como afirma Dupont et al. (2015), a grande dependência da matriz elétrica por hidrelétricas vem ameaçando a geração de energia elétrica no Brasil principalmente por fatores climáticos, como as estiagens prolongadas que fazem com que os reservatórios de diversas hidrelétricas atinjam níveis críticos. Esta situação faz com que as termelétricas sejam acionadas para suprir a demanda de energia elétrica nos períodos de seca. Como a produção de energia elétrica nas termelétricas é mais cara em relação à produção nas hidrelétricas, isso acarreta um aumento do custo de produção de eletricidade o que reflete no aumento da tarifa de energia elétrica. Além disso, os impactos ambientais e sociais causados pela implantação de grandes hidrelétricas também não podem ser desprezados (DUPONT et al., 2015).

Também é importante notar que, devido à distância entre os grandes centros de geração e os consumidores de energia elétrica, muitas perdas ocorrem ao longo dos sistemas de transmissão e distribuição. Segundo dados apresentados por Dupont et al. (2015), no ano de 2015, de toda a energia produzida no Brasil, 15,3% são perdas, o que equivale a 93,6 TWh. A título de comparação, a usina de Itaipu, uma das maiores hidrelétricas do mundo, gerou 98,3 TWh no mesmo ano (DUPONT et al., 2015).

Devido a todas essas questões, um processo de conscientização tem sido observado em governos e na sociedade como um todo acerca da necessidade de se utilizar os recursos energéticos do planeta com mais racionalidade e sustentabilidade. Nas últimas décadas, há uma clara tendência no mundo de se empreender incentivos à implementação e integração de fontes renováveis e distribuídas de energia elétrica, como a eólica, fotovoltaica e biomassa,

em detrimento às fontes tradicionais de energia elétrica, principalmente aquelas que agridem o meio ambiente, como as usinas de carvão.

No caso específico do Brasil, como aponta Pacheco (2006), a diversificação da matriz enérgica é muito importante para garantir a continuidade de fornecimento de energia elétrica. Portanto, há um entendimento na literatura de que, a longo prazo, a tendência é de que a topologia da rede elétrica seja, em maior parte, distribuída, dando preferência para a geração e microgeração a partir de fontes renováveis de energia.

2.1.2 O conceito de *Smart Grids*

Em geral, as fontes renováveis de energia elétrica possuem a característica de serem intermitentes e variáveis ao longo do tempo (DUPONT et al., 2015). Os painés solares fotovoltaicos só produzem energia elétrica em quantidade razoável durante o dia e quando o céu não está nublado; as turbinas eólicas produzem energia elétrica de acordo com a força dos ventos, o que pode variar durante o dia e durante as diferentes épocas do ano; as usinas hidrelétricas podem ser dramaticamente afetadas pelos períodos de seca, pois elas dependem das chuvas para manter os reservatórios em níveis adequados para a produção de energia elétrica.

Tornar o sistema elétrico mais resiliente e flexível para acomodar as variações e a intermitência das fontes renováveis, além de suportar a integração cada vez maior da geração distribuída e a conexão de veículos elétricos à rede requer um sistema elétrico com automação avançada, mais inteligente e com uma grande capacidade de sensoreamento e processamento de dados (GHARAVI; GHAFURIAN, 2011; CORRÊA, 2018).

O avanço tecnológico da computação e das tecnologias de informação e comunicação abriu grandes oportunidades para realizar a transição da rede elétrica tradicional para uma rede elétrica mais flexível, eficiente, sustentável e resiliente. *Smart Grid* é o termo designado para descrever esta "nova" rede elétrica. Em português, o termo "*Smart Grid*" quer dizer "Rede Inteligente", em tradução livre. A ideia é exatamente adicionar inteligência à rede elétrica através da utilização de comunicação e processamento de dados, computação, sensoriamento em toda a rede, automação e integração entre os dispositivos e subsistemas que compõem o sistema de energia elétrica (DOE, 2020).

Não existe uma definição única para *Smart Grid* na literatura. Nas palavras de Yu et al. (2011), "*O termo Smart Grid refere-se à rede elétrica que pode integrar inteligentemente os comportamentos e ações de todos os usuários conectados a ela, e.g., produtores, consumidores, e aqueles que são ambos – para, de maneira eficiente, entregar eletricidade sustentável, econômica e segura.*". Para Gharavi e Ghafurian (2011), "*Smart Grid pode ser definida como um sistema elétrico que utiliza informação, tecnologias de comunicação, e inteligência computacional de maneira integrada ao longo de todo o espectro*

do sistema de energia, da geração ao ponto de consumo da eletricidade."

Portanto, em essência, *Smart Grid* diz respeito a integrar tecnologias avançadas de informação, comunicação, sensoriamento e controle ao sistema de potência para criar um ambiente integrado de geração e consumo de energia elétrica que favorece o surgimento de novas oportunidades de negócio e maior participação dos consumidores (ARNOLD, 2011). Entre os principais benefícios de *Smart Grids*, Fang et al. (2011) cita os seguintes:

- Melhora da qualidade e confiabilidade da energia elétrica;
- Otimização da utilização das instalações elétricas, evitando a construção de plantas de geração sobressalentes;
- Melhora da capacidade e eficiência das redes elétricas existentes;
- Melhora da resiliência a faltas;
- Manutenção preditiva e autorrecuperação em caso de falhas e distúrbios da rede;
- Facilitar a implementação de fontes renováveis de energia elétrica;
- Acomodar as fontes distribuídas de energia;
- Automatizar a manutenção e operação;
- Redução da emissão de gases poluentes através da acomodação dos veículos elétricos e fontes alternativas de energia;
- Redução do consumo de óleo através da redução da necessidade por geração ineficiente durante os horários de pico;
- Apresentar oportunidades para melhorar a segurança da rede;
- Habilitar a transição para os veículos elétricos e novas opções para armazenamento de energia;
- Aumento de escolha do consumidor;
- Possibilidades de novos produtos, serviços e mercados.

2.1.3 *Smart Grids* como um sistema de sistemas

A utilização de computação, comunicação e sistemas de automação para gerenciar sistemas elétricos não é uma novidade. Os sistemas SCADA (do inglês, *Supervisory Control and Data Acquisition*), por exemplo, surgiram na década de 1960 (UJVAROSI, 2016), com o objetivo de monitorar e controlar equipamentos em indústrias. A evolução dos computadores e das infraestruturas de comunicação permitiram que sistemas cada vez mais

sofisticados fossem desenvolvidos para monitorar subestações e as redes de transmissão e distribuição de energia elétrica. A partir dos anos 2000 - com a crescente utilização da internet pública em aplicações industriais - aplicações de monitoramento e análise de dados para os sistemas de energia elétrica, como EMS (do inglês, *Energy Management System*) e DMS (do inglês, *Distribution Management System*), se popularizaram, e os sistemas de monitoramento já existentes, como o SCADA, ganharam arquiteturas mais avançadas (ANTON et al., 2017).

A novidade que a ideia de *Smart Grids* traz para este contexto é integrar os sistemas de automação existentes, como o SCADA, e os novos sistemas e aplicações inteligentes como o EMS e o DMS utilizando uma arquitetura de comunicação avançada (GUNGOR et al., 2012). Desta forma, como aponta Arnold (2011), a *Smart Grid* pode ser caracterizada como um "sistema de sistemas". A palavra-chave para *Smart Grids* é, portanto, *integração*.

As Tecnologias de Informação possuem um papel crítico dentro desse ambiente de integração em *Smart Grids*, pois este sistema de sistemas envolverá o *networking* de um vasto número de sensores, *smart meters*, sistemas SCADA e aplicações de análise dados (ARNOLD, 2011). Prover interoperabilidade entre todos esses elementos é um grande desafio, dada a dimensão dos sistemas de potência com todos os seus subsistemas e a heterogeneidade dos dispositivos, normas e requisitos de aplicações (IVANOV et al., 2015; ARNOLD, 2011).

2.1.4 Padronização de *Smart Grids*

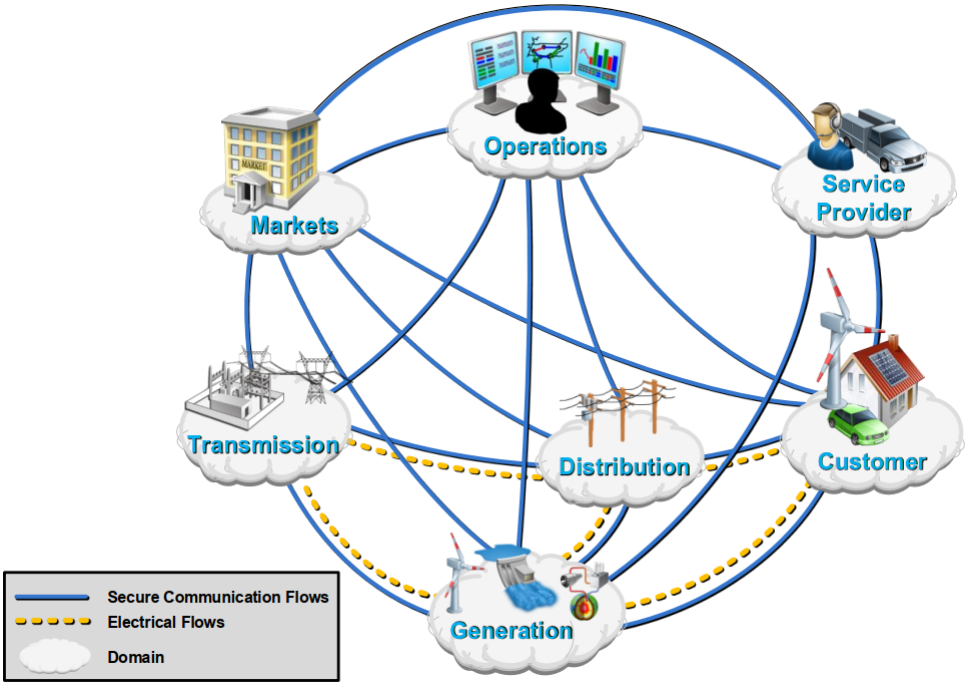
Garantir a interoperabilidade em *Smart Grids* requer uma padronização dos projetos e dos dados a serem compartilhados entre as aplicações. Nos EUA, o NIST (*National Institute of Standards and Technology*) é o órgão responsável por coordenar o desenvolvimento de um *framework* que inclui protocolos e modelos padronizados de gerenciamento da informação para garantir a interoperabilidade dos sistemas e dispositivos de *Smart Grids* (NIST, 2014).

Dentro dos esforços para padronizar *Smart Grids*, o NIST desenvolveu um modelo conceitual de *Smart Grids*, apresentado na Figura 2.2. Segundo o NIST (2014), "*Este modelo conceitual suporta o planejamento, elaboração de requisitos, documentação e a organização de uma coleção de redes interconcetadas e equipamentos que irão compor a Smart Grid*".

Dentro do modelo conceitual do NIST, a *Smart Grid* é dividida em seis domínios (NIST, 2014):

1. *Customer* (Consumidor): O usuário final de eletricidade. Pode também gerar, armazenar e gerenciar o uso de eletricidade. Tradicionalmente, três tipos de consumidores são considerados: residencial, comercial e industrial;

Figura 2.2 – Modelo conceitual de *Smart Grids*.



Fonte: (NIST, 2014).

2. *Markets* (Mercados): Os operadores e participantes do mercado de energia elétrica;

3. *Service Provider* (Provedor de Serviço): As organizações que fornecem serviços para os consumidores de eletricidade e para as concessionárias;

4. *Operations* (Operações): Consiste em quem faz o gerenciamento do fluxo da energia elétrica;

5. *Generation* (Geração): Os geradores de energia elétrica. Podem também armazenar energia para distribuição futura. Este domínio inclui as fontes tradicionais de geração elétrica e as fontes de geração distribuída;

6. *Transmission* (Transmissão): Responsável por levar grandes quantidades de energia elétrica por longas distâncias. Pode também armazenar e gerar eletricidade;

7. *Distribution* (Distribuição): Os distribuidores de eletricidade para e a partir dos consumidores. Pode também armazenar e consumir eletricidade.

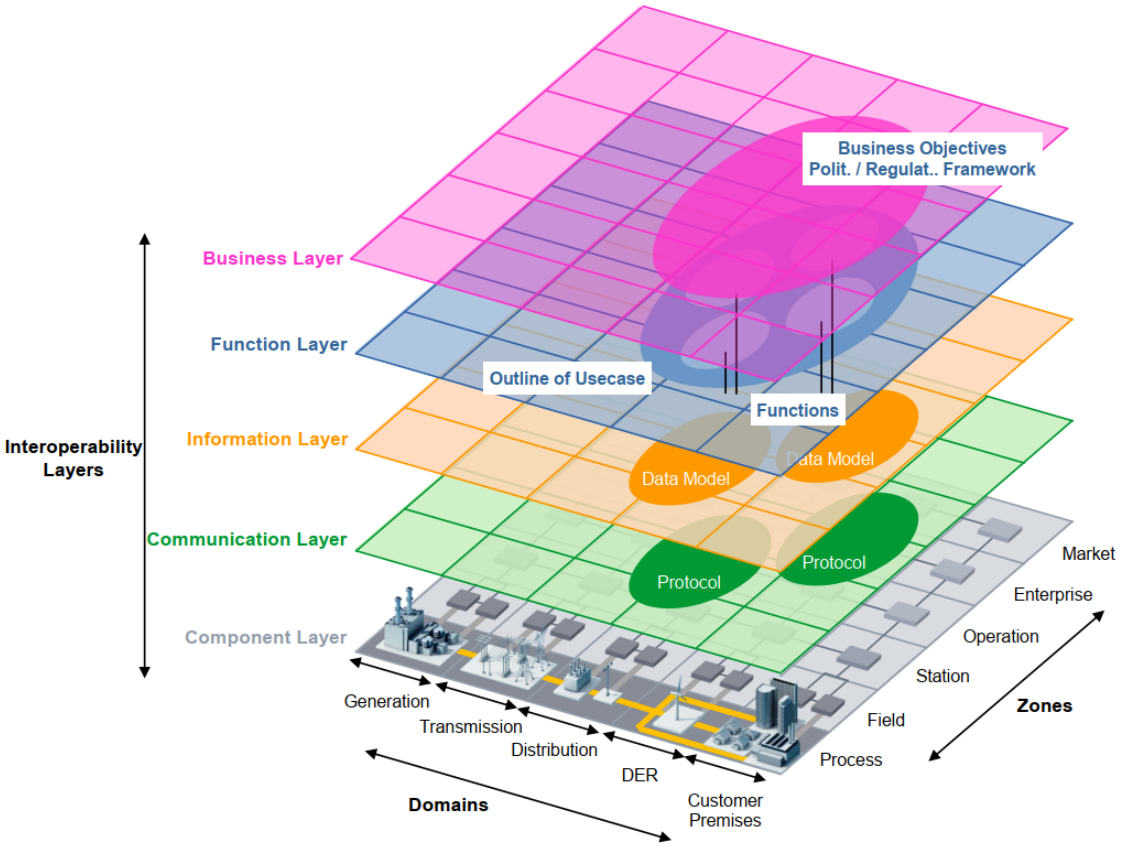
O modelo conceitual de *Smart Grids* fornecido pelo NIST (Figura 2.2) é utilizado por outros órgãos de padronização em todo o mundo como referência para elaborar normas, arquiteturas e *frameworks* relacionados a *Smart Grids*.

Na União Europeia, um dos principais esforços para a padronização de *Smart Grids* foi a instauração do "*M/490 Mandate for Standardization*" com o objetivo de garantir que

todos os órgãos de padronização estejam "na mesma página" em relação aos conceitos de alto nível de *Smart Grids* e, dessa forma, alcançar a interoperabilidade e facilitar a implementação de serviços e funcionalidades de *Smart Grids* na Europa (EC - European Commission, 2011).

O principal resultado da comissão M/490 foi a concepção do SGAM (*Smart Grid Architecture Model*) (ANDRÉN et al., 2017). O SGAM (Figura 2.3) é um framework que tem como objetivo padronizar a definição de casos de uso para sistemas, aplicações e *stakeholders* dentro do ambiente de *Smart Grids* utilizando um modelo contendo três dimensões: camadas de interoperabilidade, domínios e zonas (CEN-CENELEC-ETSI, 2012).

Figura 2.3 – Arquitetura definida pelo *framework* SGAM contendo as camadas de interoperabilidade, zonas e domínios de *Smart Grids*.



Fonte: (CEN-CENELEC-ETSI, 2012).

O SGAM se baseia nos conceitos de interoperabilidade estabelecidos no *framework GridWise*, no qual existem várias camadas de interoperabilidade (GWAC, 2008). O SGAM define cinco camadas de interoperabilidade (CEN-CENELEC-ETSI, 2012): *Business Layer* (Camada de Negócios), *Function Layer* (Camada de Funções), *Information Layer* (Camada de Informação), *Communication Layer* (Camada de Comunicação) e *Component Layer* (Camada de Componentes). O SGAM utiliza também os domínios definidos no modelo

conceitual do NIST (Figura 2.2) e inclui um domínio extra, que é o DER (*Distributed Energy Resources*). A terceira dimensão do SGAM trata das zonas ("*Zones*" na Figura 2.3), que representam os níveis hierárquicos de gerenciamento do sistema de potência. Do nível mais baixo ao nível mais alto de gerenciamento, temos (CEN-CENELEC-ETSI, 2012): *Process*, *Field*, *Station*, *Operation*, *Enterprise* e *Market*.

2.2 Normas IEC 61970-301 e 61968-11 (CIM - *Common Information Model*)

A norma IEC 61970-301 (IEC, 2016) consiste em um modelo semântico definido em UML que descreve os componentes elétricos do sistema de energia elétrica e as relações existentes entre esses componentes. A norma IEC 61968-11 (IEC, 2019) estende o modelo definido pela norma IEC 61970-301 para incluir outros aspectos dos sistemas de potência, como detalhes relacionados à medição e faturamento de consumidores (MCMORRAN, 2007). O modelo definido por essas duas normas é conhecido como *Common Information Model* - CIM (Modelo de Informação Comum, em português) (GUNGOR et al., 2011).

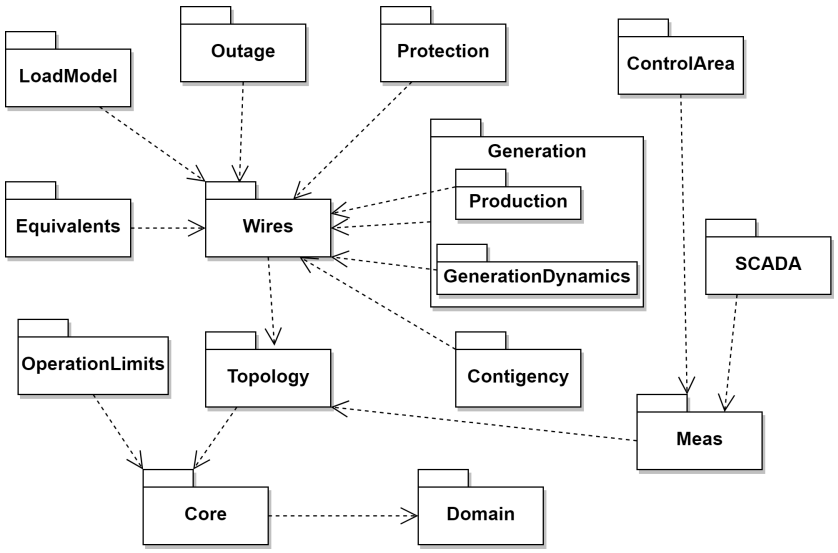
Sobre o escopo de aplicação do CIM, podemos afirmar que:

> O CIM tem a sua origem na modelagem de transmissão, principalmente na troca de dados entre sistemas EMS/SCADA e, portanto, o seu foco original foi em modelos operacionais de transmissão. A partir de então, o escopo do CIM foi expandido para além dos modelos de transmissão, com extensões para a modelagem de distribuição, mercados elétricos, planejamento do sistema, dinâmica do sistema, gerenciamento de recursos, gerenciamento de medidores, cobrança ao consumidor, gerenciamento de trabalho, compartilhamento de dados georreferenciados (GIS) e diagramas de *layouts* (MCMORRAN, 2007).

O CIM é um modelo hierárquico de dados organizado em pacotes, como é mostrado na Figura 2.4. Cada pacote contém classes que cumprem um determinado papel no sentido de representar diferentes aspectos dentro do sistema de energia elétrica. Os principais pacotes utilizados para construir modelos de topologias de redes elétricas e as medições de parâmetros da rede são os pacotes *Core*, *Wires*, *Topology* e *Meas*, que são descritos a seguir.

- **Pacote *Core***. O pacote *Core* contém a classe *PowerSystemResource*, a partir da qual todas as outras classes representando as propriedades físicas da rede elétrica herdam;
- **Pacote *Wires***. Contém todos os equipamentos eletricamente conectados à rede elétrica, como os Transformadores, os equipamentos de proteção, as linhas, as barras,

Figura 2.4 – Diagrama de pacotes da norma IEC 61970-301.



Fonte: (IEC, 2016).

os equipamentos de regulação e as configurações das instalações dos consumidores e dos produtores de energia elétrica;

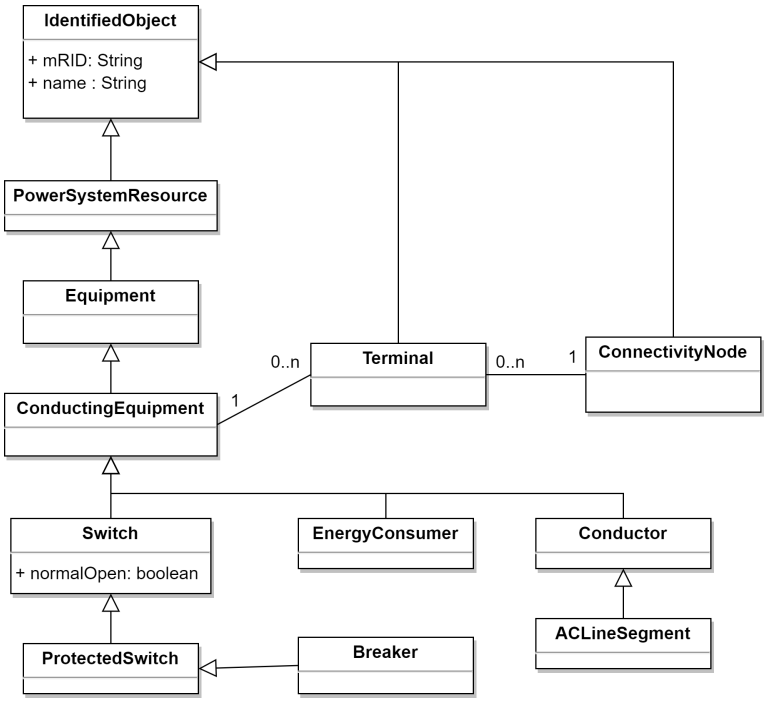
- **Pacote *Topology***. Define elementos que determinam como os equipamentos se conectam para formar as topologias de redes elétricas, através da classe *ConnectivityNode*;

- **Pacote *Meas***. Este pacote define os elementos que representam as medições feitas em relação a qualquer recurso do sistema de potência (representado pela classe *PowerSystemResource*). Estas medições podem ser de qualquer tipo de parâmetro, podendo ser valores analógicos (classe *Analog*) ou discretos (classe *Discrete*).

Uma das principais aplicações do CIM é a representação de topologias dos sistemas de potência, onde os equipamentos da rede são modelados, assim como a interconexão entre os equipamentos. Para definir as interconexões entre elementos do sistema de potência, o CIM utiliza as classes *Terminal* e *ConnectivityNode*. A classe *Terminal* consiste na interface de conexão elétrica de qualquer equipamento condutivo do modelo CIM. O *ConnectivityNode* representa uma conexão entre dois ou mais terminais para formar uma rede elétrica. A Figura 2.5 contém um fragmento do modelo CIM que representa a conectividade entre equipamentos, por exemplo entre um disjuntor (*Braker*), um consumidor de energia elétrica (*EnergyConsumer*) e um condutor de energia elétrica AC (*ACLineSegment*).

A classe *IdentifiedObject*, do pacote *Core*, serve como base para todas as classes dentro do CIM que precisam de alguma identificação. Essa identificação se dá pelos atributos *mRID*, que é um atributo que contém um identificador único para cada instância

Figura 2.5 – Fragmento do CIM, contendo o mecanismo utilizado para representar a conexão entre equipamentos.



Fonte: IEC (2016), McMorran (2007).

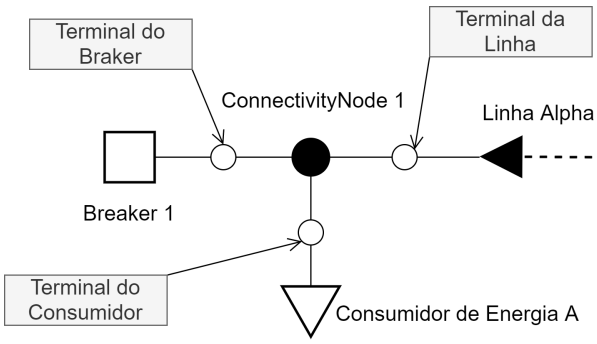
da classe, e o *name*, que pode ser utilizado para identificar uma instância com um nome legível ao ser humano.

Observa-se que a classe *ConductingEquipment* pode ter nenhum ou vários terminais. No entanto, um terminal só pode pertencer a um *ConductingEquipment*. A conexão entre equipamentos é feita por meio do *ConnectivityNode*, que pode fazer uma relação com vários terminais. Como as classes *Breaker*, *EnergyConsumer*, *Conductor*, *ACLineSegment*, *Switch* e *ProtectedSwitch* herdam de *ConductingEquipment*, então todas elas podem ser associadas com terminais que se conectam através de *ConnectivityNode*.

A partir das relações entre as classes definidas no fragmento do modelo CIM da Figura 2.5, o arranjo de topologia de rede da Figura 2.6 se torna possível, onde cada elemento do CIM é representado por uma simbologia tal qual a utilizada por McMorran (2007).

A topologia da Figura 2.6 representa uma conexão entre um *Breaker*, um consumidor de energia elétrica (*EnergyConsumer*) e um condudor do tipo *ACLineSegment*, ou seja, um segmento de linha. A conexão entre estes equipamentos é representada pela relação entre os seus respectivos terminais e um *ConnectivityNode*. Este modelo pode ser expandido para incluir outros equipamentos e instalações elétricas, como transformadores e subestações.

Figura 2.6 – Topologia simples de uma rede elétrica representada pelo CIM.



Fonte: McMorran (2007).

2.3 *Data Distribution Service* (DDS)

O DDS é uma especificação da OMG (*Object Management Group)* que define um *middleware* para conectividade centrada em dados baseada em um mecanismo de publicador-subscritor (*pub-sub*) (OMG, 2021). A especificação de DDS define o modelo DCPS (do inglês, "*Data-Centric Publish-Subscribe*"), que permite a comunicação e integração de aplicações através de uma API que pode ser implementada em qualquer linguagem de programação (OMG, 2015).

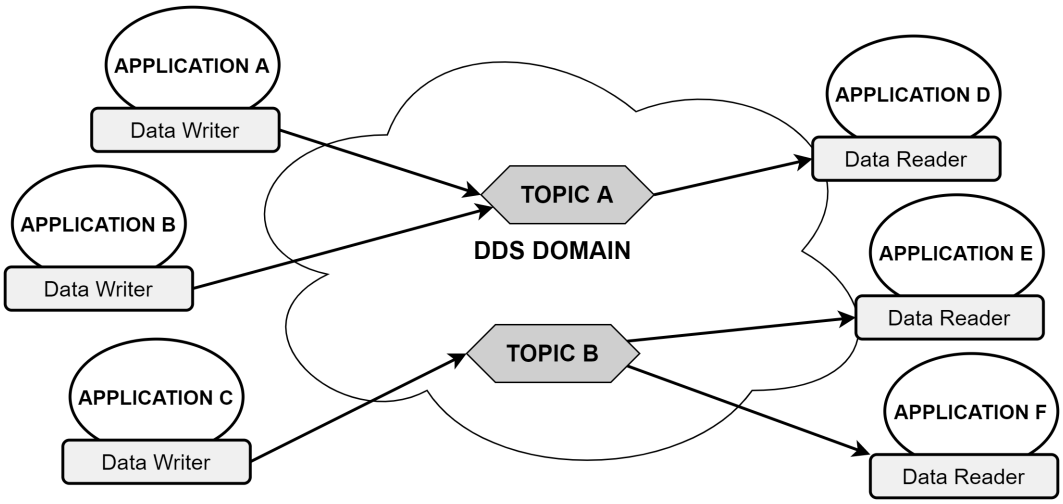
2.3.1 Principais características do DDS

O mecanismo de funcionamento do DDS é ilustrado na Figura 2.7. Os publicadores utilizam elementos chamados *Data Writers* para publicar dados em um determinado tópico do DDS. As aplicações que subscrevem aos tópicos utilizam elementos chamados *Data Readers* para receberem os dados que são publicados nos tópicos correspondentes. Um tópico é descrito pelo tipo de dado que ele representa e por um nome único dentro de um domínio. Isso quer dizer que o DDS possui informações sobre a estrutura dos dados compartilhados e as aplicações podem compartilhar objetos diretamente no DDS, sem necessitar de um formato intermediário, desde de que os tipos correspondentes estejam previamente registrados no *middleware*. Isso é chamado de *centricidade em dados* (ALABEYI, 2015; OMG, 2021).

Em um *middleware* centrado em dados, como o DDS, o próprio *middleware* é responsável por manter uma estrutura de dados na qual o estado dos dados compartilhados é mantido. É diferente, por exemplo, de um *middleware* centrado em mensagem, no qual a manutenção do estado dos dados compartilhados é de responsabilidade das aplicações (ALABEYI, 2015).

A seguir, algumas características relevantes do DDS são listadas e descritas (ALABEYI, 2015; OMG, 2021; OMG, 2015):

Figura 2.7 – Modelo de funcionamento do DDS. As aplicações se comunicam através da publicação e subscrição em tópicos dentro de um domínio de DDS.



Fonte: (OMG, 2021).

- **Suporte a QoS**: O DDS permite a utilização de QoS (*Quality of Service*) no compartilhamento de dados como confiabilidade, saúde do sistema (*liveliness*), durabilidade, histórico, limitação de recursos, *deadline* etc. O DDS gerencia e aplica as políticas de QoS automaticamente, necessitando das aplicações que apenas especifiquem quais políticas irão utilizar;
- **Escalabilidade**: A arquitetura do DDS é projetada para ser escalável e pode ser utilizada por milhares ou milhões de participantes, entregando dados em alta velocidade, gerenciando milhares de objetos, e fornecendo alta disponibilidade e segurança;
- **Descoberta dinâmica**: O DDS permite a descoberta dinâmica de publicadores e subscritores. Isso significa que as aplicações não precisam conhecer ou configurar os *endpoints* para as comunicações, já que os *endpoints* são descobertos automaticamente pelo DDS;
- **Segurança**: O DDS inclui mecanismos de segurança que fornecem autenticação, controle de acesso, confidencialidade e integridade;
- **Arquitetura distribuída de ponto-a-ponto**: O DDS utiliza uma arquitetura de comunicação de ponto-a-ponto. Isto é, não há um elemento centralizador responsável pela comunicação entre publicadores e subscritores, o que aumenta a robustez e flexibilidade do sistema.

2.3.2 *OpenDDS* - Uma implementação de DDS em código aberto

OpenDDS é uma implementação em C++ de código aberto de DDS, possuindo *binding* JNI (*Java Native Interface*) que permite a utilização de todas as suas funcionalidades através de uma API Java (OCI, 2020).

O *OpenDDS* implementa, atualmente (versão 3.16), três especificações da OMG (OCI, 2020):

1. **Data Distribution Service (DDS) for Real-Time Systems** (OMG, 2015) especifica as funcionalidades principais do DDS descritas na subseção anterior;
2. **The Real-Time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification (DDSI-RTPS) v2.3** (OMG, 2019): especifica um protocolo de interoperabilidade para DDS. O seu propósito é assegurar que aplicações baseadas em diferentes implementações de DDS possam de comunicar;
3. **DDS Security** (OMG, 2018): estende o DDS com funcionalidades de autenticação e criptografia.

2.4 *Model Driven Engineering* (MDE)

A Engenharia Dirigida por Modelos ou, em inglês, *Model Driven Engineering* (MDE), caracteriza um conjunto de abordagens para o desenvolvimento de *software* que têm surgido nas últimas duas décadas com o objetivo de suportar o gerenciamento da complexidade, aumentar a produtividade e confiabilidade, e diminuir custos de produção de *software*.

2.4.1 Contexto histórico

O gerenciamento da complexidade de desenvolvimento de *software* tem sido uma preocupação constante dentro da computação, desde o surgimento dos primeiros computadores digitais, no final da década de 1940 e início da década de 1950. Nessa época, o termo "engenharia de *software*" ainda não tinha um significado prático e, de fato, a computação era totalmente focada em *hardware* (BOEHM, 2006). A lógica utilizada para programar os computadores era feita por matemáticos ou engenheiros de *hardware* e a programação era feita feita diretamente no *hardware* através de códigos binários. A ideia que se tinha era de que *hardware* e *software* tinham essências parecidas. Alguns anos depois, as reais diferenças entre *hardware* e *software* começaram a ficar mais nítidas. O *software* é mais complexo, mais maleável, mais susceptível ao erro humano, não possui uma forma definida (por esta razão, é difícil dizer o quanto um *software* está próximo de ser finalizado), é intangível, possui muito mais estados e formas de testar em relação

ao *hardware* (BOEHM, 2006; BROOKS, 1987; MAHONEY, 2008), o que faz com que o *software* seja, por sua essência, difícil de projetar e manter.

No final da década de 1960, a falta de padronização e o tratamento da tarefa de programação como uma pura expressão da criatividade humana, levou à produção de sistemas de *software* praticamente impossíveis de modificar e manter devido ao chamado "código *spaghetti*". Essa situação ficou conhecida como a "crise de *software*", como foi descrita pela célebre carta de Edsger Dijkstra enviada à ACM em 1960, *Go to statement considered harmful* (DIJKSTRA, 1968) e, posteriormente, em 1972, em seu discurso de recebimento do Prêmio Turing, publicado pela ACM com o título *The Humble Programmer* (DIJKSTRA, 1972).

A partir da década de 1970, a realização de conferências e debates dentro da comunidade da computação possibilitou um melhor entendimento acerca do estado da prática do desenvolvimento de *software* e a proposta de melhorias para superar a crise de *software*. Estava claro, à época, que era necessário tratar a programação como uma disciplina de engenharia, com o desenvolvimento de padrões, métodos e processos para auxiliar o desenvolvimento de *software*. Nesse contexto, surgiu a engenharia de *software* (BOEHM, 2006).

A evolução da engenharia de *software* é marcada, por um lado, por avanços em métodos e padrões de desenvolvimento e, por outro lado, avanços em novas ferramentas, linguagens de programação, arquiteturas de *software* e novas plataformas. Na década de 1970, os métodos formais de engenharia de *software*, métodos estruturados (DAHL et al., 1972) e linguagens de programação estruturadas surgiram e ganharam popularidade, como a linguagem C (RITCHIE, 1993). Na década de 1980, podemos destacar o intenso emprego de reúso de *software*, o surgimento dos padrões de projeto (GAMMA et al., 1993) e das linguagens orientadas a objetos (RENTSCH, 1982), como Java (JOY et al., 2000) e C++ (WALLACE, 1993). O objetivo à época era melhorar a produtividade e a escalabilidade da engenharia de *software* (BOEHM, 2006), tendência que se manteve na década de 1990, com o aprimoramento nos processos de *software*, por exemplo, com a criação do *Rational Unified Process* (RUP), da IBM e Rational (KRUCHTEN, 2004), utilizado em muitos projetos de sucesso.

O início dos anos 2000, no entanto, apresentava um contexto no qual novas plataformas e tecnologias baseadas em serviços Web estavam surgindo muito rapidamente, jutamente com um mercado e ambiente de negócios cada vez mais dinâmicos. Com isso, houve uma frustração na indústria de *software* com os processos de *software* que exigiam muita documentação, planejamento e especificações (BOEHM, 2006). Não se conseguia alinhar uma boa documentação e especificação da lógica de negócios com os prazos cada vez mais curtos. Além disso, o ganho de produtividade que as linguagens de programação de terceira geração deram aos desenvolvedores na década de 1990 parecia ter se exaurido

e já não eram sufientes para lidar com a heterogeneidade das plataformas, padrões e *frameworks* emergentes.

Schmidt (2006) identifica o problema da disparidade entre a intenção de projeto dos desenvolvedores de *software* e a implementação do *software* nas plataformas finais. Os desenvolvedores de *software* geralmente expressam as mudanças de requisitos ou a necessidade de se adaptar a uma nova versão da plataforma diretamente no código. Porém, enquanto que as APIs das novas plataformas crescem em complexidade, as linguagens de programação não possuem capacidade de expansão de sua expressividade para acomodar tal aumento de complexidade. Nas palavras de Schmidt (2006): "*A falta de uma visão integrada do sistema leva os desenvolvedores a implementar soluções subótimas.*".

A Engenharia Dirigida por Modelos (MDE) ganhou força nesse contexto e tem sido proposta para abordar os desafios relacionados à complexidade de *software* e a mudança rápida das plataformas (SCHMIDT, 2006; BOOCH et al., 2004; BÉZIVIN, 2005). As abordagens para desenvolvimento de *software* baseadas em MDE buscam facilitar o fluxo de informação entre os diferentes artefatos no desenvolvimento de *software*, entre as fases de desenvolvimento (engenharia de requisitos, projeto, teste, implantação) e mesmo entre sistemas a partir da utilização de modelos definidos formalmente para representar: conceitos de domínio, lógica de negógio, plataformas e código (SIEGEL, 2014; BÉZIVIN, 2005).

2.4.2 Conceitos básicos no contexto de MDE

A ideia de MDE é apresentada por Schmidt (2006) como uma tentativa de construir uma ponte entre a especificação da arquitetura de *software*, os conceitos específicos de domínio e as plataformas de implementação: "*As tecnologias baseadas em MDE oferecem uma abordagem promissora para resolver o problema da inabilidade das linguagens de terceira geração para aliviar a complexidade das plataformas e expressar os conceitos de domínio de maneira eficiente*" (SCHMIDT, 2006).

Em MDE, todo o ciclo de desenvolvimento de *software* é centrado na definição, manipulação e manutenção de modelos (BÉZIVIN, 2005). Para entender melhor qual o significado que os modelos possuem dentro do contexto de MDE, alguns conceitos são fornecidos a seguir:

- **Abstração**: Dijkstra define abstração como "*A única ferramenta mental por meio da qual um raciocínio muito finito pode abranger uma infinidade de casos*" (DIJKSTRA, 1972). A utilização dos termos "finito" e "infinidade" nesta definição é bem adequada. Uma infinidade, ou seja, um conjunto muito grande de informações pode ser descrito ou representado por um conjunto bem pequeno (muito finito) de informações;
- **Modelo**: Utilizando a definição anterior, pode-se dizer que modelo é uma abstração

que é apresentada como uma forma, imagem ou esquema. No contexto de MDE, segundo Siegel (2014), um modelo é "*informação que representa seletivamente algum aspecto de um sistema baseada em um específico conjunto de interesses"*;

- **Metamodelo**: É uma linguagem utilizada para descrever modelos (BÉZIVIN, 2005);
- **Sistema**: É uma coleção de partes e relacionamentos entre estas partes que podem ser organizadas para atingir algum propósito (BÉZIVIN, 2005). O termo 'sistema' pode se referir a sistema de informação, mas é aplicado também de um modo mais geral. Portanto, um sistema pode incluir qualquer coisa, como *software*, pessoas, empresas, processo de negócio, computadores, sistemas embarcados etc (SIEGEL, 2014);
- **Plataforma**: É um conjunto de recursos no qual um sistema é implementado (SIEGEL, 2014). Ou seja, este conjunto de recursos é utilizado para suportar o sistema;
- **Arquitetura**: Nas palavras de Siegel (2014), "*O propósito da arquitetura é definir ou melhorar sistemas ou sistemas de sistemas. O processo arquitetural requer o entendimento do escopo do sistema de interesse, dos requisitos dos stakeholders, e chegar a um projeto que satisfaça esses requisitos*".

2.4.3 Princípios da MDE

Bézivin (2005) apresenta três princípios básicos da MDE. Combinados, estes três princípios definem o significado prático dos modelos dentro de abordagens baseadas em MDE.

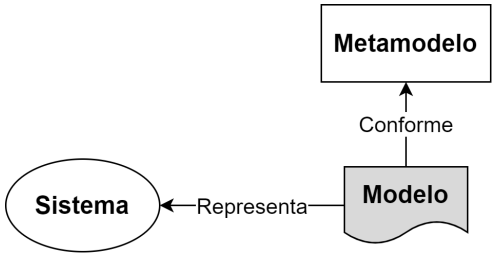
O primeiro princípio diz que *tudo é modelo*. Ou seja, todos os artefatos produzidos e utilizados durante o processo de desenvolvimento de *software* baseado em MDE são modelos, incluindo a especificação dos requisitos, a arquitetura do sistema, as configurações, as APIs utilizadas, a descrição das plataformas e o código fonte. Os próprios modelos são definidos por outros modelos, como veremos a seguir.

O segundo princípio básico da MDE apresentado por Bézivin (2005) é o *princípio da conformidade*, que diz que todo modelo deve ser conforme um *metamodelo*. O metamodelo é uma linguagem ou formalismo que descreve modelos. Logo, se diz que um modelo é conforme um metamodelo quando este metamodelo define todos os elementos contidos no modelo (BÉZIVIN, 2005).

O terceiro princípio apresentado por (BÉZIVIN, 2005) vai de encontro com o caráter de engenharia da MDE e diz que "*modelos representam sistemas*". Este é o *princípio da representação*.

A Figura 2.8 apresenta uma ilustração dos princípios apresentados. Em MDE, os sistemas são representados por essas abstrações chamadas de modelos, que, por sua vez, são definidos conforme metamodelos (BÉZIVIN, 2005; SIEGEL, 2014). Nota-se que os metamodelos também são modelos e, dessa forma, obedecem ao princípio de que *tudo é modelo*.

Figura 2.8 – Em MDE, modelos representam sistemas e são conforme metamodelos.



Fonte: (BÉZIVIN, 2005).

A definição formal de modelos permite que os sistemas computacionais possam processá-los, o que abre a possibilidade de manipular os modelos computacionalmente e aplicar vários tipos de operações aos diversos artefatos dentro do processo de desenvolvimento de *software*, como as transformações de modelo-a-modelo.

2.4.3.1 Linguagens específicas de domínio (DSLs)

Os metamodelos definem linguagens que descrevem modelos. Estas linguagens podem ser especificadas arbitrariamente (dentro das restrições do metametamodelo) para representar sistemas de *software* dentro de contextos específicos de domínio, representando os conceitos particulares, restrições, regulamentações e requisitos. Schmidt (2006) faz uma discussão sobre as DSLs (do inglês, *Domain Specific Languages*):

> As DLS formalizam a estrutura da aplicação, comportamento e requisitos dentro de domínios específicos, como rádios definidos por *software*, aviação, serviços financeiros, gerenciamento de depósito ou mesmo o domínio das plataformas de *middleware*. DSLs são descritas por metamodelos, que definem relações entre conceitos em um domínio e especifica precisamente a semântica e as restrições associadas a esses conceitos de domínio. Os desenvolvedores usam DSLs para criar aplicações utilizando elementos do sistema capturados por metamodelos e expressam a intenção de projeto de forma declarativa ao invés de imperativa (SCHMIDT, 2006).

As DSL podem auxiliar, portanto, na representação de três aspectos fundamentais dos sistemas de *software*:

- **Conceitos de domínio**: restrições, regulamentações, mercado, normas etc;

- **Intenção de projeto**: especificação da lógica de negócio e dos requisitos funcionais e não-funcionais;
- **Arquitetura de sistema**: definição de plataformas, tecnologias de comunicação e componentes de *software* a serem utilizados.

2.4.4 *Model Driven Architecture* (MDA)

Booch et al. (2004) define a Arquitetura Dirigida por Modelos (MDA - *Model Driven Architecture*) como um "*estilo de desenvolvimento e integração de aplicações empresariais, baseado no uso de ferramentas de automação para construir modelos independentes de plataforma e transformá-los em implementações eficientes*".

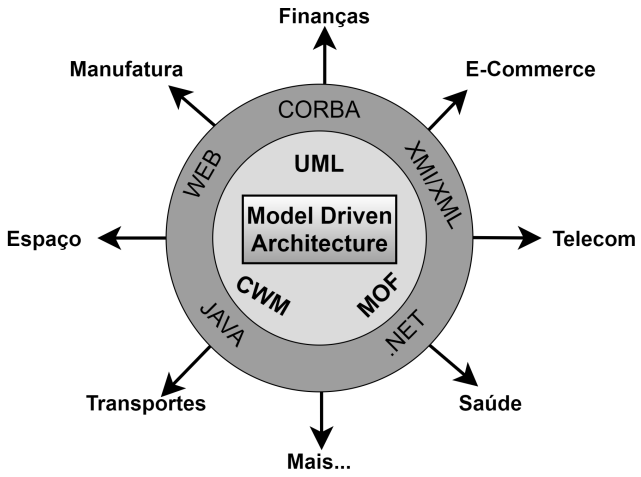
A MDA surgiu em um contexto, no início dos anos 2000, no qual as empresas tinham dificuldades para integrar as aplicações e tecnologias existentes e, ao mesmo tempo, acompanhar o surgimento de novas plataformas e a evolução das plataformas existentes (BOOCH et al., 2004). Várias especificações de *middleware* surgiram no final dos anos 1980 e na década de 1990 para prover interoperabilidade (SOLEY, 2000), como CORBA (OMG, 2012), XML/SOAP (BOX et al., 2000), EJB (ORACLE, 2010), COM+ (SCHOFIELD et al., 2018) e .NET (MICROSOFT, 2020). No entanto, a proliferação de especificações torna a integração de aplicações difícil e cara.

A OMG especificou a MDA com o intuito de fornecer uma arquitetura para integração e desenvolvimento de aplicações de forma independente de linguagem de programação, empresa ou *middleware* (SOLEY, 2000). A Figura 2.9 mostra a Arquitetura Dirigida por Modelos proposta pela OMG. A MDA propõe que as aplicações de *software* sejam modeladas utilizando os padrões da OMG, como UML, CWM (do inglês, *Common Warehouse Metamodel*) e MOF (centro da Figura 2.9), que são independentes da plataforma de implementação. Daí, os modelos independentes de plataforma da aplicação são convertidos para os modelos de plataformas específicas, como Web, CORBA, Java e .NET (anel externo da Figura 2.9) através de mapeamento entre modelos (SOLEY, 2000).

Em essência, os fundamentos da MDA consistem em três ideias complementares, apresentadas na Figura 2.10 (BOOCH et al., 2004):

1. **Representação direta**: Tira o foco do desenvolvimento de *software* do domínio da tecnologia de implementação, e traz o foco para ideias e conceitos do problema do domínio de aplicação;
2. **Automação**: Utiliza ferramentas computacionais para mecanizar aquelas faces do desenvolvimento de *software* que não dependem do talento humano. Um dos principais propósitos da automação em MDA é preencher a lacuna semântica entre os conceitos de domínio e as tecnologias de implementação através da modelagem

Figura 2.9 – A Arquitetura Dirigida por Modelos (MDA) proposta pela OMG.



Fonte: (SOLEY, 2000).

explícita do domínio e das escolhas da tecnologia em *frameworks* para então aplicar esse conhecimento em *frameworks* específicos da aplicação;

3. **Padrões abertos**: Os padrões têm sido um dos principais impulsionadores de progressos ao longo da história da tecnologia. Os padrões da indústria não apenas ajudam a eliminar divergências, mas encorajam um ecosistema de fabricantes que produzem ferramentas para propósito geral, ou nichos especializados, que aumentam muito a atratividade dessas ferramentas para os usuários. O desenvolvimento em código aberto assegura que os padrões são implementados de maneira consistente e encoraja a adoção de padrões por parte das empresas.

Figura 2.10 – Os fundamentos básicos da MDA: Padrões, Representação Direta e Automação.

Fonte: (BOOCH et al., 2004).

2.4.4.1 Definições básicas da MDA

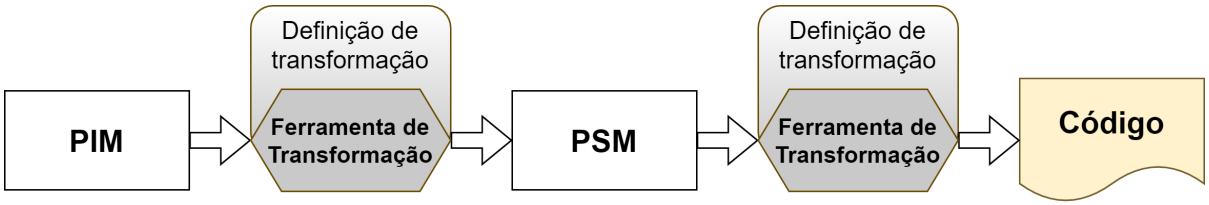
Tão logo a MDA foi definida, os trabalhos prosseguiram para desenvolver metodologias e ferramentas para aplicar a MDA ao processo de desenvolvimento de

software. Um dos conceitos mais importantes utilizado dentro de um *framework* MDA é o de independência de plataforma. Quando um modelo de um sistema é definido nos termos de uma plataforma específica, este modelo é chamado de "*Platform Specific Model*" (**PSM**), cuja tradução para o português é "Modelo Específico de Plataforma" (SIEGEL, 2014). Do contrário, quando um modelo de um sistema é independente de qualquer plataforma de implementação, este modelo é chamado "*Platform Independent Model*" (**PIM**), cuja tradução para o português é "Modelo Independente de Plataforma" (SIEGEL, 2014).

A MDA propõe a separação de preocupações dentro do processo de desenvolvimento a partir das definição de PIM e PSM. O PIM contém a lógica de negócio do sistema descrito. Já o PSM contém a modelagem do sistema considerando a implementação em uma plataforma ou tecnologia específica (KLEPPE et al., 2003). Há também um outro tipo de modelo em MDA, chamado de "*Computing Independent Model*" (**CIM**[3]), ou modelo independente de computação, que descreve apenas conceitos de lógica de negócios, sem fazer referência a qualquer arquitetura de sistema (SIEGEL, 2014).

O processo de passagem de informação entre os modelos definidos no contexto da MDA segue o esquema da Figura 2.11. O PIM é transformado de maneira automática ou semiautomática em um ou mais PSMs (SIEGEL, 2014). O código da aplicação, ou quaisquer outros artefatos de implementação de *software* são gerados a partir do PSM (BOOCH et al., 2004; KLEPPE et al., 2003).

Figura 2.11 – O processo básico de transformação entre os modelos definidos no contexto da MDA.



Fonte: (KLEPPE et al., 2003).

As ferramentas de transformação dão o caráter de automação para metodologias de desenvolvimento de *software* baseadas em MDA. Kleppe et al. (2003) define a **transformação entre modelos** como "*a produção de um modelo alvo a partir de um modelo fonte baseado em uma definição de transformação*" (KLEPPE et al., 2003). Ainda segundo Kleppe et al. (2003), **definição de transformação** "*é um conjunto de regras de transformação que descreve como um modelo em uma linguagem fonte pode ser transformado em um modelo descrito em uma linguagem alvo*" (KLEPPE et al., 2003). As transformações podem ser divididas em transformações de modelo-a-modelo, como é o caso da transformação entre o

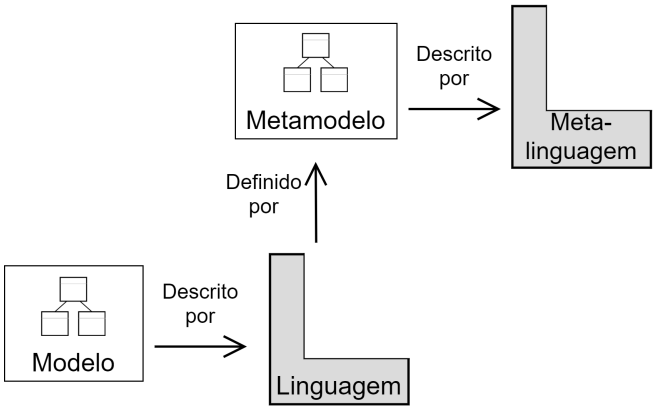
[3] O acrônimo *CIM* também é utilizado no contexto de *Smart Grids*, com outro significado (*Common Information Model*). No restante deste manuscrito, o acrônimo *CIM* será utilizado apenas no contexto de *Smart Grids*.

PIM e o PSM na Figura 2.11, e em transformações de modelo-a-código, como é o caso da transformação entre o PSM e o código na Figura 2.11 (KLEPPE et al., 2003).

2.4.4.2 Arquitetura de quatro níveis e o *framework* MDA

No contexto da MDA, um modelo é a descrição de um sistema (ou parte de um sistema) em uma linguagem bem definida. O mecanismo que se utiliza para definir linguagens de modelagem é a *metamodelagem* (KLEPPE et al., 2003; BÉZIVIN, 2005). A Figura 2.12 ilustra este mecanismo.

Figura 2.12 – As relações entre modelos, linguagens, metamodelos e metalinguagens.



Fonte: (KLEPPE et al., 2003).

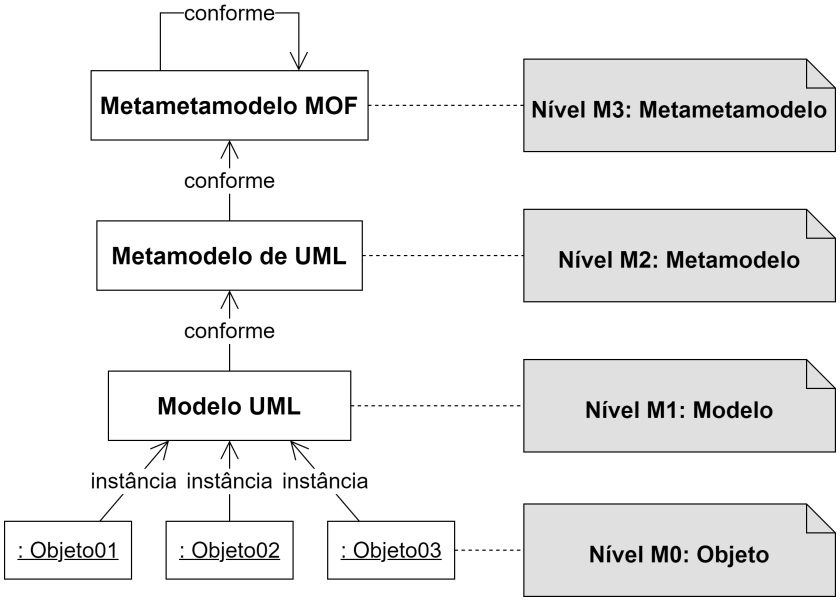
Uma linguagem de modelagem define quais elementos podem existir em um determinado modelo e é definida por um metamodelo. Segundo Kleppe et al. (2003), para todos os fins práticos, um metamodelo é uma linguagem de modelagem. O metamodelo é também um modelo e, portanto, é descrito por uma linguagem, que é chamada de *metalinguagem* (KLEPPE et al., 2003).

Teoricamente, infinitas camadas de relações entre linguagem de modelagem e metalinguagem podem existir. Ou seja, uma metalinguagem é descrita por uma outra linguagem que é descrita por uma outra linguagem e assim por diante. No entanto, os padrões da OMG contemplam quatro níveis (BÉZIVIN, 2005), como ilustrado na Figura 2.13.

Na Figura 2.13, o metamodelo de UML é utilizado como exemplo, porém a arquitetura de quatro níveis se aplica a *qualquer metamodelo* utilizado dentro do contexto da MDA. A seguir, uma rápida descrição de cada nível da arquitetura de quatro níveis ilustrada na Figura 2.13 é fornecida:

- **Nível M0**: Compreende os objetos que são instanciados a partir dos modelos do nível M1;

Figura 2.13 – A arquitetura de quatro níveis.



Fonte: (BÉZIVIN, 2005).

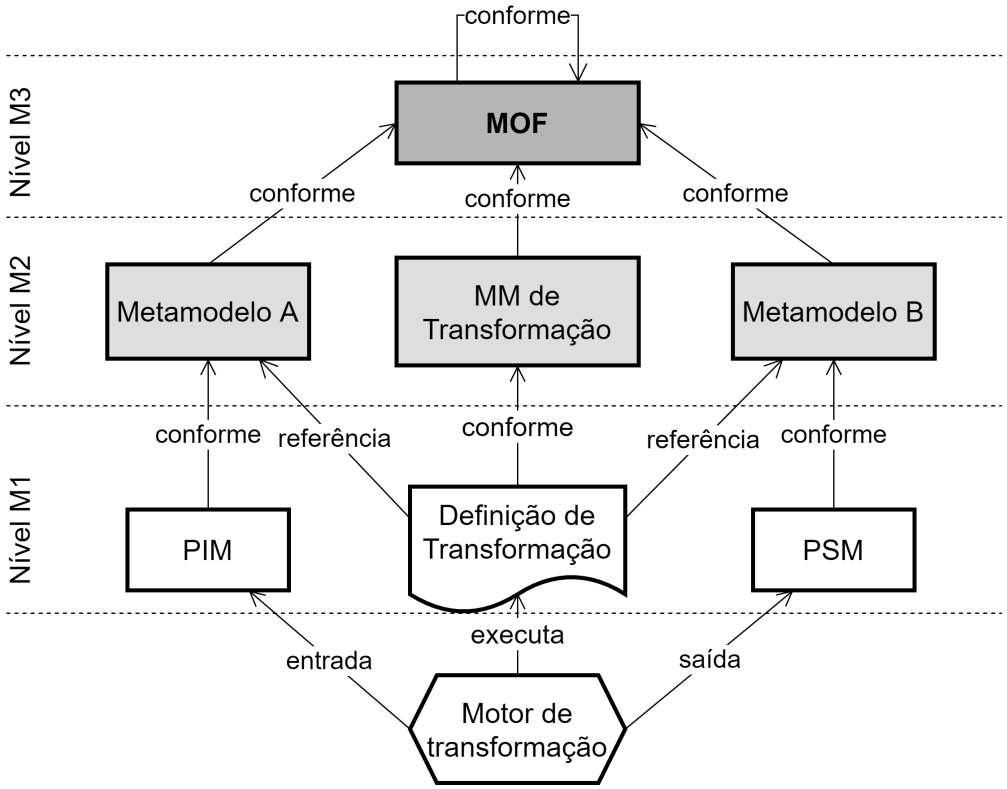
- **Nível M1**: Compreende os modelos que são conforme um metamodelo definido no nível M2;
- **Nível M2**: Compreende os metamodelos que definem os modelos do nível M1. Logo, os modelos do nível M1 são instâncias dos metamodelos do nível M2;
- **Nível M3**: Neste nível é definido o *metametamodelo*, que define uma linguagem de metamodelagem e descreve os metamodelos do nível M2. Logo, os metamodelos do nível M2 são instâncias do metametamodelo do nível M3.

2.4.4.3 O *framework* básico da MDA

A Figura 2.14 contém o *framework* básico da MDA tal como é definido em Kleppe et al. (2003). Este *framework* reúne os conceitos de PIM, PSM e transformações entre modelos em uma arquitetura de níveis. O PIM é conforme o *Metamodelo A*, que poderia ser o metamodelo de UML ou alguma DSL. O PSM é conforme o *Metamodelo B*, que poderia ser um metamodelo de CORBA, .NET, DDS, Java ou outra tecnologia de implementação. O motor de transformação executa a definição de transformação, que é conforme o metamodelo de transformação (*MM de Transformação*). A definição de transformação especifica quais elementos do PIM serão transformados em quais elementos do PSM e, para isso, ela contém referências aos elementos do *Metamodelo A* e aos elementos do *Metamodelo B*, respectivamente.

Todos os metamodelos definidos no *framework* MDA da Figura 2.14 são conforme o metametamodelo MOF. O MOF, portanto, fornece uma base semântica comum entre

Figura 2.14 – O *framework* básico da MDA.



Fonte: (KLEPPE et al., 2003).

para todas as linguagens utilizadas dentro deste *framework*, inclusive o próprio MOF.

2.4.4.4 Padrões definidos no contexto de MDA

A OMG recomenda que a MDA seja implementada em torno de padrões abertos para a representação, armazenamento, manipulação e transformação de modelos. A OMG especifica e mantém padrões para permitir que diferentes ferramentas de MDA sejam interoperáveis e para que os modelos possam ser mantidos sem perder o seu valor. Alguns dos padrões definidos pela OMG são apresentados a seguir.

Unified Modeling Language - UML

A UML é uma especificação da OMG que define uma linguagem gráfica para visualizar, especificar, construir e documentar artefatos de sistemas distribuídos baseados em objetos. A UML é constituída por um conjunto de diagramas que permitem modelar a estrutura e o comportamento de sistemas de *software* utilizando os conceitos da orientação à objetos (OMG, 2017).

Dentro do contexto da MDA, a UML é vista como uma linguagem de modelagem com uma sintaxe bem definida. A sintaxe abstrata de UML é definida em um metamodelo, conforme MOF (OMG, 2017).

MetaObject Facility - MOF

O MOF é um padrão da OMG em torno do qual os outros padrões no contexto da MDA são definidos. O MOF define uma metalinguagem que fornece meios para a definição de metamodelos (OMG, 2019). Esta metalinguagem corresponde ao nível M3 da arquitetura de quatro níveis, e é baseada em um subconjunto da UML (OMG, 2019).

XML Metadata Interchange - XMI

O XMI é um padrão da OMG para a representação de informações, modelos e metamodelos baseado em XML. O XMI define os seguintes aspectos em relação à representação de objetos utilizando XML (OMG, 2015):

- A representação de objetos em termos de elementos e atributos de XML;
- Um mecanismo padrão para fazer o link entre os objetos dentro de um mesmo arquivo ou entre arquivos;
- A validação de documentos XMI utilizando XML *Schemas*;
- A identidade de objetos, o que permite que os objetos sejam referenciados por outros objetos em termos de IDs e UUIDs.

Object Constraint Language - OCL

OCL é uma linguagem formal utilizada para escrever expressões em modelos UML. Estas expressões são utilizadas para impor restrições aos elementos de modelos, como invariantes ou pré-condições. A OCL pode ser utilizada também para (OMG, 2014):

- Especificar *queries*;
- Especificar invariantes em classes e tipos em modelos;
- Especificar pré- e pós-condições em operações e métodos;
- Descrever guardas;
- Especificar restrições e operações;
- Especificar regras derivadas para atributos de qualquer expressão em um modelo UML.

Query, View and Transformation - QVT

QVT é uma especificação da OMG para transformações, *queries* e *views* de modelos, como o nome indica. O padrão QVT especifica três linguagens de transformação de modelos (OMG, 2016):

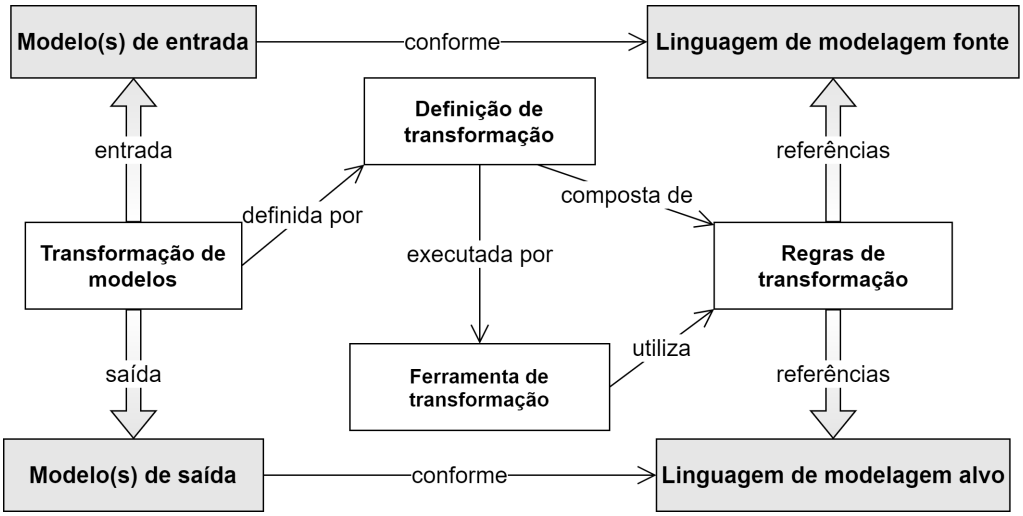
- **QVT-Operacional**: é uma linguagem imperativa projetada para escrever transformações unidirecionais;
- **QVT-Relational**: é uma linguagem declarativa projetada para permitir tanto transformações de modelos unidirecionais como bidirecionais;
- **QVT-Core**: é uma linguagem declarativa tão poderosa quanto a *QVT-Relational*, só que mais simples. *QVT-Core* pode ser implementada diretamente, ou pode ser usada como referência para a semântica das Relações, definidas em *QVT-Relational*, as quais são mapeadas para *QVT-Core*.

2.4.5 Transformação de modelos

Como já discutido na Subsubseção 2.4.4.1, onde as definições básicas da MDA são apresentadas, a transformação entre modelos é uma operação fundamental no contexto de MDE (JOUAULT et al., 2008), pois fornece automação e correção ao processo de geração de código, arquivos de configuração e modelos em diferentes níveis de abstração. Além disso, como aponta Schmidt (2006), as ferramentas de transformação ajudam a garantir a consistência e traçabilidade entre os modelos de análise de requisitos e lógica de negócio e a implementação das aplicações. Kleppe et al. (2003) define alguns termos relacionados à transformação de modelos. Estes termos são colocados em forma de mapa mental, apresentado na Figura 2.15, e são definidos a seguir:

- **Transformação de modelos**: é a produção de um ou vários modelos alvo a partir de um ou mais modelos fonte baseada em uma definição de transformação (KLEPPE et al., 2003);
- **Definição de transformação**: é um conjunto de regras de transformação que descreve como um modelo em uma linguagem fonte pode ser transformado em um modelo em uma linguagem alvo (KLEPPE et al., 2003);
- **Regra de transformação**: descrição de como uma ou mais construções na linguagem fonte podem ser transformadas em uma ou mais construções da linguagem alvo (KLEPPE et al., 2003);

Figura 2.15 – Mapa mental contendo os conceitos básicos relacionados a transformações de modelos.



Fonte: (KLEPPE et al., 2003).

- **Ferramenta de transformação**: também chamado de motor de transformação, é um *software* que é capaz de executar uma definição de transformação (KLEPPE et al., 2003).

2.4.5.1 Classificação de abordagens de transformação de modelos

Czarnecki e Helsen (2003) propõem uma taxonomia para a classificação de abordagens para transformação de modelos. A partir da análise de várias abordagens de transformação de modelos disponíveis na literatura, os autores classificaram estas abordagens em cinco categorias, que são brevemente descritas a seguir (CZARNECKI; HELSEN, 2003):

- **Abordagens de manipulação direta**: estas abordagens fornecem uma representação interna dos modelos e uma API para manipulá-los. Os usuários devem implementar as regras de transformação utilizando uma linguagem de programação, por exemplo, Java;

- **Abordagens relacionais**: esta categoria agrupa abordagens declarativas onde o principal conceito utilizado são relações matemáticas. A ideia básica é definir o elemento fonte e o elemento alvo de uma relação e especificar esta relação utilizando restrições formais;

- **Abordagens baseadas em grafos**: as regras de transformação em abordagens baseadas em grafos consistem em um padrão de grafo da esquerda - LHS (do ingês, *Left-Hand Side*) e um padrão de grafo da direita - RHS (do inglês, *Right-Hand Side*). O padrão LHS é transformado e substituído pelo padrão RHS;

- **Abordagens orientadas à estrutura**: as abordagens que se encaixam nesta categoria possuem duas fases distintas: a primeira fase é focada na criação de uma estrutura hieráquica do modelo alvo; a segunda fase se encarrega de definir os atributos e referências no modelo alvo;

- **Abordagens híbridas**: As abordagens híbridas combinam diferentes técnicas das categorias anteriores. A linguagem ATL, por exemplo, é uma abordagem híbrida. Uma regra de transformação em ATL pode ser completamente declarativa, híbrida ou completamente imperativa.

2.4.5.2 Características desejáveis de transformações de modelos

Existem algumas características que são desejáveis em abordagens e ferramentas para a transformação de modelos. Kleppe et al. (2003) citam as mais relevantes:

- **Tunabilidade**: significa que, embora uma regra de transformação tenha sido especificada na definição de transformação, a aplicação desta regra pode ser ajustada;

- **Traçabilidade**: significa que é possível rastrear um elemento do modelo alvo de volta ao elemento no modelo fonte a partir do qual ele foi gerado;

- **Consistência Incremental**: significa que uma informação específica que foi adicionada ao modelo alvo permanece consistente mesmo após a regeneração deste modelo;

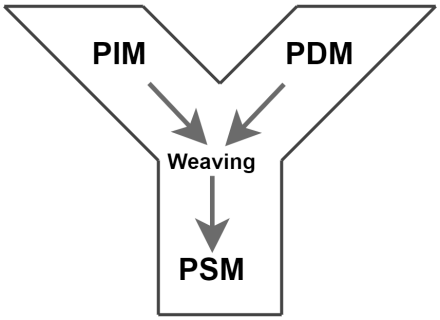
- **Bidirecionalidade**: significa que uma transformação pode ocorrer em dois sentidos, ou seja, a transformação pode ser aplicada também para gerar o modelo fonte a partir do modelo alvo.

2.4.6 Desenvolvimento baseado em Y e *weaving* de modelos

O desenvolvimento baseado em Y é um guia metodológico para desenvolvimento de *software* baseado em MDA que tem sido proposto na literatura (BELANGOUR et al., 2006; FABRO et al., 2005). A Figura 2.16 mostra o esquema básico do desenvolvimento baseado em Y.

Belangour et al. (2006) propõe que as características específicas da plataforma sejam descritas em um modelo separado do PIM, chamado de *Platform Description Model* - PDM (Modelo de Descrição de Plataforma). Desta forma, o ramo esquerdo do esquema da Figura 2.16 corresponde ao PIM, que contém a lógica de negócio e a arquitetura de *software*. O PDM contém as características e especificações (por exemplo, os tipos de dados utilizados) da plataforma de destino (BELANGOUR et al., 2006; FABRO et al., 2005; FABRO et al., 2006). O elementos do PIM são combinados com os elementos do PDM de

Figura 2.16 – Esquema básico do desenvolvimento baseado em Y.



Fonte: (BELANGOUR et al., 2006).

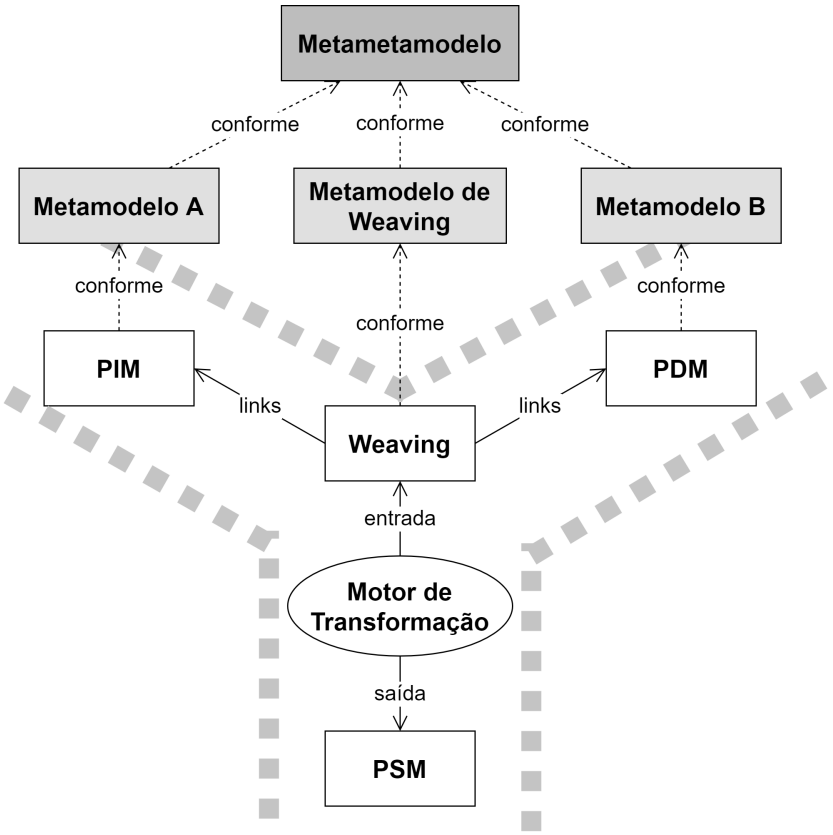
acordo com as decisões de projeto tomadas pelo especialista de domínio, por exemplo, qual tipo de dado específico da plataforma (elemento do PDM) corresponde a uma determinada informação do PIM (BELANGOUR et al., 2006). Dessa forma, relações entre os elementos do PIM e do PDM são criadas, e o PSM é gerado a partir destas relações. No entanto, estas relações entre PIM e PDM necessitam ser representadas de alguma forma.

Conforme esclarece Fabro et al. (2006), as transformações de modelo-a-modelo não são suficientes para capturar relações entre modelos de forma eficiente. Desta forma, Belangour et al. (2006) propõe que as relações entre o PIM e o PDM, dentro do desenvolvimento baseado em Y, sejam capturadas por uma operação chamada *weaving* de modelos (BELANGOUR et al., 2006; FABRO et al., 2005).

O *weaving* (entrelaçamento, em português) de modelos corresponde à especificação de *links* entre os elementos de dois ou mais modelos (FABRO et al., 2005; FABRO et al., 2006), podendo ser utilizado em várias aplicações, como o *merging* de modelos (CARVALHO et al., 2015), composição de modelos (BÉZIVIN et al., 2006) e traçabilidade (JOUAULT, 2005).

O *weaving* de modelos tem sido utilizado dentro de metodologias de desenvolvimento de *software* baseadas em Y para relacionar os elementos do PIM e do PDM. Em alguns trabalhos, como os de Junior (2017) e Stefanello (2017), PIM, PDM e PSM são organizados em uma arquitetura semelhante à da Figura 2.17. Nesta arquitetura, o *weaving* é um modelo conforme o *Metamodelo de Weaving* que contém links entre o PIM e o PDM. PIM e PDM são conforme os seus respectivos metamodelos e todos os metamodelos são conforme um metametamodelo em comum. O motor de transformação tem como entrada o PIM, PDM e o modelo de *Weaving*, e tem como saída o PSM. Portanto, PIM e PDM são entrelaçados com o auxílio do *weaving* de modelos para gerar o PSM.

Figura 2.17 – Uma arquitetura para o desenvolvimento de *software* baseado em Y com *weaving de modelos*.



Fonte: Junior (2017) e Stefanello (2017).

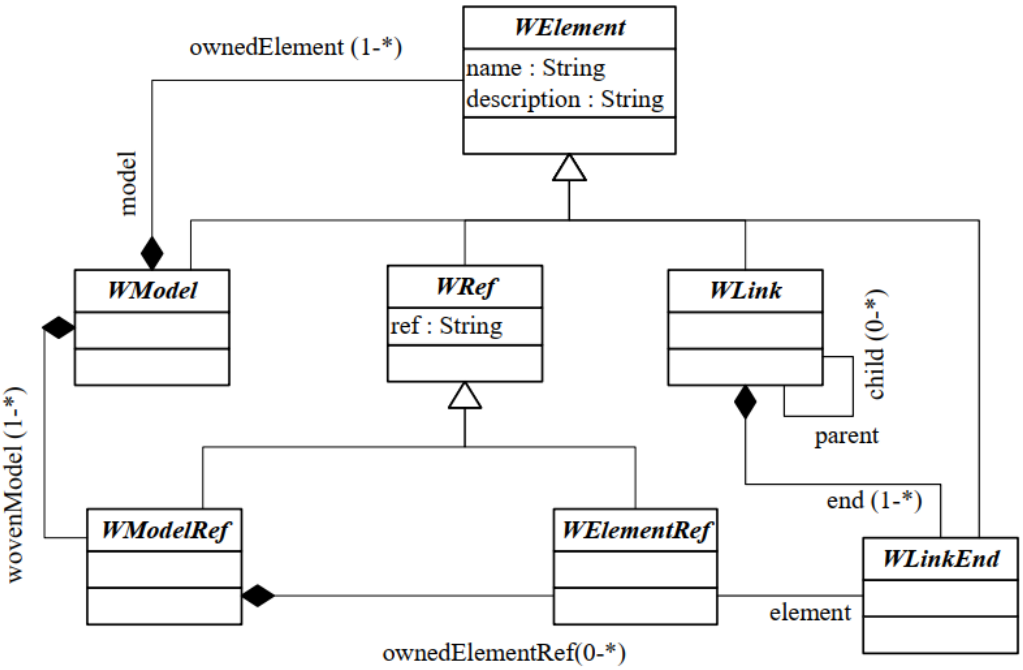
2.4.6.1 Metamodelo de *weaving*

Dentro do contexto de MDE, em que a suposição básica é de que tudo é modelo (BÉZIVIN, 2005), a especificação de *weaving* entre modelos é também um modelo que é conforme um metamodelo. Fabro et al. (2006) propõe um metamodelo de *weaving* básico, conforme ilustra a Figura 2.18. Os elementos deste metamodelo são descritos a seguir:

- *WElement* é o elemento base a partir do qual todos os outros elementos herdam os atributos *name* e *description*;
- *WModel* representa o elemento raiz que contém todos os elementos do modelo de *weaving* e contém também referências para os modelos que estão sendo entrelaçados;
- *WLink* expressa uma conexão entre elementos dos modelos. Para dar um significado mais específico para este elemento, ele deve ser extendido;
- *WLinkEnd* expressa um ponto de conexão de um *WLink*;
- *WElementRef* contém uma referência a um elemento de modelo;

- *WModelRef* é similar ao *WElementRef*, mas faz referência a um modelo completo, e não um elemento particular.

Figura 2.18 – Metamodelo básico de *weaving*.



Fonte: (FABRO et al., 2006).

Podemos então ler o metamodelo de *weaving* da Figura 2.18 da seguinte forma: o modelo de *weaving* (*WModel*) contém referências para os modelos que estão sendo entrelaçados (*WModelRef*) que, por sua vez, contêm referências para os seus elementos (*WElementRef*). O modelo de *weaving* (*WModel*) possui também links (*WLink*) cujas terminações ou conexões (*WLinkEnd*) fazem referência aos elementos dos modelos (*WElementRef*) que estão sendo entrelaçados.

O metamodelo de *weaving* da Figura 2.18 possui apenas tipos abstratos. A ideia proposta por Fabro et al. (2006) é que este metamodelo seja extendido para se adaptar aos diferentes casos de uso, já que os elementos do metamodelo de *weaving*, como *WLink* e *WLinkEnd*, podem ter diferentes significados, a depender da aplicação (FABRO et al., 2006).

2.4.7 *Eclipse Modeling Framework* (EMF)

O EMF é um *framework* baseado em estrutura hieráquica de modelos para a geração de código e construção de ferramentas e aplicações. O EMF consiste em três partes fundamentais (ECLIPSE, 2020):

- **EMF Core**: O núcleo do EMF inclui uma linguagem de modelagem chamada Ecore para descrever modelos, suporte em tempo de execução para os modelos, incluindo notificação de mudanças, serialização dos modelos em XMI e uma API Java para manipular os objetos em EMF de forma genérica;
- **EMF.Edit**: O framework *EMF.Edit* inclui classes generéricas reusáveis para construir editores para os modelos baseados em EMF;
- **EMF.Codegen**: o mecanismo de geração de código incluída no EMF é capaz de gerar tudo o que é necessário para se construir um editor completo para modelos em EMF.

Os modelos em EMF são construídos conforme uma linguagem de metamodelagem chamada Ecore. O Ecore assume um papel no EMF que é análogo ao papel desempenhado pelo MOF no contexto da MDA. Na arquitetura de 4 camadas (Figura 2.13), o Ecore é colocado na camada M3, assim como o MOF (BÉZIVIN, 2005).

Segundo Bézivin (2005), os objetivos do EMF e da MDA estão alinhados e, inclusive, a MDA pode ser vista como um conjunto de padrões, enquanto o EMF consiste em uma implementação baseada nos mesmos padrões.

2.5 Síntese

Este capítulo apresentou uma revisão bibliográfica dos principais conceitos e tecnologias utilizadas durante o desenvolvimento deste trabalho de pesquisa.

O conceito de *Smart Grids* foi explorado levantando a problemática da sustentabilidade da matriz elétrica no Brasil e no mundo. Devido às limitações dos recursos naturais e dos impactos ambientais causados pela produção de energia elétrica a partir de fontes não-renováveis de energia e, especialmente no caso do Brasil, dos frequentes problemas de estiagem nas bacias hidrográficas que condicionam o funcionamento adequado das hidrelétricas, há a uma crescente necessidade de se diversificar a matriz energética em todo o mundo, especialmente com a adição de novas fontes renováveis como a energia solar e a energia eólica.

Neste contexto, devido às características de variabilidade no tempo e da incerteza da produção de energia elétrica em determinados períodos destas novas fontes renováveis, além da crescente participação da geração distribuída, microgeração e penetração dos veículos elétricos à rede elétrica, será cada vez mais necessária a utilização de uma automação avançada no sistema de potência para que a energia elétrica possa continuar a ser entregue ao consumidor e com ainda mais qualidade.

Smart Grids é o termo utilizado para descrever a integração de tecnologias avançadas de informação, comunicação, sensoriamento e controle ao sistema de potência para criar um ambiente integrado de geração e consumo de energia elétrica que favorece o surgimento de novas oportunidades de negócio e maior participação dos consumidores.

Os padrões que surgiram para guiar o desenvolvimento de sistemas de *software* para *Smart Grids* foram descritos, como é o caso do SGAM, e a norma CIM para sistemas de potência, que descreve um modelo padrão para a representação de topologias de redes elétricas e equipamentos foi apresentada.

O *middleware* DDS (*Data Distribution Service*) foi apresentado. O DDS é uma especificação da OMG que define o modelo DCPS (*Data-Centric Publish Subscribe*) que permite a comunicação e integração de aplicações via um *middleware* publicador-subscritor centrado em dados. O OpenDDS, que é uma implementação *opensource* de DDS foi descrito.

Por fim, a Engenharia Dirigida por Modelos (MDE) foi apresentada. Uma contextualização histórica sobre a Engenharia de *Software* foi fornecida, dando ênfase às tentativas de gerenciar a complexidade de *software* ao longo dos últimos 50 anos.

A MDE surge com mais força no início dos anos 2000 como um paradigma dentro da Engenharia de *Software* para gerenciar a complexidade considerando o modelo como o principal artefato de desenvolvimento. Ainda no início dos 2000 a OMG especifica a Arquitetura Dirigida por Modelos, que é um *framework* baseado nos princípios da MDE para suportar o gerenciamento dos artefatos de *software*. A MDA permite a separação de preocupações ao definir o modelo independente de plataforma (PIM), que contém a lógica de negócio do *software*, e o modelo específico de plataforma (PSM), que contém a especificação do *software* para uma plataforma específica, como DDS, Java, Android, .NET, CORBA, *Web Services* etc.

As transformações entre modelos permitem que o PIM possa ser transformado em PSM automaticamente e que o código fonte das aplicações seja gerado (pelo menos em parte) a partir do PSM.

No avanço das teorias e tecnologias no contexto da MDA, a técnica de *weaving* (entrelaçamento entre modelos) de modelos é apresentada. A partir da técnica de *weaving* surge a possibilidade de se implementar um outro estilo de gerenciamento de modelos a partir da MDA, que é o M2T (*MDA 2 Tracks*), ou desenvolvimento baseado em Y. O M2T separa a definição do PIM da definição dos modelos de descrição de plataforma (PDM), que são modelos que representam aspectos específicos do *software*, como aspectos de segurança, definição de *datatypes*, aspectos de sistemas distribuídos, entre outros aspectos.

No M2T, PIM e PDM são entrelaçados através da técnica de *weaving*. O PIM, o PDM e o *weaving* são os modelos de entrada de uma transformação de modelos que

tem como saída o PSM abstrato. A partir do PSM abstrato, o PSM concreto é gerado e, posteriormente, o código fonte pode ser gerado assim como os arquivos de configuração da plataforma alvo.

3 Estado da Arte

Este capítulo contém o levantamento e a análise dos trabalhos mais recentes e relevantes na literatura que discorrem sobre abordagens para prover interoperabilidade e integração de aplicações em *Smart Grids*. A Seção 3.1 descreve o procedimento utilizado para realizar o levantamento das pesquisas analisadas. No total, seis trabalhos foram selecionados para compor o Estado da Arte apresentado neste capítulo. Na Seção 3.2, estes trabalhos são apresentados. Na Seção 3.3, as pesquisas selecionadas são analisadas e os critérios para a comparação entre os trabalhos do Estado da Arte e a abordagem proposta neste manuscrito são apresentados. Uma tabela é apresentada no final do capítulo em que cada trabalho do Estado da Arte é avaliado de acordo com os critérios elaborados.

3.1 Descrição do processo de levantamento do Estado da Arte

A seleção dos trabalhos incluídos no Estado da Arte deu-se a partir de buscas em bases digitais de artigos como *IEEE Xplore*, *ACM Digital Library*, *Science Direct*, além de buscas realizadas através do *Google Scholar*. Não se fez uso de uma metodologia específica de busca e seleção. O artigos foram selecionados a partir de análise empírica dos resultados das buscas nessas bases digitais. A seguir, os termos comumente utilizados para busca nessas bases são fornecidos:

- **“model driven” AND “smart grids” AND “applications”**;
- **“smart grid” AND (“interoperability” OR “integration”)**;
- **(“power system” OR “smart grid”) AND “model”**;
- **“smart grid” AND “middleware”**;
- **“smart grid” AND “cim” AND “integration”**.

Os resultados das buscas foram filtrados com base no ano em que as pesquisas foram realizadas. Só foram considerados para compor o Estado da Arte trabalhos publicados entre os anos de 2014 e 2019. O outro critério utilizado para a seleção dos trabalhos foi o objetivo das pesquisas. Os trabalhos selecionados são aqueles que propõem aplicar a MDE para auxiliar o desenvolvimento de aplicações de *Smart Grids* ou que, mesmo não aplicando diretamente os métodos da MDE, utilizam algum modelo, padrão, norma ou arquitetura de *software* para integrar aplicações em *Smart Grids* e garantir a interoperabilidade.

3.2 Abordagens para o desenvolvimento e integração de aplicações em *Smart Grids*

Nesta seção, seis trabalhos relevantes para o campo da engenharia de *software* e integração de aplicações no domínio de *Smart Grids* são apresentados.

3.2.1 *Model-driven engineering applied to Smart Grid automation using IEC 61850 and IEC 61499* (ANDRÉN et al., 2014)

Andrén et al. (2014) afirmam que abordagens formais compatíveis com normas para a modelagem da automação dos sistemas de potência e aplicações de controle estão escassas atualmente. Uma das soluções mais promissoras para dar suporte à troca de dados entre aplicações de *Smart Grids* são aquelas baseadas no padrão IEC 61850, que lida com a automação dos sistemas de potência. No entanto, o padrão IEC 61850 não faz a representação da implementação de funções de automação e proteção dos sistemas elétricos.

Andrén et al. (2014) propõem, então, utilizar o padrão IEC 61499 para implementar as funções descritas através do padrão IEC 61850. O padrão IEC 61499 apresenta um modelo que descreve a automação de processos industriais e de sistemas de controle e medição utilizando abstrações chamadas *function blocks*. A principal ideia, segundo Andrén et al. (2014), é gerar aplicações baseadas em IEC 61499 a partir de descrições baseadas em IEC 61850. Para isso, Andrén et al. (2014) utilizam uma abordagem baseada em MDE com objetivo de transformar automaticamente modelos conforme IEC 61850 em modelos conforme IEC 61499.

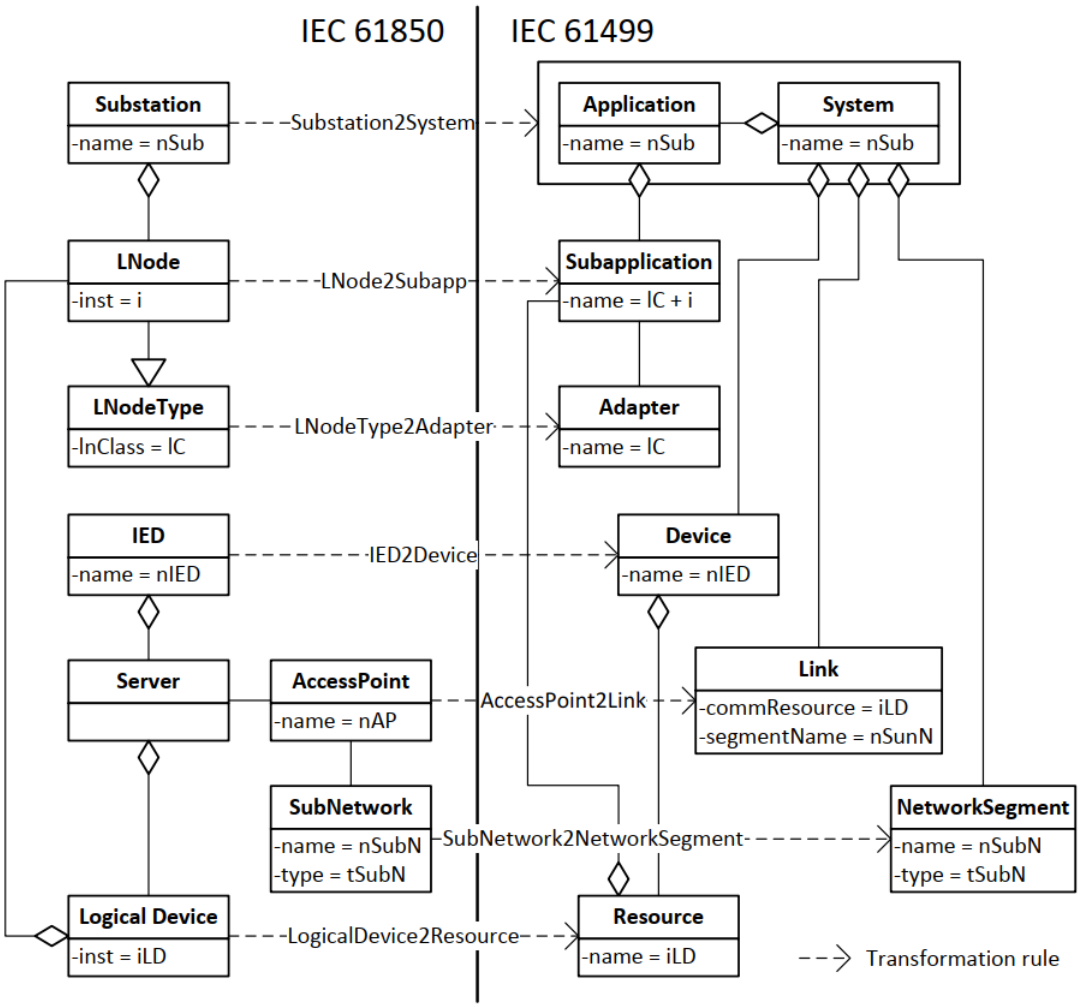
Metamodelos representando tanto o IEC 61850 quanto o IEC 61499 foram fornecidos, e o mapeamento entre os dois metamodelos foi criado, como é ilustrado na Figura 3.1. Este mapeamento foi utilizado para produzir as regras de transformação entre IEC 61850 e IEC 61499.

Um exemplo foi fornecido por Andrén et al. (2014) para ilustrar a abordagem proposta. Uma aplicação que especifica a comunicação entre dois IEDs foi modelada utilizando uma configuração descrita conforme IEC 61850. A partir desta configuração, uma aplicação conforme IEC 61499 foi gerada e, posteriormente, foi simulada no 4DIAC, que consiste em um ambiente *open source* para validação de modelos conforme IEC 61499.

Para a implementação das transformações, o ambiente Eclipse foi utilizado. Os metamodelos foram produzidos utilizando o *Eclipse Modeling Framework* e as regras de transformação foram definidas utilizando o ATL.

Segundo Andrén et al. (2014), a abordagem baseada em MDE apresentada abre novas possibilidades para o desenvolvimento de aplicações de automação para *Smart*

Figura 3.1 – Mapeamento entre os metamodelos de IEC 61850 e IEC 61499 proposto por Andrén et al. (2014).



Fonte: (ANDRÉN et al., 2014).

Grids, como: geração automática e implementação de aplicações de controle baseadas em descrições conforme IEC 61850; maior consistência através das transformações entre modelos; melhor simulação e validação das aplicações baseadas em IEC 61499 e maior reuso através da independência de plataforma.

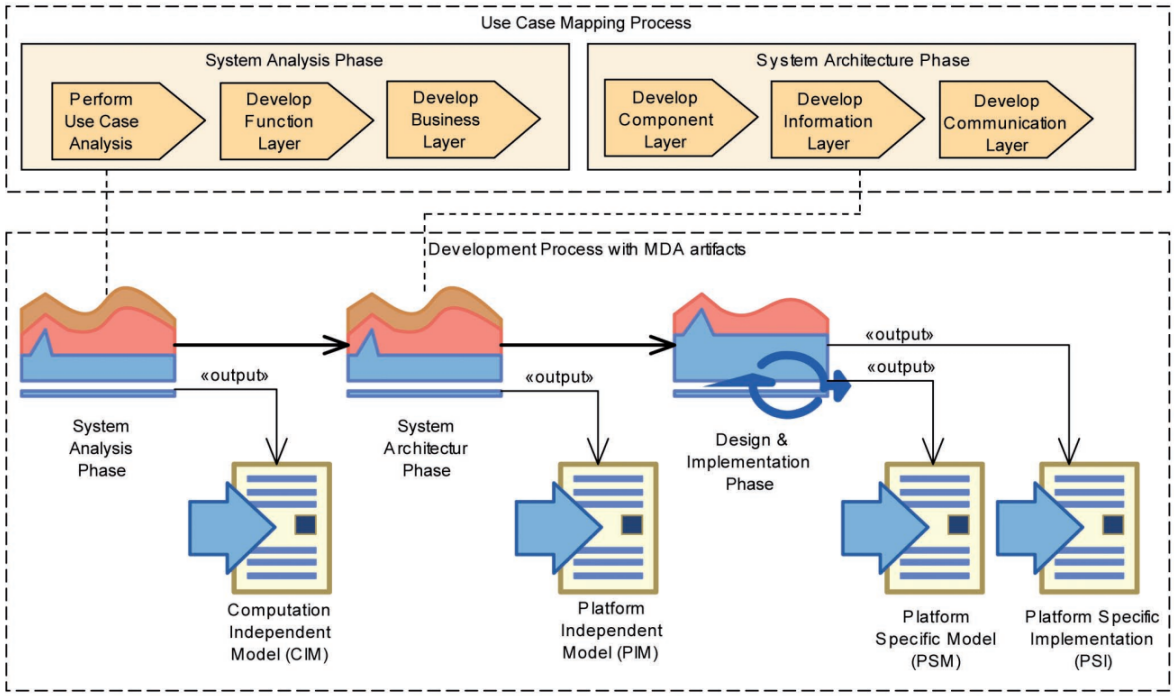
3.2.2 *Towards a model-driven-architecture process for smart grid projects* (DÄNEKAS et al., 2014)

Dänekas et al. (2014) apontam a necessidade de processos e ferramentas apropriadas para facilitar o gerenciamento da arquitetura dos sistemas de *software* em *Smart Grids*. O SGAM fornece uma base para a construção de arquiteturas específicas para o domínio de sistemas de energia elétrica. Além disso, o Mandato M/490 propõe uma metodologia para o gerenciamento de casos de uso, o que inclui um *template* para a descrição de casos de uso.

No entanto, existe uma distância entre o propósito original do SGAM, que é identificar lacunas para padronização, e a sua aplicação na prática. A falta de formalismo do SGAM e das descrições de casos de uso em *Smart Grids* é apontada por Dänekas et al. (2014) como um problema relacionado a essa distância.

Dänekas et al. (2014) propõem a utilização de metamodelos para descrever formalmente arquiteturas baseadas no SGAM e a especificação de casos de uso. Uma abordagem baseada em MDA para suportar o desenvolvimento de sistemas de *software* para *Smart Grids* é proposta, como mostra Figura 3.2. A abordagem consiste em um processo de engenharia que contém três fases: Análise do Sistema, Arquitetura do Sistema e a fase de *Design* e Implementação. Os entregáveis de cada uma das fases correspondem aos *viewpoints* definidos no contexto de MDA: CIM (*Computation Independent Model*), PIM e PSM, respectivamente.

Figura 3.2 – Abordagem para a engenharia de sistemas para *Smart Grids* proposta por Dänekas et al. (2014).



Fonte: (DÄNEKAS et al., 2014).

Na parte de cima da Figura 3.2, as seis tarefas correspondentes à especificação dos casos de uso são definidas e mapeadas ao processo de desenvolvimento. A ideia principal da abordagem proposta por Dänekas et al. (2014) é realizar este mapeamento direto entre a especificação dos casos de uso e a arquitetura do sistema.

Para avaliar a abordagem proposta, Dänekas et al. (2014) fizeram a análise de dois projetos de *Smart Grids* que implementam os conceitos apresentados: o INTEGRA e o DISCERN. Uma DSL foi criada para cada projeto. No projeto INTEGRA, a DSL criada

descreve casos de uso de mais alto nível que invocam vários casos de uso primários que, por sua vez, podem conter cenários e passos para descrever as funcionalidades em detalhes. A DSL do INTEGRA também prevê que a informação que é trocada entre sistemas pode ser definida por um padrão ou norma.

O projeto DISCERN tem como objetivo principal fornecer uma visão comum para soluções de *Smart Grids* baseado em uma especificação conceitual. O conceito central utilizado na DSL proposta para o DISCERN consiste nas sub-funcionalidades, que são utilizadas para estruturar os objetivos e abordagens para soluções de *Smart Grids* em redes de distribuição. Existe uma separação dentro dessa DSL entre os casos de uso e o SGAM. Os casos de uso são criados para expressar objetivos e aspectos comportamentais de uma sub-funcionalidade, enquanto os elementos que representam o SGAM são utilizados para fornecer informações sobre as opções da arquitetura.

O INTEGRA utiliza uma *toolbox* no *Enterprise Architect* chamada *SGAM-Toolbox* para manipular e gerenciar os modelos. O DISCERN utiliza modelos criados em *Microsoft Visio*, porém, Dänekas et al. (2014) afirmam que o desenvolvimento de ferramentas de suporte para o DISCERN é planejado para implementações futuras.

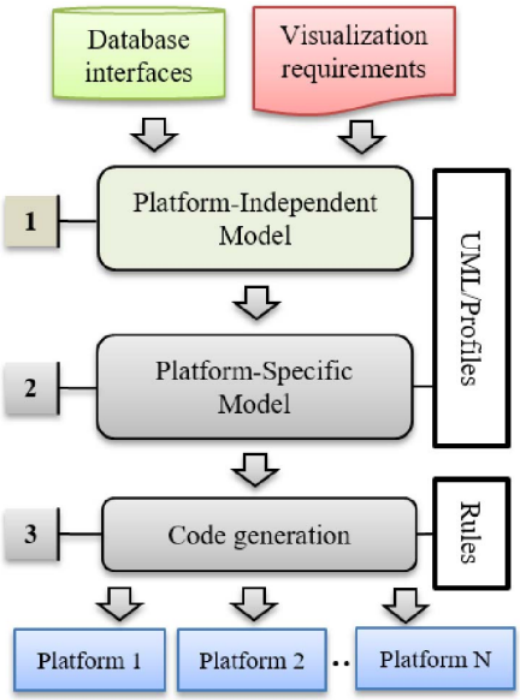
3.2.3 *Model-Driven Design Approach for Building Smart Grid Applications* (EBEID et al., 2016)

Ebeid et al. (2016) afirmam que a utilização de aplicações para visualização e análise de uma grande quantidade de dados é relevante dentro do contexto de *Smart Grids*. No entanto, com a rápida evolução dos dispositivos móveis, surgiu a necessidade de se desenvolver aplicações de visualização de dados para esses dispositivos. No entanto, customizar e adaptar as aplicações para cada tipo de dispositivo que o usuário possui - computador, *smartphone*, *tablet* etc - é um processo tedioso e propenso a erro.

Ebeid et al. (2016) propõem então uma abordagem baseada em MDA para modelar aplicações de visualização de dados para o domínio de *Smart Grids* e gerar as implementações para diferentes plataformas. A abordagem proposta pode ser visualizada na Figura 3.3. O PIM é conforme um perfil de UML que contém alguns aspectos de *Smart Grids* e de aplicações para visualização de dados de uma forma bem genérica. A intenção é que o PIM seja totalmente independente da plataforma de implementação. O PSM é conforme um perfil UML que contém conceitos específicos de plataformas, por exemplo *Angular JS* e *Bootstrap*, que são *frameworks* para desenvolvimento de páginas Web que podem ser visualizadas em diferentes dispositivos.

A metodologia proposta por Ebeid et al. (2016) para gerar aplicações de visualização de dados em *Smart Grids* pode ser descrita em três passos:

Figura 3.3 – Abordagem baseada em MDA para o desenvolvimento de aplicações para *Smart Grids* proposta por Ebeid et al. (2016).



Fonte: (EBEID et al., 2016).

1. Definir o PIM utilizando um conjunto de diagramas UML: um diagrama de classes representando a estrutura da aplicação; um diagrama de estados representando os diferentes estados da aplicação e as transições; um diagrama de classe da API Rest a ser consumida pela aplicação; e um diagrama de implantação que define as plataformas alvo;

2. Converter o PIM criado em um PSM: Esta conversão é feita utilizando um conjunto de regras de transformação. No entanto, a tecnologia utilizada para realizar as transformações não foi citada pelos autores no texto;

3. Converter o PSM em código fonte. Isso é feito através do mapeamento dos elementos do PSM a blocos de código parametrizados. A geração de código é automática. No entanto, como aponta Ebeid et al. (2016), o código gerado não é completo, e necessita de ajustes manuais. Os autores recomendam que os aspectos que não foram gerados automaticamente sejam representados em uma versão mais completa do PSM.

A abordagem proposta por Ebeid et al. (2016) foi avaliada experimentalmente através da criação de uma aplicação para visualização de dados de consumo residencial de energia elétrica como parte de um projeto chamado *SmartHG*, que disponibiliza os dados de consumo de energia através de uma API *RESTful*.

Ebeid et al. (2016) realizaram o desenvolvimento da aplicação de forma manual (sem utilização da abordagem proposta) e utilizando a abordagem proposta para comparar os resultados. Foi avaliado que, utilizando a abordagem proposta, o esqueleto do código foi gerado corretamente. Porém, alguns detalhes de implementação não foram gerados automaticamente e tiveram que ser supridos manualmente. As transformações de modelos geraram uma parte suficientemente grande da aplicação o que, segundo os autores, reduz o esforço de desenvolvimento e o tempo necessário para desenvolver as aplicações. As aplicações para visualização de dados para Web e para *Smartphone Android* foram geradas e testadas.

Como trabalhos futuros Ebeid et al. (2016) citaram a utilização dos padrões SCADA e CIM para descrever os objetos transferidos entre os usuários e os operadores do sistema elétrico.

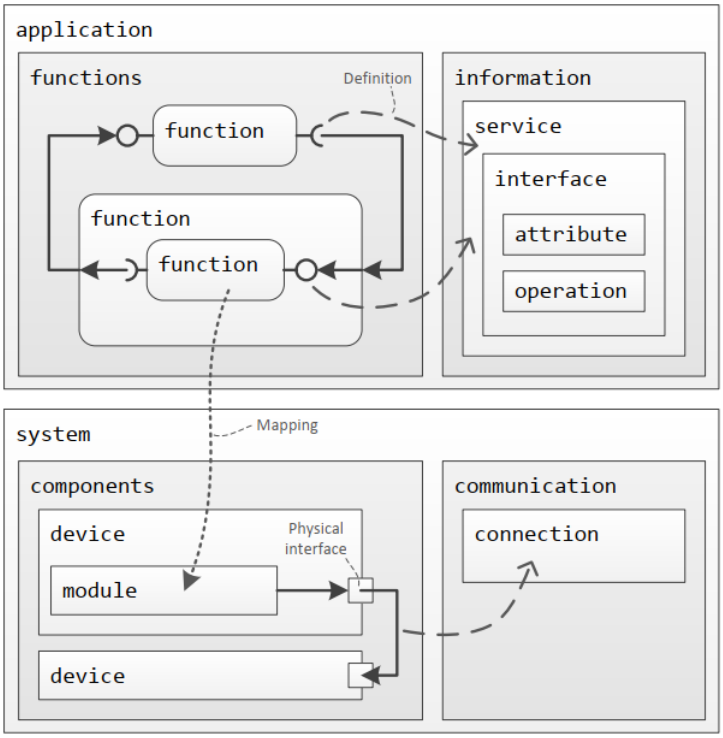
3.2.4 *Applying the SGAM methodology for rapid prototyping of smart grid applications* (ANDRÉN et al., 2016)

Segundo Andrén et al. (2016), para gerenciar a crescente complexidade dos sistemas em *Smart Grids* o detalhamento de casos de uso e engenharia de requisitos é nececessário. Várias metodologias existem na literatura para abordar essas questões, inclusive utilizando o SGAM como referência. A utilização de metodologias desse tipo gera uma grande quantidade de informações acerca da especificação dos sistemas e casos de uso. No entanto, Andrén et al. (2016) identificam que abordagens para derivar uma implementação diretamente da especificação do sistema são escassas em projetos de *Smart Grids*. Desta forma, muito trabalho é necessários ao menos duas vezes: durante a especificação e durante a implementação.

Andrén et al. (2016) propõem uma DSL textual baseada no SGAM, com a definição de sintaxe, semântica e um modelo formal que pode ser utilizado por métodos da MDE para, por exemplo, gerar código de implementação dos sistemas. O principal objetivo do trabalho de Andrén et al. (2016) é demonstrar como a DSL proposta pode ser utilizada não apenas para modelar casos de uso, mas também gerar a implementação dos casos de uso de forma semiautomática.

A DSL apresentada por Andrén et al. (2016) chama-se PSAL (do inglês, *Power System Automation Language*), que foi definida em EBNF. Uma representação gráfica da PSAL é apresentada na Figura 3.4. A linguagem PSAL fornece uma sintaxe simples para projetar sistemas conforme o SGAM. Ela é composta de duas abstrações principais: *application* e *system*. A *application* contém as descrições das camadas de *Business*, *Function* e *Information* do SGAM, enquanto que *system* contém elementos das camadas *Communication* e *Component* do SGAM.

Figura 3.4 – Representação gráfica da linguagem PSAL proposta por Andrén et al. (2016).



Fonte: (ANDRÉN et al., 2016).

Andrén et al. (2016) fazem uma discussão sobre uma metodologia de prototipação rápida que consiste em gerar código a partir de um modelo conforme PSAL. Andrén et al. (2016) dão três passos básicos para a prototipação rápida: (i) as funções (elementos do tipo *function* em PSAL) devem ser transformadas em código fonte executável utilizando alguma ferramenta de transformação; (ii) as interfaces de comunicação devem ser implementadas e configuradas; (iii) as aplicações recém criadas precisam ser compiladas e implantadas nos dispositivos correspondentes.

Andrén et al. (2016) dão um exemplo simples de utilização da linguagem PSAL, que consiste na implementação de um controle de potência reativa em um sistema de distribuição. Este sistema consiste em um IED para aquisição de dados, um controlador para atuar no sistema e um computador que contém a aplicação de DMS para controle de potência reativa. Este caso de uso foi inteiramente modelado conforme a arquitetura do SGAM utilizando a DSL proposta (linguagem PSAL). Segundo Andrén et al. (2016), a única função do sistema que necessitou a geração de código foi a aplicação de DMS. A geração de código foi feita através de uma transformação de modelo-a-modelo cujo modelo de entrada é conforme PSAL e o modelo de saída é conforme IEC 61499. O modelo IEC 61499 foi importado para o 4DIAC (uma ferramenta para aplicações de controle baseadas em IEC 61499) para utilização. A ferramenta de transformação utilizada é baseada em QVT.

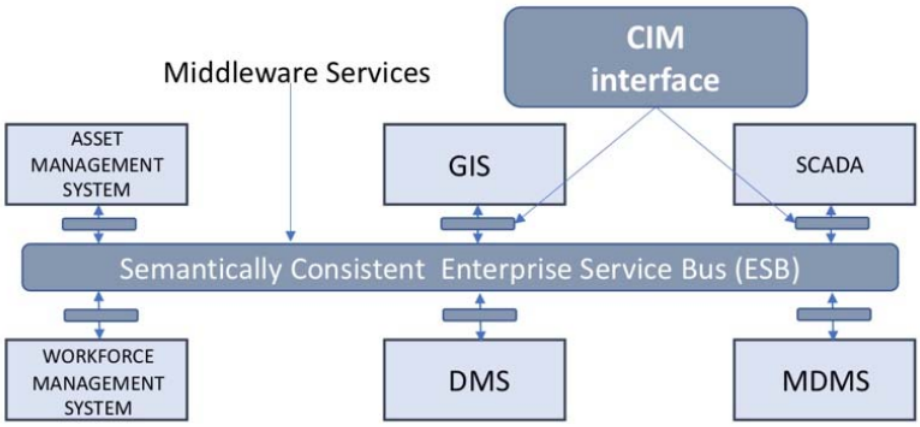
3.2.5 *CIM-based integration in smart grids: Slovenian use cases* (SOUVENT et al., 2019)

Souvent et al. (2019) afirmam que, dentro do contexto de *Smart Grids*, os operadores dos sistemas de transmissão - TSOs (do inglês, *Transmission System Operators*) irão cada vez mais interagir e trocar dados com os operadores dos sistemas de distribuição - DSOs (do inglês, *Distribution System Operators*) e outros operadores, entidades e usuários do sistema. Nesse cenário de integração de sistemas, regras e conceitos que irão permitir a interoperabilidade ao longo de todo o sistema de potência são necessários. No entanto, diferentes entidades e dispositivos utilizam vários formatos para representar a informação e diferentes protocolos de comunicação, o que torna a tarefa de integração em *Smart Grids* um grande desafio.

Segundo Souvent et al. (2019), arquiteturas de ICT baseadas em CIM, com foco na representação padrão da informação e no seu significado, pode habilitar a interoperabilidade para os sistemas de energia elétrica do futuro.

Souvent et al. (2019) fazem uma explanação sobre arquiteturas de sistemas de *software* que podem ser utilizadas para integração baseada em CIM de aplicações para *Smart Grids*. Uma arquitetura baseada em *middleware* é apresentada, como mostra a Figura 3.5. Nessa arquitetura, a integração é alcançada através de um "*message bus*" (barramento de mensagens, em português) implementado como um *middleware* que permite que as aplicações se comuniquem utilizando formatos de mensagem pré-definidos. O *message bus* é denotado como "CIM *Enterpripe Service Bus*" (ESB).

Figura 3.5 – Arquitetura de *Enterprise Service Bus* baseada em CIM proposta por Souvent et al. (2019) para prover interoperabilidade em *Smart Grids*.



Fonte: (SOUVENT et al., 2019).

Esta metodologia para integração de aplicações de *Smart Grids* foi demonstrada através de um projeto nacional de *Smart Grids* da Eslovênia chamado projeto NEDO, onde a integração baseada em CIM foi realizada para várias aplicações de DSOs e TSOs como

GIS (*Geographical Information System*), DMS (*Distribution Management System*), MDMS (*Meter Data Management System*) e DR (*Demand Response*). Os dados compartilhados entre as aplicações foram preparados em formato XSD ou RDFS. As ferramentas *Enterprise Architect* e *CIMTool* foram utilizadas para criar os perfis de CIM utilizados no projeto.

Segundo Souvent et al. (2019), a integração das aplicações do projeto NEDO utilizando ESB-CIM foi bem sucedida. Os resultados mostraram que a integração de sistemas baseada em CIM reduz a complexidade de integração e evita várias integrações um-a-um, que podem levar à chamada "arquitetura de integração *spaghetti*" que é cara e difícil de manter no longo prazo (SOUVENT et al., 2019).

3.2.6 *Towards a model-centric approach for developing dependable smart grid applications* (FISCHINGER et al., 2019)

Fischinger et al. (2019) abordam a necessidade de se garantir a *confiança* de sistemas críticos e complexos como *Smart Grids*. *Confiança* é um requisito de sistemas críticos que inclui os atributos *confiabilidade*, *disponibilidade*, *mantenabilidade*, *segurança*, *integridade* e *privacidade* (FISCHINGER et al., 2019; AVIZIENIS et al., 2004). Segundo Fischinger et al. (2019), "*dado que os métodos convencionais de engenharia de sistemas podem cumprir esses requisitos em sistemas simples, a questão é: como a confiança pode ser abordada no contexto de sistemas de sistemas?*". Abordagens baseadas em MDA e SGAM têm sido propostas para lidar com a traçabilidade de requisitos para sistemas de *Smart Grids*, porém, como identifica Fischinger et al. (2019), a traçabilidade desses requisitos até a implementação de *software* executável ainda não é suportada nesse contexto.

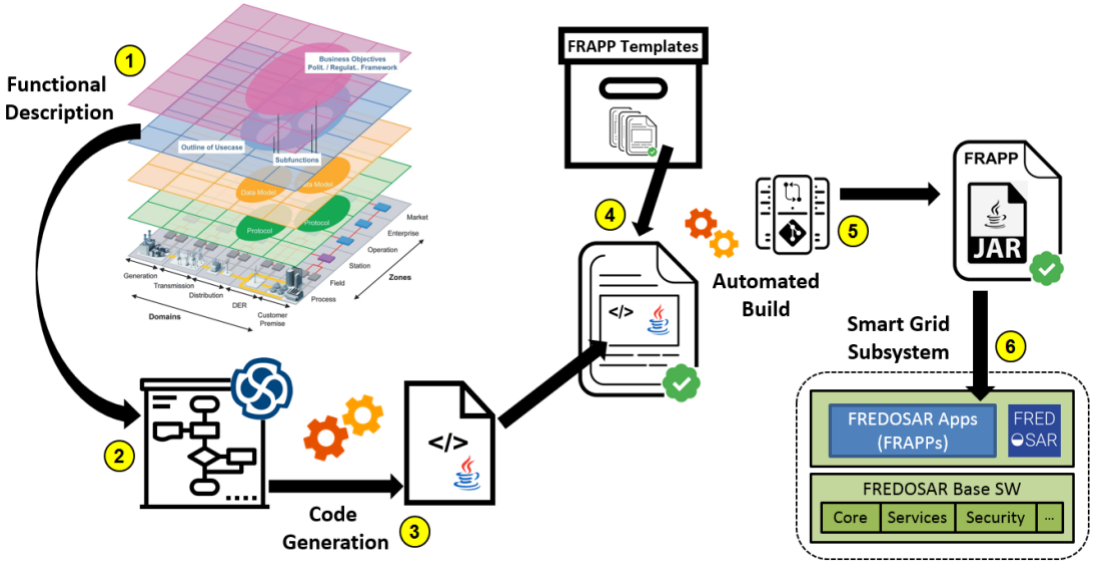
Fischinger et al. (2019) propõem uma abordagem para desenvolvimento de *software* centrada em modelos para *Smart Grids* com a implementação de geração de código para alinhar os modelos de alto nível do sistema com a sua implementação concreta. Uma metodologia para avaliar a traçabilidade dos requisitos é aplicada.

A abordagem proposta por Fischinger et al. (2019), ilustrada na Figura 3.6, chama-se *SGAM-Model-Centric Development* e tem como objetivo gerar a implementação de código a partir das funções do sistema descritas na camada *Function* do SGAM. Essa abordagem consiste em 6 passos: (1) criar um modelo do sistema baseado em SGAM utilizando o *SGAM-toolbox*, ferramenta disponível para *Enterprise Architect*; (2) descrição detalhada das funcionalidades com um diagrama de atividade; (3) geração de código Java utilizando o gerador de código do *Enterprise Architect* a partir do diagrama de atividade; (4) aplicação de um *template* de código da plataforma FREDOSAR[1]; (5) o código resultante é compilado em um servidor dedicado do GitLab; (6) o resultado da compilação é uma aplicação executável que pode ser executada em uma máquina com o

[1] Mais informações em: <https:://www.fredosar.org/>. Acesso em: 08/03/2021.

framework FREDOSAR configurado.

Figura 3.6 – Abordagem centrada em modelos proposta por Fischinger et al. (2019) para o desenvolvimento de aplicações em *Smart Grids*.



Fonte: (FISCHINGER et al., 2019).

Duas aplicações para exemplificar a abordagem foram desenvolvidas: *Electric Vehicle Charging*, uma aplicação do lado do consumidor que verifica o preço da energia elétrica para o usuário, e *Variable Renewable Energy Tariff*, uma aplicação do DSO que calcula o preço da energia elétrica a partir da demanda e oferta de energia elétrica atual. As duas aplicações trocam informações sobre o preço da eletricidade através dos serviços de *middleware* fornecidos pela plataforma FREDOSAR.

Fischinger et al. (2019) avalia que, embora a geração de código a partir das descrições das funcionalidades (diagramas de atividade) facilita o processo de desenvolvimento significamente, a traçabilidade entre os requisitos de negócio e o código fonte resultante ainda não pode ser garantida, já que mudanças na lógica de negócio requer mudanças manuais nos diagramas de atividade, que são a base para a geração de código. Como trabalhos futuros, Fischinger et al. (2019) pretendem adicionar a interface de geração de código utilizada como um *Add-in* para o *SGAM-toolbox*, estender a abordagem para implementação em outras plataformas e, por fim, levar em consideração as camadas de comunicação e informação do SGAM para configurar os dispositivos de *Smart Grid*.

3.3 Análise dos Trabalhos Relacionados

Em comum, as pesquisas descritas compartilham do desafio e da necessidade de se buscar alternativas viáveis de abordagens para o desenvolvimento de *software* para

Smart Grids. O objetivo principal, nesse campo de pesquisa, é gerar, futuramente, um ambiente de maturidade de normas, padrões, arquiteturas de *software* e metodologias de desenvolvimento para que haja segurança do ponto de vista técnico e científico para a implantação de sistemas tão complexos e críticos como os sistemas de energia elétrica do futuro.

Nesse contexto, é importante que as limitações dos trabalhos do estado da arte sejam demonstradas. Assim, será possível colocar de maneira mais clara quais foram as contribuições do trabalho apresentado neste manuscrito. Para isso, os trabalhos descritos na seção 3.2 foram submetidos a uma análise na qual estas pesquisas foram avaliadas a partir de critérios pré-definidos.

Os critérios de avaliação dos trabalhos do estado da arte são descritos a seguir:

1. *É uma abordagem baseada em MDE*: determina se o trabalho analisado utiliza modelos como artefatos principais dentro da abordagem de desenvolvimento. Para ser considerada uma abordagem baseada em MDE, os modelos utilizados devem ser obedecer aos princípios da conformidade e da representação (ver a subseção 2.4.3), mesmo que nem todo o sistema seja baseado em modelos. Este critério pode ser respondido com "SIM" ou "NÃO";

2. *Auxilia na análise e definição das funcionalidades do software*: determina se a abordagem proposta dá algum suporte para a especificação das funcionalidades do *software*. Este suporte pode ser a representação das funções do sistema através de diagramas, uma DSL, um modelo conforme UML ou outra linguagem de modelagem ou linguagem textual. Este critério pode ser respondido com "SIM" ou "NÃO";

3. *Auxilia o projeto da arquitetura de software*: determina se a abordagem leva em consideração a arquitetura de *software* utilizada para realizar a implementação das aplicações, o que corresponde à fase de projeto em engenharia de *software*. Este critério pode ser respondido com "SIM" ou "NÃO";

4. *Auxilia na implementação do software*: determina se a abordagem auxilia na produção de artefatos de implementação das aplicações como código e arquivos de configuração. Esse critério pode ser respondido com "NÃO", "MANUAL", "AUTOMÁTICO" ou "SEMIAUTOMÁTICO";

5. *Utiliza transformações de modelo-a-modelo*: determina se a abordagem faz uso de transformações de modelo para automatizar o processo de desenvolvimento. A resposta para esta pergunta pode ser "NÃO" ou, em caso positivo, indica-se qual tecnologia foi utilizada para implementar as transformações. Caso o trabalho analisado indique que as transformações são utilizadas em sua abordagem, mas não

deixa claro qual tecnologia é utilizada ou as transformações não foram implementadas na prática, a resposta é apenas "SIM";

6. *Uma metodologia para implementar a abordagem proposta é fornecida*: determina se o trabalho analisado descreve os passos necessários para implementar a abordagem que está sendo proposta. As respostas possíveis para este critério são "SIM" ou "NÃO";

7. *Auxilia o desenvolvimento de aplicações completas*: determina se a abordagem proposta auxilia o desenvolvimento de sistemas completos, com a definição de suas funcionalidades (lógica de negócio) e a modelagem dos dados utilizados. Este critério pode ser respondido com "SIM" ou "NÃO";

8. *Utiliza padrões internacionais para a representação de dados*: determina se a abordagem proposta utiliza modelos definidos por padrões do domínio de sistemas de energia elétrica como IEC 61970, IEC 61499 ou IEC 61850 para representar os dados que são processados pelas aplicações desenvolvidas. A resposta para esse critério pode ser "NÃO" ou "SIM, <padrão utilizado>";

9. *Possui uma clara separação entre a modelagem de dados e a lógica de negócio a nível de modelo*: determina se, na abordagem analisada, a modelagem dos dados é feita de maneira totalmente independente da modelagem das funcionalidades do sistema (lógica de negócio). A resposta a esse critério pode ser "SIM" ou "NÃO";

10. *Fornece uma abordagem para suportar a integração de aplicações em Smart Grids*: determina se a abordagem analisada propõe alguma arquitetura de *software* e/ou a utilização de algum protocolo, plataforma ou *framework* para a comunicação e integração de aplicações de *software* de *Smart Grids*. A resposta para este critério pode ser "SIM, EM TEORIA" se o trabalho analisado apresenta uma abordagem para a integração de aplicações, mas não apresenta uma comprovação experimental, ou seja, uma implementação prática da abordagem proposta; pode ser "SIM, NA PRÁTICA", se o trabalho analisado implementa uma solução para a integração de aplicações de *Smart Grids* na prática; ou pode ser "NÂO" se o trabalho analisado não apresenta uma proposta para a integração de aplicações.

A Tabela 3.1 apresenta uma análise comparativa dos trabalhos relacionados utilizando os critérios elaborados. Todos os trabalhos analisados oferecem uma abordagem que utiliza MDE, com exceção de Souvent et al. (2019). De fato, o trabalho apresentado por Souvent et al. (2019) não se propõe a fornecer uma abordagem para auxiliar no desenvolvimento de aplicações, mas é uma das poucas abordagens entre os trabalhos analisados que apresenta uma arquitetura de *software* baseada em *middleware* ESB e CIM para integração de aplicações em *Smart Grids*. A ideia apresentada por Souvent et al.

Tabela 3.1 – Análise dos trabalhos relacionados.

Critérios de avaliação	Andrén et al. (2014)	Dänekas et al. (2014)	Ebeid et al. (2016)	Andrén et al. (2016)	Souvent et al. (2019)	Fischinger et al. (2019)
É uma abordagem baseada em MDE?	Sim	Sim	Sim	Sim	Não	Sim
Auxilia na análise e definição das funcionalidades do *software*?	Não	Sim	Sim	Sim	Não	Sim
Auxilia no projeto da arquitetura de *software*?	Não	Sim	Sim	Sim	Sim	Sim
Auxilia a implementação do *software*?	Sim	Sim	Sim	Sim	Não	Sim
Utiliza transformações de modelo-a-modelo?	Sim, QVT	Não	Sim	Sim, QVT	Não	Não
Uma metodologia para implementar a abordagem proposta é fornecida?	Sim	Sim	Sim	Sim	Não	Sim
Auxilia o desenvolvimento de aplicações completas?	Não	Sim	Sim	Sim	Não	Sim
Utiliza padrões internacionais para representação dos dados?	Sim	Sim	Não	Sim	Sim	Não
Possui uma clara separação entre a modelagem de dados e a lógica de negócio?	Não	Não	Não	Não	Sim	Não
Fornece uma abordagem para suportar a integração de aplicações em *Smart Grids*?	Não	Não	Não	Não	Sim, em teoria	Sim, na prática

(2019) é que todas as aplicações dentro de um ambiente de *Smart Grids* tenham acesso a um mesmo repositório de modelos (que são conforme o CIM) e que estas aplicações possam se comunicar através de uma plataforma de *middleware* em comum. No entanto, uma limitação importante do trabalho de Souvent et al. (2019) é que ele não apresenta uma implementação prática da arquitetura de ESB apresentada.

Entre os trabalhos que utilizam MDE para o desenvolvimento de aplicações para *Smart Grids*, o trabalho de Andrén et al. (2014) não apresenta meios para o especialista de domínio representar a lógica de negócio da aplicação, tampouco a arquitetura de

software utilizada para comunicação é especificada. O trabalho de Andrén et al. (2014) se concentra em criar descrições de aplicações de controle conforme o padrão IEC 61499 a partir de modelos conforme o padrão IEC 61850, o que reduz consideravelmente o escopo de aplicação da abordagem proposta. Andrén et al. (2014) se propõem, portanto, a gerar artefatos de partes dos sistemas, não sistemas completos.

O trabalho de Dänekas et al. (2014) tem como objetivo principal alinhar as especificações de casos de uso com as abstrações definidas no SGAM. Desta forma, existe um alinhamento entre os modelos da aplicações com as especificações de casos de uso. Além disso, um guia metodológico é apresentado para auxiliar no desenvolvimento de aplicações para *Smart Grids*. No entanto, a abordagem proposta por Dänekas et al. (2014) carece de ferramentas para a automação do processo de desenvolvimento de *software*. Eles não utilizam transformações de modelos e, além disso, não fornecem uma abordagem para geração de código.

Os trabalhos produzidos por Ebeid et al. (2016) e Fischinger et al. (2019), por sua vez, carecem da utilização de padrões internacionais para representação de dados no domínio de *Smart Grids*, como os padrões fornecidos pelo IEC ou pelo IEEE. A utilização dos padrões e normas internacionais para representação de dados é muito importante em *Smart Grids*, pois garante a chamada interoperabilidade de dados, ou interoperabilidade semântica.

A abordagem proposta por Andrén et al. (2016) atende a quase todos os critérios utilizados. Eles propõem uma DSL para facilitar a utilização do SGAM para o desenvolvimento de aplicações para *Smart Grids*. A DSL proposta é uma linguagem textual chamada PSAL (*Power System Automation Language*), definida em EBNF. A linguagem PSAL expressa todas as camadas do SGAM e facilita o processo de desenvolvimento, pois possui uma sintaxe simples. Andrén et al. (2016) propõem uma metodologia para utilizar a abordagem proposta, na qual as construções de *software* descritas em PSAL são transformadas automaticamente em modelos conforme o padrão IEC 61499 para automação de sistemas de energia elétrica, o que limita a abordagem proposta a problemas de automação. Esta abordagem também suporta a utilização do padrão IEC 61850 para a representação de dados através do mecanismo de herança, isto é, os serviços descritos na linguagem PSAL que desejam utilizar os dados definidos em um modelo IEC 61850 ou CIM devem herdar as suas interfaces. Isso é feito diretamente em código PSAL.

A partir da análise dos trabalhos relacionados, verifica-se que existe a necessidade por abordagens mais abrangentes que auxiliem o desenvolvimento de aplicações para *Smart Grids* com diferentes escopos de atuação, indo além das aplicações de automação baseadas em IEC 61499. Além disso, a julgar pelas respostas ao último critério da Tabela 3.1, pode-se verificar também que abordagens para o desenvolvimento de *software* em *Smart Grids* que fazem uma clara separação, a nível de modelo, entre a modelagem dos

dados da rede elétrica e a lógica de negócio das aplicações são escassas na literatura assim como abordagens que suportem a integração de aplicações em *Smart Grids*.

3.4 Síntese

Este capítulo contém uma revisão de seis trabalhos relacionados à abordagens baseadas em modelos que visam auxiliar a atividade de desenvolvimento de *software* para *Smart Grids*. Uma análise comparativa entre estes trabalhos foi realizada e uma tabela (Tabela 3.1) com o resultado desta análise é fornecida. Esta revisão da literatura possibilitou verificar que vários trabalhos têm utilizado a MDE para suportar o desenvolvimento de aplicações para *Smart Grids*. Uma limitação importante observada entre os trabalhos que aplicam MDE nesse contexto é que as abordagens propostas abordam um escopo pequeno de aplicações, por exemplo, aplicações de automação e controle. Outra constatação é a de que existe a necessidade de se propor abordagens que separem os modelos que representam os equipamentos e dispositivos do sistema elétrico dos modelos que representam a lógica de negócio das aplicações de *Smart Grids*, com o intuito de preservar os investimentos, diminuir o retrabalho e aumentar a flexibilidade.

4 FMDE4SGRID: Um *Framework* baseado em MDE para suportar o desenvolvimento de *software* para *Smart Grids*

Este capítulo contém uma descrição do Framework baseado em MDE e *weaving* de modelos para suportar a atividade de desenvolvimento de aplicações de *software* para o domínio de *Smart Grids* (FMDE4SGRID).

O FMDE4SGRID tem como objetivo definir uma arquitetura contendo modelos, metamodelos, transformações de modelos e as relações entre estes elementos para auxiliar a especificação da lógica de negócio, o projeto de arquitetura de *software* e a tarefa de gerar o código fonte e arquivos de configuração de aplicações para *Smart Grids*.

Este capítulo é organizado como descrito a seguir. A Seção 4.1 contém uma descrição da construção do FMDE4SGRID, descrevendo o principal fundamento utilizado na sua criação. A Seção 4.2 contém a definição formal do FMDE4SGRID a partir da sua arquitetura. A Seção 4.3 contém os metamodelos definidos dentro do escopo do FMDE4SGRID. Na Seção 4.4, uma metodologia para a utilização do FMDE4SGRID é fornecida. E, por fim, a Seção 4.5 traz uma síntese deste capítulo.

4.1 Fundamentação para a proposta do *framework*

Como já exposto na Seção 2.1, *Smart Grid* é um sistema no qual as aplicações que o constituem precisam trocar dados e serem integradas para que este novo sistema de energia elétrica possa alcançar todos os seus benefícios (IVANOV et al., 2015).

As aplicações de *Smart Grids* consistem em sistemas que coletam dados da rede elétrica (*e.g.*, *status* de equipamentos, leituras de sensores e mensagens de alerta), fazem o processamento destes dados e fornecem determinados serviços aos operadores da rede elétrica como o monitoramento da rede, atuação em caso de faltas, faturamento dos consumidores, entre outras funções.

Alguns exemplos de aplicações para *Smart Grids* são listados a seguir (GUNGOR et al., 2012):

- *Demand Response Management* - DRM: sistema que atua para controlar o balanceamento entre o fornecimento e a demanda de energia elétrica;

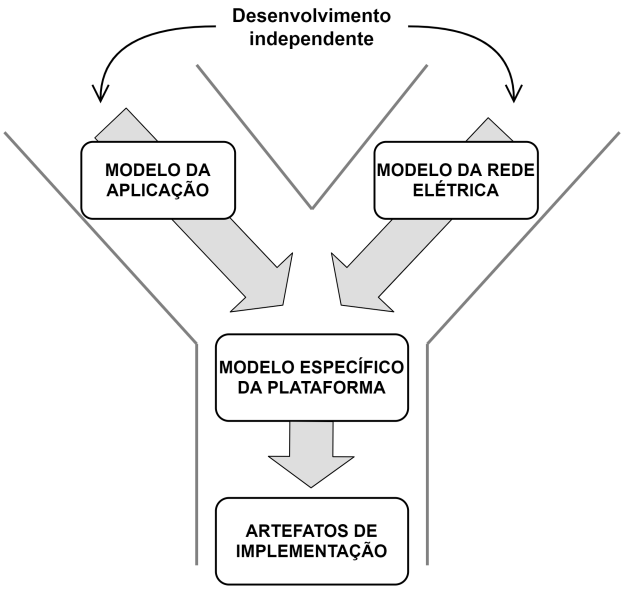
- *Home Energy Management* - HEM: sistema que faz o controle, o monitoramento e a comunicação com as instalações dos consumidores residenciais;
- *Wide-Area Situational Awareness* - WASA: monitoramento em tempo real e de uma grande área geográfica do sistema elétrico;
- *Outage Management System* - OMS: sistema responsável por identificar, gerenciar e localizar as faltas da rede elétrica;
- *Distribution Management System* - DMS: fornece um gerenciamento avançado e completo do sistema de distribuição;
- *Asset Management System* - AMS: fornece gerenciamento, automação, rastreamento e monitoramento de equipamentos, equipes de campo, veículos etc;
- *Meter Data Management System* - MDMS: sistema responsável por armazenar e processar os dados de medidores antes de enviá-los para outras aplicações.

Para garantir a integração e a interoperabilidade dessas aplicações, é necessário que dois requisitos básicos sejam cumpridos. O primeiro diz que as aplicações precisam representar os dados da mesma forma e os dados precisam ter um significado comum. Isso é o que Kim et al. (2017) chama de "*data-driven interoperability*". O segundo requisito básico diz que as aplicações precisam estabelecer a troca de dados por meio de protocolos comuns de comunicação. Isto é o que Kim et al. (2017) chama de "*communication-driven interoperability*".

Satisfazer o primeiro requisito básico para a integração de aplicações em *Smart Grids* significa utilizar padrões internacionais para a representação dos dados da rede elétrica (por exemplo, o padrão CIM, apresentado na seção 2.2). Desta forma, a lógica de negócio das aplicações de *Smart Grids* deve ser definida de forma independente do modelo de dados da rede elétrica, já que o modelo de dados deve ser comum para todas as aplicações. Satisfazer o segundo requisito básico requer o projeto de uma arquitetura de comunicação eficiente e a utilização de uma plataforma específica que permita a integração entre as aplicações de *Smart Grids*.

Neste trabalho, propõe-se uma abordagem em Y para o desenvolvimento de aplicações para *Smart Grids* tal como ilustrada na Figura 4.1. Nesta proposta baseada em desenvolvimento em Y, o desenvolvimento do modelo da lógica de negócio das aplicações ocorre independentemente do desenvolvimento do modelo da rede elétrica. Após a definição desses dois modelos, um modelo específico de plataforma é gerado a partir do entrelaçamento dos dois modelos de entrada. Esse entrelaçamento corresponde à forma como os elementos do modelo da aplicação interagem com o modelo da rede elétrica.

Figura 4.1 – Fundamento básico para a construção do *framework* proposto.



Fonte: baseado em Junior (2017), Stefanello (2017), Matos (2015)

O modelo específico de plataforma contém as especificações necessárias para que as aplicações de *Smart Grids* possam se comunicar umas com as outras. Por fim, os artefatos de implementação, como código fonte e arquivos de configuração, podem ser gerados a partir da especificação da plataforma.

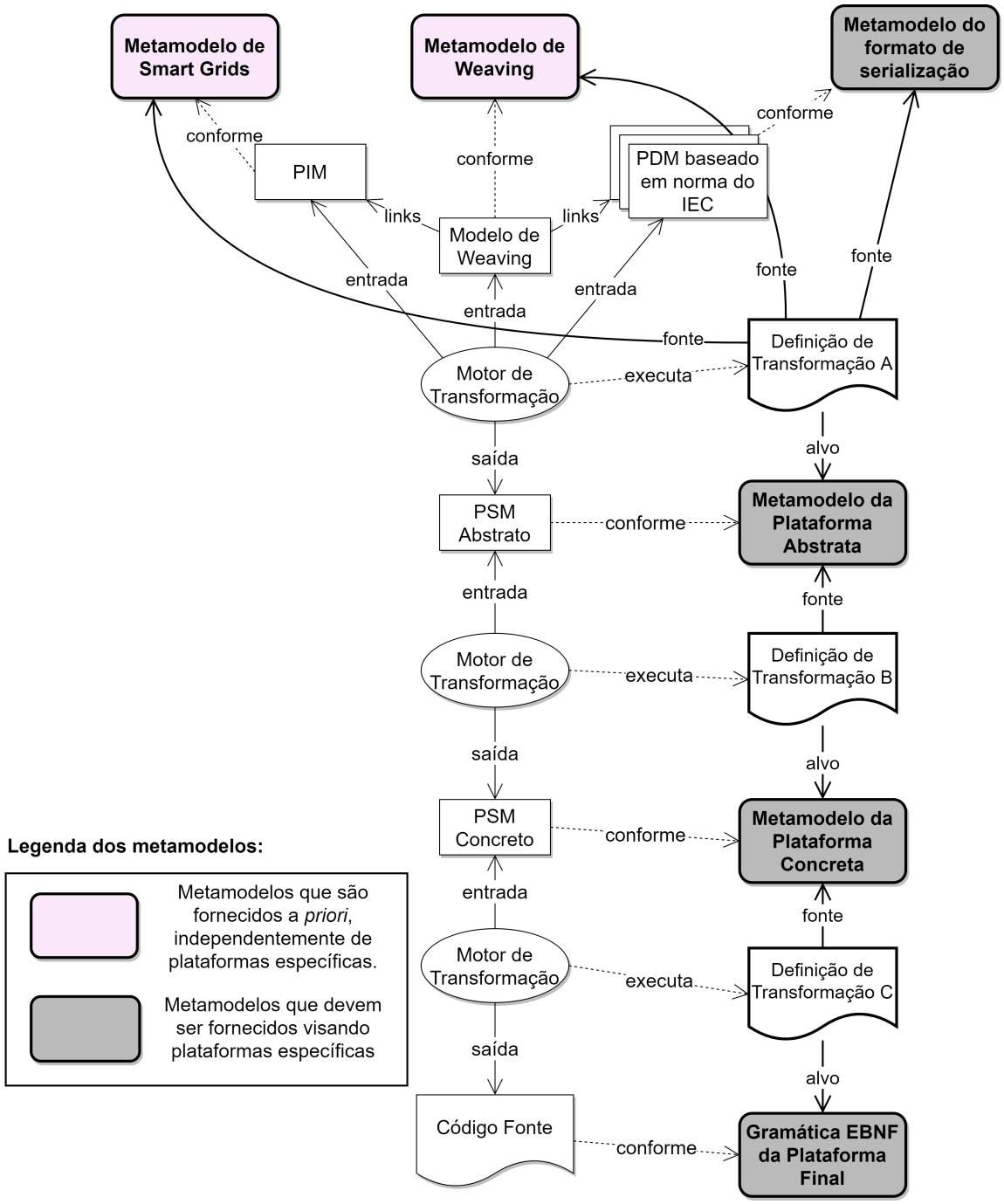
Esse estilo de desenvolvimento de *software* combina com o *desenvolvimento baseado em Y*, apresentado na seção 2.4.6. O "Modelo da Aplicação" da Figura 4.1 corresponde ao PIM, o "Modelo da Rede Elétrica" corresponde ao PDM e o "Modelo Específico da Plataforma" corresponde ao PSM. O *weaving* de modelos é utilizado para entrelaçar o PIM e o PDM e, a partir deste entrelaçamento, o PSM é gerado (JUNIOR, 2017; STEFANELLO, 2017; MATOS, 2015). Na seção seguinte, o FMDE4SGRID é formalmente definido como uma arquitetura de modelos baseada em MDE e *weaving* para suportar o desenvolvimento de *software* para *Smart Grids*.

4.2 Arquitetura do FMDE4SGRID

A arquitetura proposta para o FMDE4SGRID é apresentada na Figura 4.2. Esta arquitetura é baseada na ideia exposta na Figura 4.1 da seção anterior.

O PIM, o PDM e o *weaving* são os modelos de entrada da arquitetura do FMDE4SGRID. O PIM corresponde à lógica de negócio da aplicação de *Smart Grids* e é conforme o metamodelo de *Smart Grids*. O PIM (parte superior à esquerda da Figura 4.2) contém, portanto, a especificação de atores, atividades, funções e parâmetros de uma aplicação de *Smart Grids*, como o MDMS, DMS, *Asset Management*, entre outras.

Figura 4.2 – Arquitetura do FMDE4SGRID: Um *framework* baseado em MDE para suportar o desenvolvimento de *software* para *Smart Grids*.



Fonte: baseado em Junior (2017) e Stefanello (2017).

Os dados da rede elétrica, como a topologia da rede, leituras de sensores ou *status* de equipamentos são modelados em um ou mais PDM (parte superior à direita da Figura 4.2), que são baseados em uma norma do IEC. Para manter a flexibilidade do *framework* proposto, qualquer norma do IEC pode ser utilizada como base para os PDM. Os PDM são conforme o metamodelo do formato de serialização que se deseja utilizar, como XML, XSD, JSON etc.

O modelo de *weaving* (parte superior, ao centro da Figura 4.2) especifica as ligações entre o PIM e um ou mais PDMs. O significado destas ligações é explicado a seguir. O PIM possui, dentro dos seus elementos, a especificação dos dados da rede elétrica que são manipulados e compartilhados com outras aplicações. O objetivo da construção do FMDE4SGRID é dar uma representação ou formato padrão a esses dados. O modelo de *weaving* cumpre esta tarefa ao especificar *links* entre os dados da rede elétrica representados no PIM e os dados equivalentes representados em um PDM. A razão pela qual mais de um PDM pode ser utilizado é que uma aplicação de *Smart Grids* pode fazer referência a mais de um modelo dentro do contexto de *Smart Grids*, por exemplo, um modelo que representa transformadores e um outro modelo que representa a topologia da rede elétrica.

O FMDE4SGRID possui dois níveis de modelos específicos de plataforma, conforme o que foi inicialmente proposto por Almeida et al. (2004) e Siegel (2014): O PSM Abstrato e o PSM Concreto.

O *PSM Abstrato* (parte central da Figura 4.2) é gerado automaticamente a partir de um Motor de Transformação que executa a "*Definição de Transformação A*". Esta transformação de modelos tem como entrada o PIM, um ou mais PDMs e o *weaving*. O *PSM Abstrato* corresponde ao modelo da aplicação que é específico para uma plataforma, mas que não é uma implementação concreta desta plataforma. O metamodelo da Plataforma Abstrata deve ser fornecido ao se implementar o FMDE4SGRID para o desenvolvimento de *software* visando uma plataforma específica.

O *PSM Concreto* (parte central inferior da Figura 4.2) representa a implementação concreta da aplicação em desenvolvimento em uma plataforma específica. Em geral, o PSM Concreto representa a implementação do sistema em uma linguagem de programação específica, como Java, Python ou C++. O PSM Concreto é gerado por um motor de transformação que executa a "*Definição de Transformação B*". Esta transformação de modelos tem como entrada o *PSM Abstrato*.

O código fonte da aplicação se encontra na parte inferior da arquitetura do FMDE4SGRID na Figura 4.2. O código fonte é gerado a partir de uma transformação de modelo-a-texto que tem como entrada o PSM Concreto e executa a "*Definição de Transformação C*".

Como apresentado na Figura 4.2, os metamodelos de *Smart Grids* e de *Weaving* são definidos independentemente da plataforma utilizada para implementar as aplicações ou do formato de serialização utilizado para representar os dados da rede elétrica. Os metamodelos do formato de serialização, da Plataforma Abstrata, da Plataforma Concreta e a gramática EBNF da Plataforma Final devem ser fornecidos na implementação do FMDE4SGRID visando uma plataforma específica. Pretende-se, com isso, manter a flexibilidade do FMDE4SGRID, permitindo que ele possa ser aplicado para diferentes arquiteturas de *software*.

Os metamodelos de *Smart Grids* e *Weaving* são definidos na próxima seção. Os demais metamodelos são definidos no capítulo 5, onde uma implementação do FMDE4SGRID visando a plataforma DDS é apresentada.

4.3 Metamodelos

Esta seção contém os metamodelos que são definidos no contexto do FMDE4SGRID. Os metamodelos de *Smart Grids* e de *Weaving* são apresentados e as suas construções e elementos são detalhados.

4.3.1 Metamodelo de *Smart Grids*

No FMDE4SGRID, o PIM é conforme o metamodelo de *Smart Grids*, apresentado na Figura 4.3. O metamodelo de *Smart Grids* tem como objetivo definir a lógica de negócio das aplicações para *Smart Grids*, ou seja, as atividades que a aplicação deve realizar para alcançar os seus objetivos. O metamodelo proposto não foi criado para ser um metamodelo definitivo para aplicações de *Smart Grids*, mas para atender a necessidade de representar como estas aplicações manipulam informações e como elas se comunicam. Além disso, é importante notar que este metamodelo representa a parte estrutural das aplicações. A parte comportamental poderá ser suportada através da extensão do metamodelo proposto.

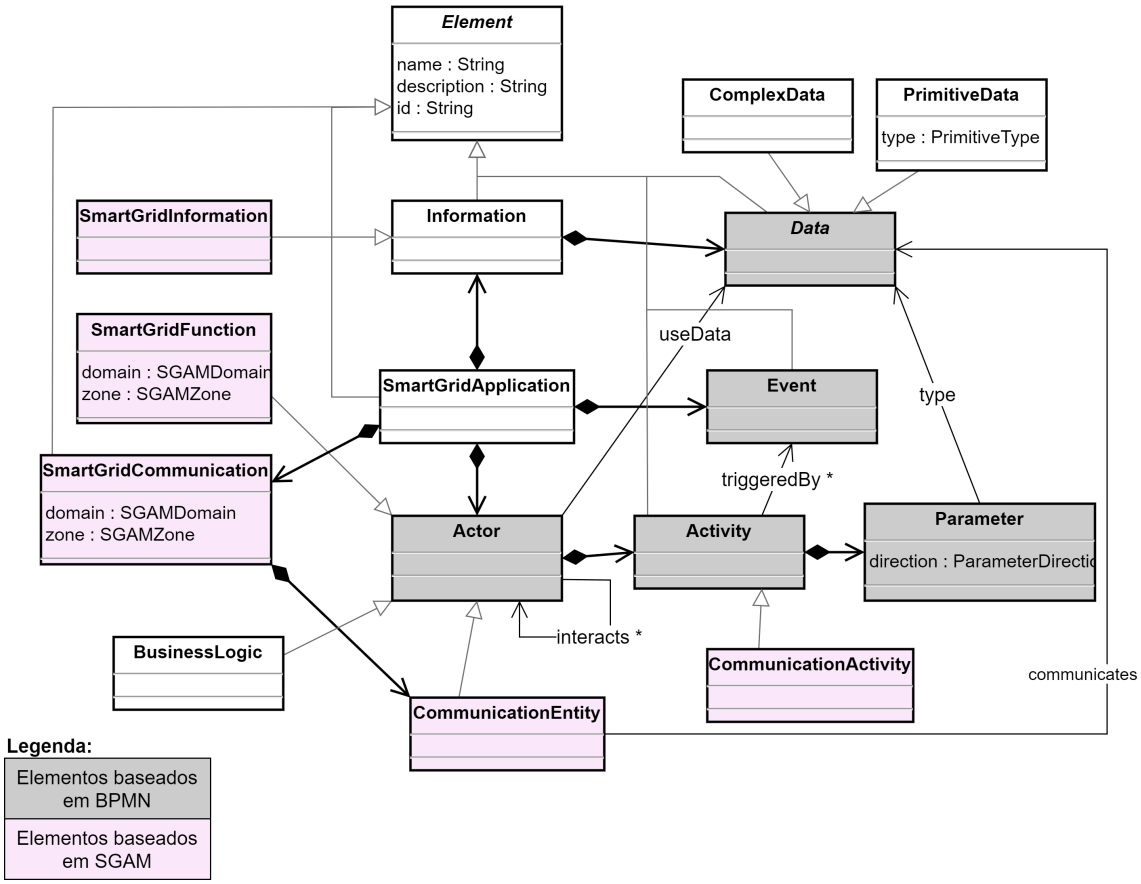
A construção do metamodelo de *Smart Grids* se dá pela extensão de conceitos básicos de modelagem de processos de negócio definidos na especificação de BPMN (OMG, 2013) para incluir os conceitos de *Smart Grids*, que são oriundos principalmente do *framework* SGAM (CEN-CENELEC-ETSI, 2012). A Figura 4.3 indica quais elementos do metamodelo são baseados em BPMN e quais elementos são baseados no SGAM. Os demais elementos foram criados para completar o sentido do metamodelo.

Uma leitura básica do metamodelo proposto de *Smart Grids* mostra que este metamodelo consiste de uma aplicação de *Smart Grids* que contém atores que realizam atividades. Estas atividades manipulam e gerenciam a informação e fazem a comunicação com outras aplicações. Os conceitos de Ator, Atividade, Informação e Comunicação são estendidos para representar as funções de *Smart Grids*. Desta forma, tem-se que *SmartGridFunction* manipula *SmartGridInformation* e interage com *SmartGridCommunication* e *CommunicationEntity* para realizar a comunicação dentro de *Smart Grids*.

Os elementos do metamodelo de *Smart Grids* são detalhados como segue:

- ***Element***: tipo abstrato que serve como base para todos os outros elementos do metamodelo, contendo nome, descrição e um *id*;

Figura 4.3 – Metamodelo proposto para representar a lógica de negócio de aplicações para *Smart Grids*.



Fonte: baseado em OMG (2013) e CEN-CENELEC-ETSI (2012).

- ***SmartGridApplication***: elemento raiz do metamodelo. Representa uma aplicação de *Smart Grids* com os seus atores, eventos, informações e configuração de comunicação;
- ***Information***: representa um container de dados, que pode ser um *SmartGridInformation*;
- ***SmartGridInformation***: um container de dados definido dentro do contexto de *Smart Grids*. Equivale à camada *Information* do SGAM;
- ***Data***: elemento que compõe uma *Information* e que representa um tipo de dado, que pode ser tipo primitivo, representado por *PrimitiveData* ou tipo complexo, representado por *ComplexData*;
- ***Actor***: entidade que pratica ações dentro do modelo de negócios. As ações são modeladas por *Activity*. Um ator pode interagir com outros atores e pode utilizar dados;
- ***SmartGridFunction***: um ator definido dentro do contexto de *Smart Grids*. Equivale à camada *Function* do SGAM. Possui os atributos '*domain*' e '*zone*' que indicam o

posicionamento desta *Function* no plano do SGAM (ver seção 2.1.4);

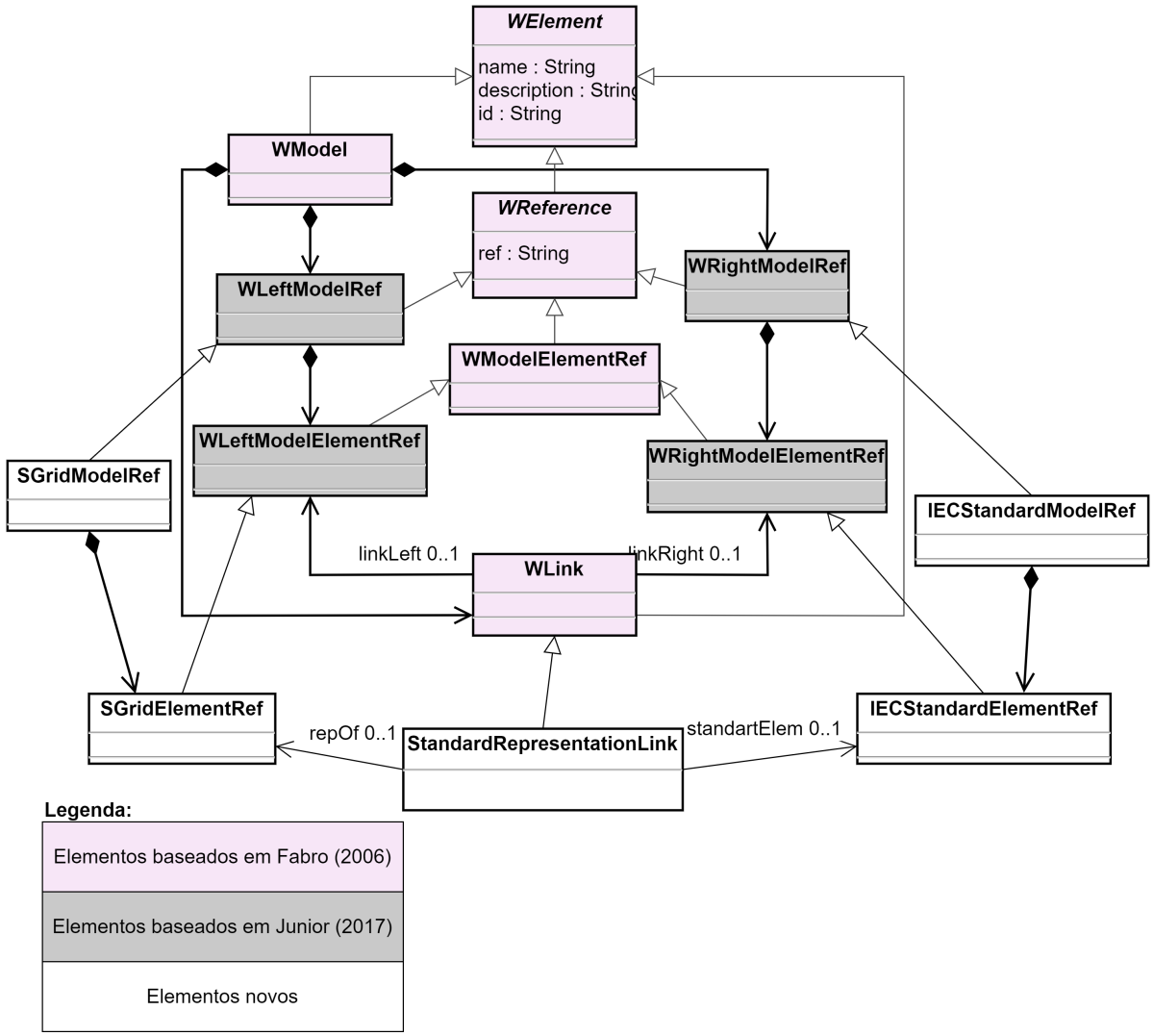
- ***SmartGridCommunication***: elemento que representa a camada *Communication* do SGAM e é composto por atores do tipo *CommunicationEntity* que realizam a comunicação em *Smart Grids*. Possui os atributos '*domain*' e '*zone*' que indicam o posicionamento desta *Communication* no SGAM;
- ***CommunicationEntity***: um ator que possui o objetivo de realizar a comunicação em *Smart Grids*;
- ***BusinessLogic***: um ator que representa o ponto de entrada do comportamento da aplicação. Para suportar a representação do comportamento, este elemento deve ser estendido e novos elementos devem ser criados;
- ***Activity***: elemento que representa uma operação que pode ser realizada por um ator. Este elemento contém parâmetros (*Parameter*) que podem ser de entrada (*in*), saída (*out*) ou entrada e saída (*inout*) e que são de um determinado tipo definido por *Data*;
- ***CommunicationActivity***: representa uma atividade especificamente voltada para realizar algum tipo de comunicação em *Smart Grids*;
- ***Event***: elemento que representa o momento em que uma determinada condição é satisfeita ou a ocorrência de um fato. Um *Event* pode ser utilizado para disparar (*trigger*) a execução de uma *Activity*.

4.3.2 Metamodelo de *Weaving*

No FMDE4SGRID, o modelo de *weaving* é conforme o metamodelo de *Weaving* apresentado na Figura 4.4. O objetivo deste metamodelo é especificar *links* entre os elementos de um PIM e os elementos de um ou mais PDMs. Os elementos do PIM que participam do *weaving* serão sempre tipos de dados (elemento do tipo *Data*). Um *link* entre um elemento do tipo *Data* no PIM e um elemento de um PDM significa que este elemento do PDM é uma representação do elemento do PIM conforme uma norma do IEC. Em outras palavras, o *link* do *weaving* representa a escolha de qual tipo de dado na norma IEC representa o tipo de dado correspondente no PIM.

A construção do metamodelo de *Weaving* proposto se deu com base nos trabalhos de Fabro et al. (2006) e Junior (2017). Os elementos aproveitados de cada trabalho são indicados na Figura 4.4. Os elementos obtidos do trabalho de Fabro et al. (2006) correspondem aos elementos centrais do metamodelo, como o elemento raiz *WModel*, o elemento que representa a referência aos modelos entrelaçados (*WReference*) e o *WLink*, que representa a associação entre os elementos dos modelos entrelaçados. O trabalho de Junior

Figura 4.4 – Metamodelo de *Weaving* proposto para representar os *links* entre PIM e PDM no FMDE4SGRID.



Fonte: baseado em Junior (2017) e Fabro et al. (2006).

(2017) utiliza elementos para incluir referências específicas aos modelos e aos elementos dos modelos da esquerda e da direita, por exemplo, *WLeftModelRef* e *WRightModelRef*. Esse tipo de referência a modelos e a elementos da esquerda e da direita dá um suporte mais explícito ao desenvolvimento baseado em Y, isto é, os elementos da esquerda correspondem ao PIM e os elementos à direita correspondem ao PDM.

O metamodelo de *Weaving* apresentado na Figura 4.4 estende os conceitos das referências à esquerda e à direita e propõe referências mais específicas para o domínio de *Smart Grids*. Assim, como é indicado na Figura 4.4, os elementos *SGridModelRef* e *SGridElementRef* fazem referência ao modelo e aos elementos do modelo de *Smart Grids* (o PIM no FMDE4SGRID), enquanto os elementos *IECStandardModelRef* e *IECStandardElementRef* fazem referência ao modelo e aos elementos do modelo conforme padrão do IEC (o PDM no FMDE4SGRID). O elemento *StandardRepresentationLink*

indica que um *IECStandardElementRef* é a representação conforme um padrão do IEC de um *SGridElementRef*.

Os elementos do metamodelo de *Weaving* proposto são descritos individualmente a seguir:

- ***WElement***: serve como base para todos os outros elementos do metamodelo, contendo atributos de nome, descrição e um *id*;
- ***WModel***: elemento raiz deste metamodelo. Representa um modelo de *weaving* que contém referências a modelos da esquerda e da direita e contém *links* entre estes modelos;
- ***WReference***: elemento base para todos os elementos do metamodelo que fazem referências aos modelos entrelaçados. O atributo '*ref*' é uma *String* que contém um identificador único para um dado modelo ou elemento de modelo;
- ***WModelElementRef***: representa uma referência a um elemento de um modelo entrelaçado;
- ***WLink***: representa uma associação entre um elemento de um modelo à esquerda (*WLeftModelElementRef*) e um elemento de um modelo à direita (*WRightModelElementRef*);
- ***WLeftModelRef***: representa uma referência a um modelo à esquerda. Um modelo à esquerda, dentro de uma abordagem baseada em Y, corresponde ao PIM;
- ***WRightModelRef***: representa uma referência a um modelo à direita. Um modelo à direita, dentro de uma abordagem baseada em Y, corresponde ao PDM;
- ***WLeftModelElementRef***: representa uma referência a um elemento de um modelo à esquerda. Este elemento compõe o *WLeftModelRef*;
- ***WRightModelElementRef***: representa uma referência a um elemento de um modelo à direita. Este elemento compõe o *WRightModelRef*;
- ***SGridModelRef***: este elemento estende o conceito de *WLeftModelRef* para representar uma referência a um modelo de *Smart Grids*;
- ***SGridElementRef***: este elemento, contido por *SGridModelRef*, estende o conceito de *WLeftModelElementRef* para representar um elemento de um modelo de *Smart Grids*;
- ***IECStandardModelRef***: este elemento estende o conceito de *WRightModelRef* para representar uma referência a um modelo de um padrão do IEC;

- ***IECStandardElementRef***: este elemento, contido por *IECStandardModelRef*, estende o conceito de *WRightModelElementRef* para representar a referência a um elemento de um modelo conforme um padrão do IEC;
- ***StandardRepresentationLink***: estende o conceito de *WLink* para representar uma associação que indica que um elemento de um *SGridModelRef* é representado por um elemento de um modelo conforme um padrão do IEC (*IECStandardElementRef*).

4.4 Metodologia para a utilização do FMDE4SGRID

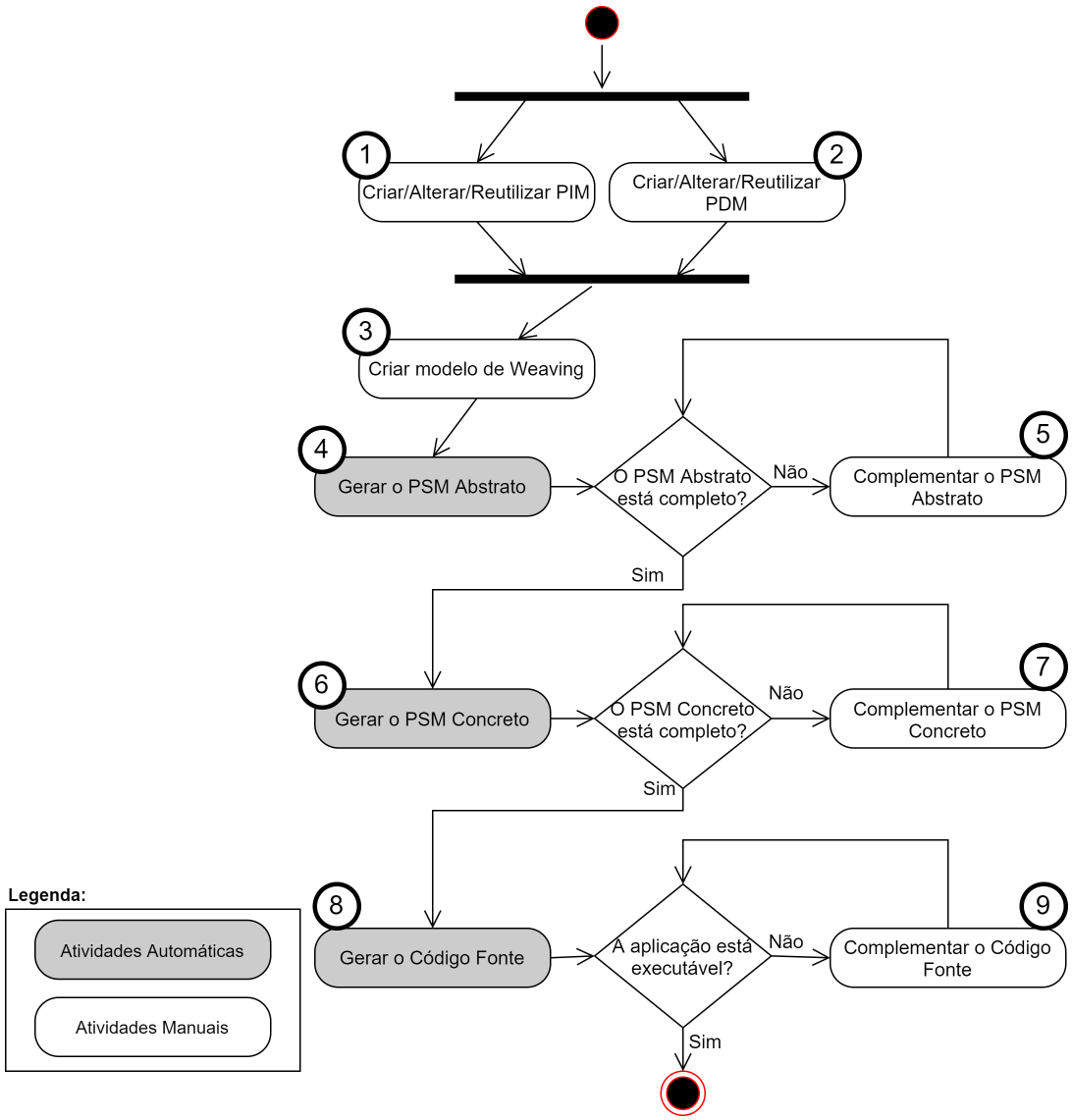
Esta seção contém uma metodologia descrita em forma de diagrama de atividades que tem como objetivo guiar a utilização do FMDE4SGRID.

A metodologia que é recomendada neste manuscrito para a correta aplicação do FMDE4SGRID é apresentada na Figura 4.5.

O passo-a-passo para a aplicação do processo da Figura 4.5 é descrito a seguir, seguindo a numeração das atividades como está ilustrado na Figura 4.5.

1. **Criar/Alterar/Reutilizar o PIM.** O PIM deve ser fornecido como um modelo conforme o metamodelo de *Smart Grids*, apresentado na subseção 4.3.1;
2. **Criar/Alterar/Reutilizar os PDMs.** Os PDMs devem ser modelos que descrevem a rede elétrica conforme uma norma do IEC, como o IEC 61850 ou IEC 61970 em um formato a ser escolhido pelo desenvolvedor, como XML, XSD, JSON ou outro formato disponível;
3. **Criar o modelo de *weaving***. O modelo de *weaving* é criado a partir dos modelos PIM e PDM. Uma ferramenta pode ser fornecida para carregar os elementos de PIM e PDM no modelo de *weaving* automaticamente. No entanto, a especificação dos *links* entre os elementos deve ser feita manualmente;
4. **Gerar o PSM Abstrato.** Uma ferramenta de transformação de modelos deve ser utilizada para transformar os modelos de entrada (PIM, PDMs e *weaving*) no PSM Abstrato automaticamente;
5. **Complementar o PSM Abstrato.** Caso o PSM Abstrato gerado automaticamente esteja incompleto ou há algum elemento do modelo que necessite ser modificado ou criado, o PSM Abstrato deve ser complementado manualmente, através de uma ferramenta para a edição do modelo;
6. **Gerar o PSM Concreto.** Uma ferramenta de transformação de modelos deve ser utilizada para gerar automaticamente o PSM Concreto a partir do PSM Abstrato, caso este esteja completo;

Figura 4.5 – Metodologia recomendada para a aplicação do FMDE4SGRID.



Fonte: baseado em Junior (2017)

7. **Complementar o PSM Concreto.** Caso o PSM Concreto esteja incompleto ou necessite de alterações, estas modificações e os complementos necessários são feitos manualmente, através de uma ferramenta para a edição dos modelos;

8. **Gerar o código fonte.** O código fonte da aplicação é gerado automaticamente com a utilização de uma ferramenta de transformação de modelo-a-texto. A entrada desta transformação é o PSM Concreto, caso este esteja completo;

9. **Complementar o código fonte.** O código fonte da aplicação deve ser complementado até que a aplicação seja executável e possa ser testada.

Os passo 4, o passo 6 e o passo 8 dão agilidade ao processo de desenvolvimento de *software* por utilizar ferramentas que geram os artefatos de *software* automaticamente.

No entanto, alterações manuais nos modelos e no código geralmente são necessárias ao longo de todo o processo de utilização do FMDE4SGRID, além das etapas que são puramente manuais, como as etapas de 1 a 3. Assim sendo, considera-se que a abordagem para o desenvolvimento de *software* que envolve o FMDE4SGRID é uma abordagem semiautomática.

4.5 Síntese

Neste capítulo, o "*Framework* baseado em MDE e *weaving* de modelos para suportar a atividade de desenvolvimento de aplicações de *software* para o domínio de *Smart Grids* - FMDE4SGRID" foi apresentado.

A arquitetura proposta do FMDE4SGRID foi concebida conforme MDE e o desenvolvimento baseado em Y. A lógica de negócio da aplicação é capturada pelo PIM, que é conforme o metamodelo de *Smart Grids*. O modelo da rede elétrica, contendo topologia da rede, equipamentos, entre outras informações é capturado por um ou mais PDMs que são baseados em uma norma do IEC para representação de dados da rede elétrica, como a norma CIM. O modelo de *weaving* captura as associações entre os tipos de dados do PIM e os tipos de dados dos PDMs. O objetivo é dar aos tipos de dados utilizados pela aplicação uma representação conforme um padrão do IEC para, desta forma, garantir a interoperabilidade de dados.

O metamodelo de *Smart Grids* foi apresentado e consiste em uma extensão dos conceitos básicos de BPMN (OMG, 2013) para incluir os conceitos de *Smart Grids*, que são baseados no *framework* SGAM. Um metamodelo de *Weaving* também foi proposto para dar suporte a esta abordagem. O metamodelo de *Weaving* proposto estende os metamodelos de Junior (2017) e Fabro et al. (2006) para incluir elementos que indicam de maneira explícita que os tipos de dados de um modelo de *Smart Grids* são representados por tipos de dados de um modelo conforme um padrão do IEC.

O FMDE4SGRID contém uma transformação de modelo-a-modelo para gerar um PSM de plataforma abstrata a partir dos modelos de entrada (PIM, PDMs e *weaving*). Também é especificado que uma transformação de modelo-a-modelo deve ser utilizada para gerar um PSM concreto a partir do PSM Abstrato. Por fim, o código fonte da aplicação deve ser gerado por uma transformação de modelo-a-texto.

O processo de utilização do FMDE4SGRID envolve a utilização de etapas automáticas, que podem agilizar o processo de desenvolvimento de *software*. Porém, a intervenção manual do desenvolvedor nos artefatos de desenvolvimento (por exemplo, os modelos, os arquivos de configuração e o código fonte) se faz necessária para garantir a consistência e correção dos modelos e para tornar o código fonte executável.

5 Implementação do FMDE4SGRID para suportar a integração de aplicações em *Smart Grids*

Este capítulo contém uma implementação do FMDE4SGRID para a plataforma DDS, com o objetivo de integrar as aplicações de *Smart Grids* com base na norma CIM. O DDS é uma especificação de *middleware* para distribuição de dados apresentado na Seção 2.3.

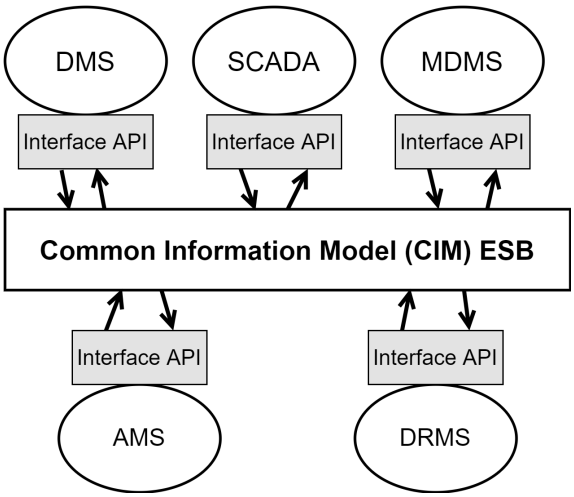
Este capítulo está organizado da seguinte forma: A Seção 5.1 contém uma arquitetura de *software* do tipo ESB (*Enterprise Service Bus*) baseada em norma CIM para integrar aplicações de *Smart Grids* e garantir a interoperabilidade. A Seção 5.2 traz a implementação do FMDE4SGRID para desenvolver aplicações de *Smart Grids* para a plataforma DDS. Os metamodelos específicos de plataforma fornecidos são apresentados na Seção 5.3 e as definições de transformação de modelos são apresentadas na Seção 5.4. A Seção 5.5 contém a descrição de uma prototipagem do FMDE4SGRID em IDE-Eclipse. Por fim, a Seção 5.6 traz uma síntese deste capítulo.

5.1 Arquitetura de *software* baseada em ESB e CIM para a integração de aplicações de *Smart Grids*

Alguns autores, como Souvent et al. (2019), Shi et al. (2014), Kim et al. (2017) e Neis et al. (2019), apresentam arquiteturas de *software* baseadas em *middleware* e norma CIM para integrar as aplicações de *Smart Gris*. A abordagem de *middleware* tem sido apontada por estes estudos como promissora para a implementação de *Smart Grids*, pois ela auxilia o gerenciamento da complexidade de comunicação de dados entre aplicações. Ao invés de cada aplicação implementar um serviço de comunicação diferente para trocar dados com cada uma das outras aplicações do sistema, um único serviço de comunicação pode ser utilizado como um "barramento de mensagens". Denota-se este barramento de mensagens como *Enterprise Service Bus* (ESB) (SOUVENT et al., 2019). Na Figura 5.1, uma arquitetura de *software* do tipo ESB baseada em CIM é apresentada.

O CIM ESB da Figura 5.1 é um *middleware* que realiza a comunicação entre as aplicações utilizando o modelo CIM como a base semântica dos dados que são compartilhados. Cada aplicação utiliza os serviços do *middleware* por meio de uma API, que é única para cada aplicação.

Figura 5.1 – Arquitetura de *sofware* do tipo ESB baseada em CIM proposta para a integração de aplicações de *Smart Grids*.



Fonte: baseado em Souvent et al. (2019) e Neis et al. (2019).

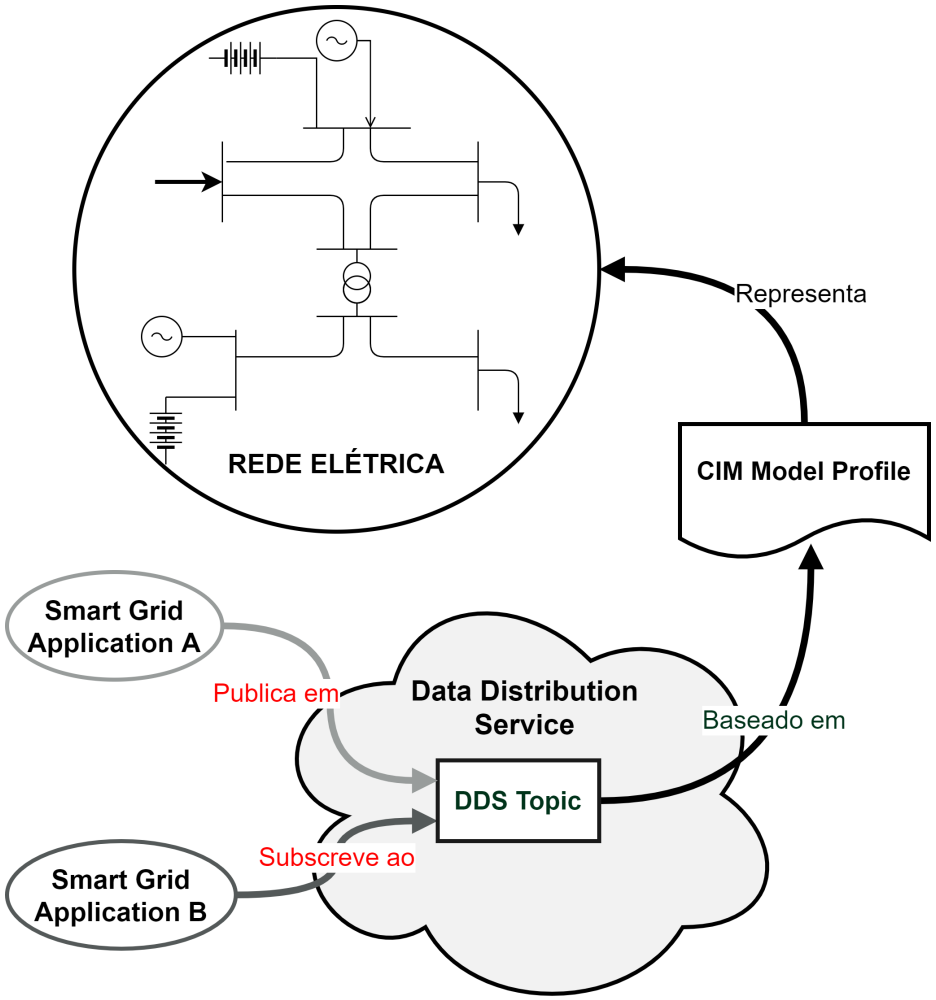
5.1.1 Implementação de um CIM ESB com o *middleware* DDS

O *middleware* DDS (OMG, 2015) foi escolhido, dentro do escopo deste trabalho, como a plataforma de implementação para a arquitetura de *software* da Figura 5.1. A centricidade em dados, que é uma característica fundamental do DDS, foi uma das principais justificativas para esta escolha. Essa característica é particularmente relevante no domínio de *Smart Grids*, pois tira da camada de aplicação a responsabilidade de gerenciar a comunicação e o estado dos dados compartilhados. Além disso, o *middleware* DDS mantém uma estrutura de dados que deve ser a mesma para todas as aplicações.

O esquema da Figura 5.2 ilustra como o DDS pode ser utilizado para realizar a integração de aplicações baseada no modelo CIM. Neste esquema, as aplicações de *Smart Grids* podem publicar dados em um tópico de DDS ou podem subscrever a este tópico para então receber os dados que são publicados nele. O tópico de DDS é baseado em um modelo conforme CIM. Os tipos de dados que as aplicações publicam ou consomem em um tópico é definido em tempo de projeto e são disponibilizados para as aplicações através de uma API na linguagem de programação utilizada.

A implementação de DDS utilizada neste trabalho é o *OpenDDS* (OCI, 2020). No *OpenDDS*, os tipos de dados, que serão a base para os tópicos, são previamente definidos em arquivos IDL. Para que os tipos de dados definidos em IDL estejam em conformidade com a norma CIM, é necessário que um perfil de CIM seja criado, representando, por exemplo, a topologia de uma rede elétrica, e que este perfil de CIM seja convertido para o formato IDL. O *OpenDDS* utiliza um compilador de IDL para gerar o código, nas linguagens C++ e Java, referente aos objetos definidos nos arquivos IDL e da API de DDS específica para os objetos criados (OCI, 2020).

Figura 5.2 – Esquema de utilização do DDS para a integração de aplicações de *Smart Grids* com base no CIM.



Fonte: baseado em OMG (2021).

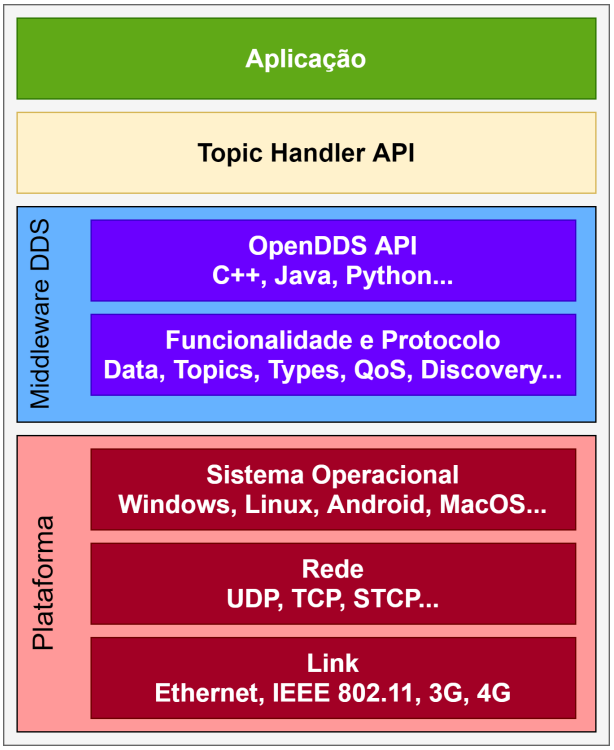
A API gerada pelo *OpenDDS* contém todas as classes e interfaces definidas na especificação de DDS, como *Publisher*, *Subscriber*, *Topic*, *DataReader* e *DataWriter*, além dos tipos de dados específicos que correspondem aos tipos definidos nos arquivos IDL.

5.1.2 *Topic Handler*: Uma camada de *software* para abstrair a API de *OpenDDS*

A API gerada pelo *OpenDDS* possui muitas classes e interfaces. Um tempo considerável deve ser dedicado ao estudo desta API e de como utilizá-la. As funcionalidades fornecidas através da API de *OpenDDS* podem ser utilizadas para modelar vários tipos de comportamento e mecanismos diferentes para realizar o compartilhamento de dados no *middleware* DDS, o que requer um certo tempo de treinamento para os desenvolvedores e, ao mesmo tempo, pode ocasionar implementações ineficientes para problemas conhecidos.

Desta forma, uma camada de *software* chamada *Topic Handler* é proposta neste trabalho. O *TopicHandler* tem como objetivo tornar a API de OpenDDS transparente à camada de aplicação que utiliza os serviços de DDS. A Figura 5.3 mostra onde o *Topic Handler* é colocado em relação à API de *OpenDDS* e à aplicação que utiliza os serviços de *middleware*.

Figura 5.3 – Pilha de protocolos e APIs que mostra onde o *Topic Handler* é definido.



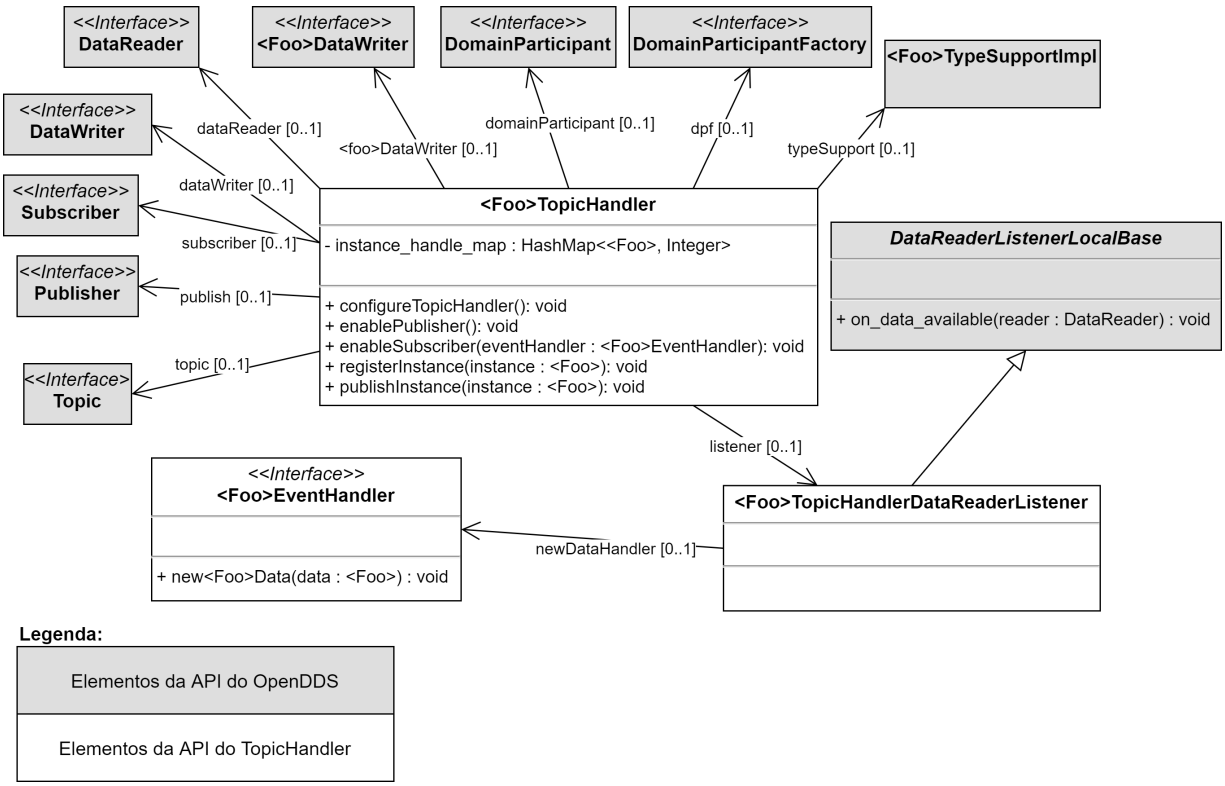
Fonte: baseado em OMG (2021).

O *Topic Handler* é uma API que utiliza a API de *OpenDDS* para realizar comportamentos básicos relacionados aos tópicos de DDS, ou seja, uma aplicação pode utilizar os métodos do *Topic Handler* para interagir com um tópico específico de DDS. Como um tópico do DDS é definido para um tipo específico de dado, o *Topic Handler* em particular também deve ser criado para lidar com um tipo específico.

A Figura 5.4 apresenta a definição da API de *Topic Handler*. O termo "<Foo>", utilizado no diagrama da Figura 5.4, refere-se ao nome do tipo específico referente ao tópico de DDS que o *Topic Handler* utiliza. Por exemplo, se o nome do tipo for *String*, então basta substituir *<Foo>* por *String*.

O *Topic Handler* define as classes *<Foo>TopicHandler* e *<Foo>TopicHandler-DataReaderListener* e a interface *<Foo>EventHandler*. A classe *<Foo>TopicHandler* acessa vários elementos da API de *OpenDDS* e oferece alguns métodos úteis para interagir com o *middleware* DDS. Os métodos definidos na classe *<Foo>TopicHandler* são descritos a seguir:

Figura 5.4 – Diagrama de classes que define a API de *Topic Handler*.



Fonte: acervo do autor.

- *configureTopicHandler()*: este método inicializa alguns parâmetros importantes do *OpenDDS*, como o *DomainParticipant*, o *TypeSupport* e o *Topic*. Este método deve ser chamado antes de qualquer outro método do *TopicHandler*;
- *enablePublisher()*: este método inicializa todos os objetos necessários para a publicação de dados no *OpenDDS*, como o *Publisher* e o *DataWriter*;
- *enableSubscriber()*: este método inicializa todos os objetos necessários para a subscrição ao tópico de DDS que o *Topic Handler* suporta;
- *registerInstance()*: registra uma instância do tipo *<Foo>* no *OpenDDS*. Uma instância só pode ser publicada após o seu registro ter sido efetuado;
- *publishInstance()*: publica uma instância registrada no *OpenDDS*.

A interface *<Foo>EventHandler* lida com eventos assíncronos relacionados ao *OpenDDS*, por exemplo, a chegada de novos dados em um tópico. A aplicação do usuário deve implementar esta interface para que a aplicação seja notificada pelo *Topic Handler* sobre as atualizações do tópico no *OpenDDS*.

A classe *<Foo>TopicHandlerDataReaderListener* é de uso interno do *Topic Handler*. Esta classe estende uma classe da API de *OpenDDS* chamada *DataReaderListenerLocalBase*,

que é um *listener* para as atualizações referentes ao *DataReader*. Esta classe contém vários métodos para os diferentes tipos de eventos que ocorrem para o *DataReader*. Por simplicidade, apenas o método *on_data_available()* é representado na Figura 5.4. A classe *<Foo>TopicHandlerDataReaderListener* é configurada pela classe *<Foo>TopicHandler* e notifica o *EventHandler* de que novos dados estão disponíveis.

Como o *TopicHandler* possui um conjunto limitado de métodos em relação à API completa de DDS, o seu uso pode restrigir o uso de todas as funcionalidades do DDS. Por esta razão, o *Topic Handler* pode ser estendido para incluir mais comportamentos e, desta forma, aumentar a sua flexibilidade. O código fonte completo do *TopicHandler* está disponível no Apêndice A deste manuscrito.

5.2 Implementação do FMDE4SGRID para a plataforma *OpenDDS*

Esta seção apresenta a implementação do FMDE4SGRID para auxiliar no desenvolvimento de aplicações para *Smart Grids* visando a plataforma *OpenDDS*. O objetivo é implementar aplicações conforme a arquitetura da Figura 5.1, apresentada na seção 5.1.

Como já discutido no Capítulo 4, a implementação do FMDE4SGRID requer que os metamodelos das plataformas específicas e as transformações de modelos sejam definidas. A Figura 5.5 apresenta a arquitetura do FMDE4SGRID modificada para incluir a identificação dos metamodelos e modelos específicos de plataforma.

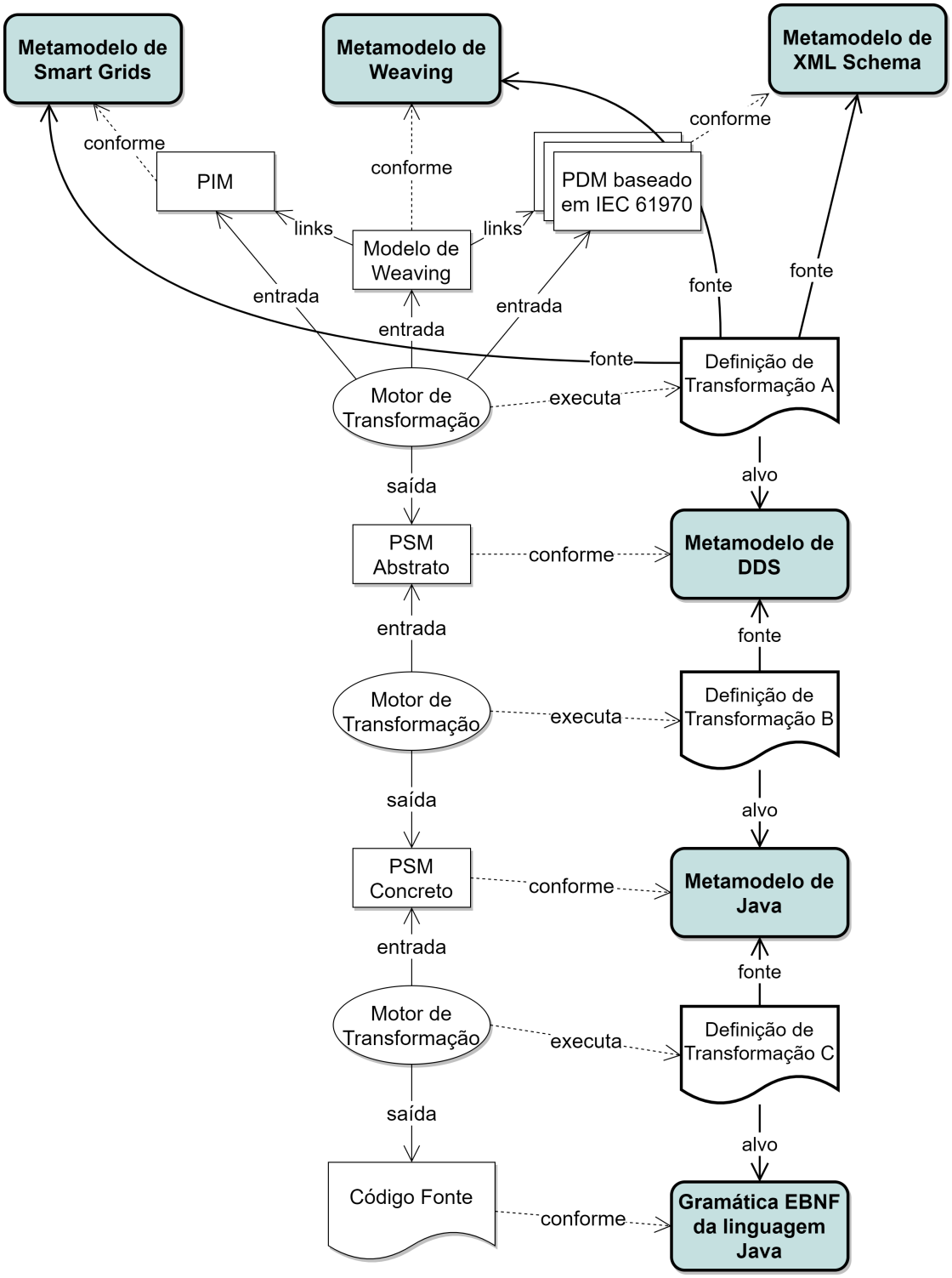
O formato de serialização escolhido para representar os modelos de PDM foi o *XML Schema* (XSD). Existe uma ferramenta chamada *CIMTool*[1], que permite a criação de perfis de CIM e permite exportar os perfis em diferentes formatos. Um dos formatos de saída do *CIMTool* é o XSD.

O PSM Abstrato é conforme o metamodelo de DDS, que contém não apenas os conceitos de DDS e de orientação à objetos, mas contém também a definição do *Topic Handler*, apresentado na seção 5.1. O *OpenDDS*, até a sua versão mais recente (versão 3.16), tem suporte nativo à linguagem C++ e fornece total suporte à linguagem Java através de *bindings* JNI. Sendo assim, o PSM Concreto é conforme a linguagem Java.

Naturalmente, a linguagem Java é utilizada para a implementação do código fonte e, desta forma, o FMDE4SGRID faz uso da gramática EBNF da linguagem Java.

[1] O *CimTool* é uma ferrramenta *opensource* disponível para *download* no site <https://wiki.cimtool.org/> (Último acesso em 13/06/2021). O código fonte do *CIMTool* pode ser encontrado no repositório disponível no endereço <https://github.com/arnolddevos/CIMTool> (Último acesso em 13/06/2021).

Figura 5.5 – Arquitetura específica do FMDE4SGRID para a plataforma DDS.



Fonte: baseado em Junior (2017) e Stefanello (2017).

5.3 Metamodelos específicos de plataforma

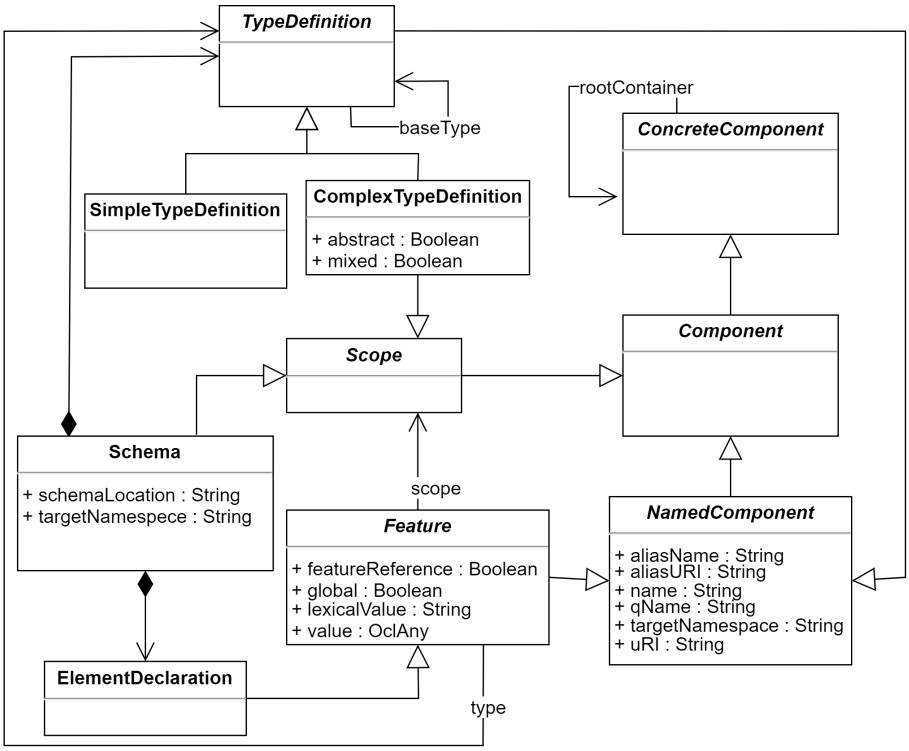
Os metamodelos específicos de plataforma propostos para compor o FMDE4SGRID, conforme a arquitetura da Figura 5.5, são apresentados nesta seção. Os metamodelos de

XML *Schema*, DDS e Java são apresentados.

5.3.1 Metamodelo de XML Schema

A Figura 5.6 apresenta um fragmento do metamodelo de XML *Schema*. O formato XML *Schema* é utilizado no FMDE4SGRID para descrever os PDMs, que consistem em perfis da norma CIM para sistemas de energia elétrica.

Figura 5.6 – Fragmento do metamodelo de XML *Schema*.



Fonte: baseado em W3C (2012).

O metamodelo da Figura 5.6 consiste na definição de um *Schema* que contém definições de tipos de dados (elemento *TypeDefinition*), que podem ser tipos simples (*SimpleTypeDefinition*) ou tipos complexos (*ComplexTypeDefinition*); *Schema* também contém a declaração de elementos (*ElementDeclaration*), que são características (*Feature*) definidas dentro de um escopo (*Scope*), que pode ser um *Schema* ou um *ComplexTypeDefinition*.

5.3.2 Metamodelo de DDS

A Figura 5.7 apresenta o metamodelo de DDS proposto. Dentro do FMDE4SGRID, o PSM Abstrato é conforme o metamodelo de DDS.

O metamodelo da Figura 5.7 tem como objetivo representar três aspectos importantes das aplicações desenvolvidas para DDS. O primeiro aspecto é o *software* propriamente dito,

que é contido pelo elemento *DSoftwarePackage*. O elemento *DSoftwarePackage* contém os tipos de dados correspondentes ao paradigma de orientação à objetos, como *DClass*, *DInterface*, *DMethod*, *DProperty* e *DParameter*. Além disso, *DSoftwarePackage* contém os tipos de dados correpsondentes ao *TopicHandler*, que representa uma camada de *software* que abstrai a API de DDS.

O segundo aspecto relevante que o metamodelo de DDS apresenta é a configuração de transporte de dados e de descoberta utilizada pela implementação de DDS (*OpenDDS*) para realizar a comunicação entre as aplicações. O elemento *DAplicationConfig*, no centro da Figura 5.7 contém as configurações de domínio (*DDomainConfig*), de transporte (*DTransportConfig*) e de descoberta (*DDiscoveryConfig*), conforme é detalhado em OCI (2020).

Por fim, o terceiro aspecto relevante capturado pelo metamodelo de DDS é a descrição dos tipos de dados que irão corresponder aos tópicos de DDS. A descrição destes tipos é feita em IDL e, desta forma, um fragmento do metamodelo de IDL foi incorporado ao metamodelo de DDS, como pode ser observado na parte à direita da Figura 5.7. Destaca-se, nesta parte do metamodelo, que o módulo IDL (*IDLModule*) é composto de tipos (*IDLType*) que podem ser *structs* (*IDLStructs*). Uma *struct* representa um tópico de DDS se o atributo *isDDSTopic* for verdadeiro. Uma *struct* contém membros (*IDLStructMember*) que possuem um determinado tipo. Na plataforma *OpenDDS*, recomenda-se utilizar um membro da *struct* como chave do objeto para identificar diferentes instâncias dentro de um mesmo tópico. O atributo *isDDSKey* do elemento *IDLStructMember* serve a este propósito.

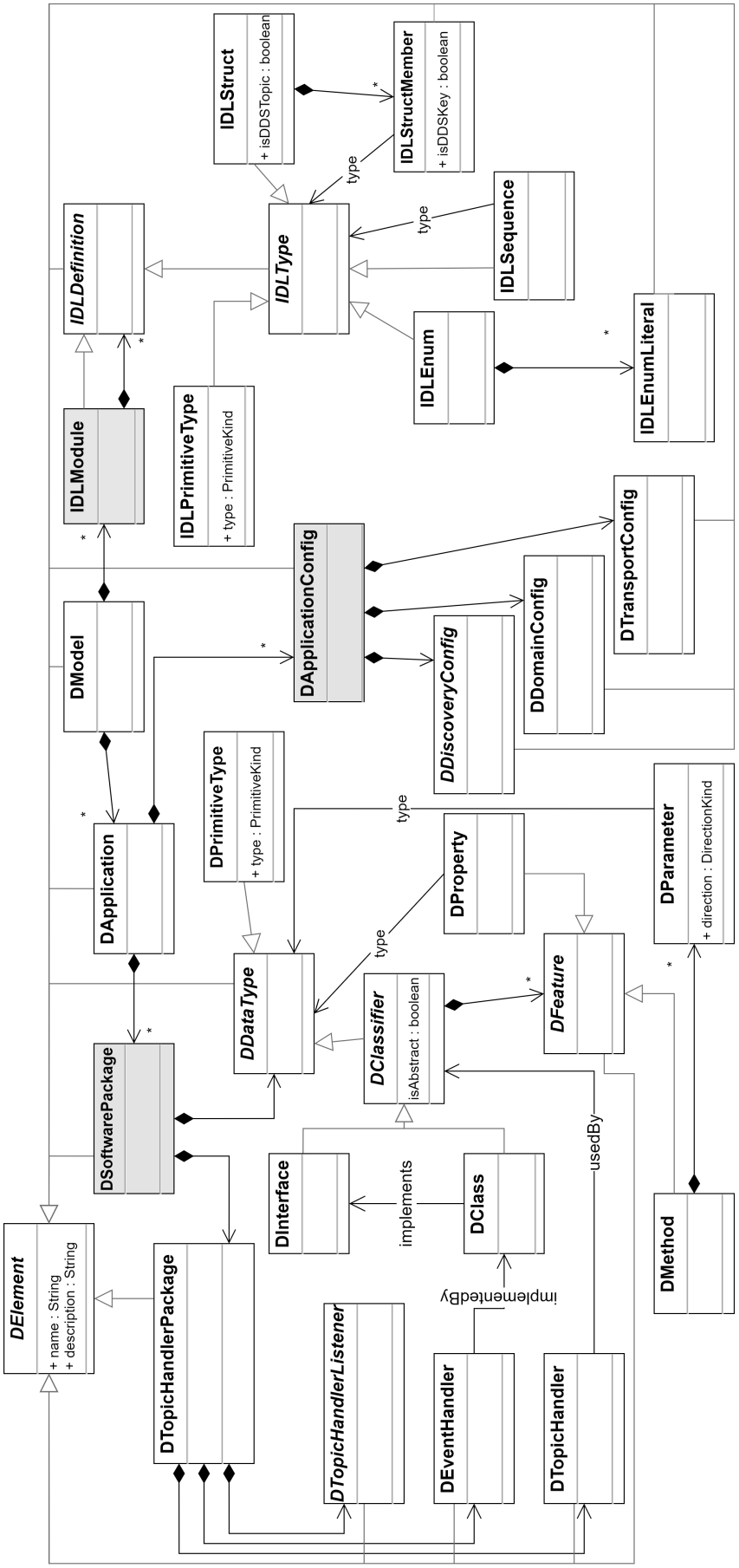


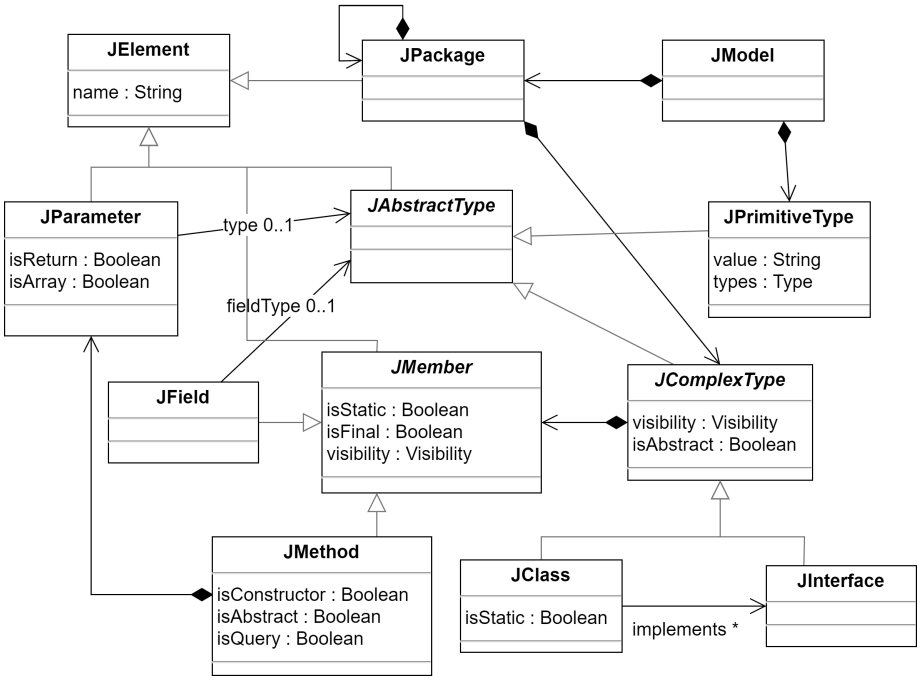
Figura 5.7 – Fragmento do metamodelo de DDS.

Fonte: baseado em OCI (2020).

5.3.3 Metamodelo de Java

A Figura 5.8 mostra um fragmento do metamodelo de Java utilizado nesta implementação do FMDE4SGRID. Este metamodelo foi reutilizado do trabalho de Junior (2017).

Figura 5.8 – Fragmento do metamodelo de Java utilizado nesta implementação do FMDE4SGRID.



Fonte: Junior (2017).

O metamodelo da Figura 5.8 contém as definições básicas da linguagem Java. O elemento raiz, *JModel*, contém os tipos primitivos (*JPrimitiveType*), e os pacotes (*JPackage*). O elemento *JPackage* contém os tipos complexos (*JComplexType*) de Java, que podem ser *JClass* ou *JInterface*. *JComplexType* contém membros que podem ser *JField* ou *JMethod* que, por sua vez, contém *JParameter*. *JField* e *JParameter* têm seus tipos especificados por *JAbstractType*.

5.4 Definições de transformação

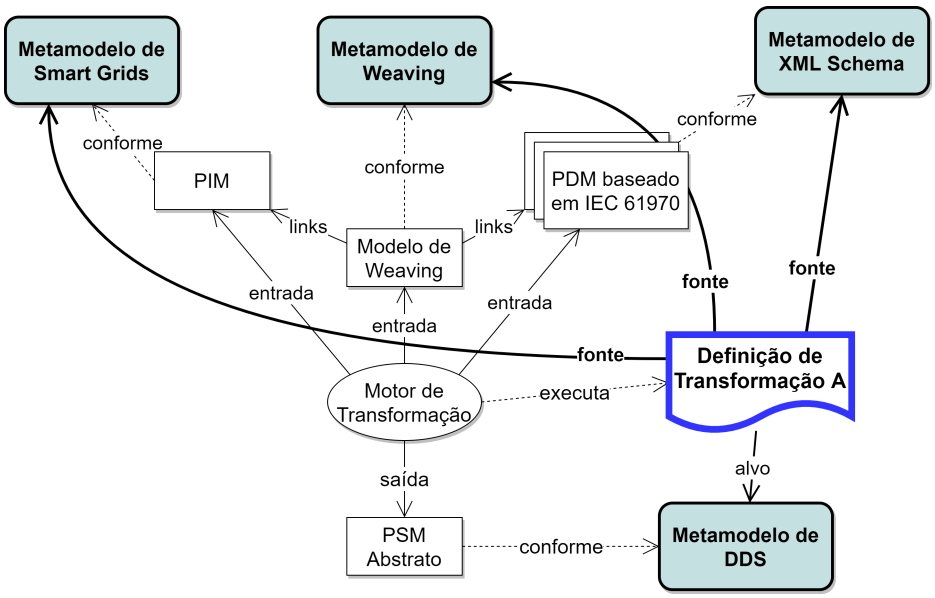
Esta seção apresenta as definições de transformação utilizadas na implementação do FMDE4SGRID. Como mostra a Figura 5.5, são necessárias três definições de transformação de modelos para apoiar esta abordagem. Duas destas definições (A e B) correspondem a transformações de modelo-a-modelo, e a Definição de Transformação C descreve uma transformação de modelo-a-texto. As definições de transformação são apresentadas nas próximas subseções.

5.4.1 Definição de Transformação A - PIM, PDMs e *Weaving* para PSM Abstrato

A Definição de Transformação A, conforme mostra a Figura 5.9, determina como os elementos dos modelos de entrada (PIM, PDMs e *weaving*), descritos pelos metamodelos de entrada, são transformados em um modelo de saída, que neste caso é o PSM Abstrato que é conforme o metamodelo de DDS.

Para compor esta definição de transformação, as regras de transformação devem ser elaboradas a partir do mapeamento entre os elementos dos metamodelos de entrada (*Smart Grid*, XML *Schema* e *Weaving*) e os elementos do metamodelo de saída (DDS).

Figura 5.9 – Seção do FMDE4SGRID que corresponde à Definição de Transformação A.



Fonte: baseado em Junior (2017).

Para facilitar o entendimento, a construção desta definição de transformação é dividida em três partes: o mapeamento entre os elementos dos metamodelos de *Smart Grids* e DDS; o mapeamento entre os elementos dos metamodelos de XML *Schema* e DDS; e, por fim, o papel que o modelo de *weaving* possui nesta transformação é descrito.

Mapeamento entre elementos do metamodelo de *Smart Grids* e o metamodelo de DDS

A Tabela 5.1 apresenta o mapeamento entre os elementos do metamodelo de *Smart Grids* e os elementos do metamodelo de DDS.

Os elementos do metamodelo de *Smart Grids*, em geral, são mapeados para elementos de DDS contidos por *DSoftwarePackage*, que visam representar o *software* que implementa a lógica de negócio da aplicação, descrita pelo PIM.

Tabela 5.1 – Correspondências entre os elementos dos metamodelo de *Smart Grids* e DDS.

Elemento de *Smart Grid*	Elemento de DDS
SmartGridApplication	*DApplication*
Actor	*DClass*
SmartGridFunction	*DClass*
BusinessLogic	*DClass*
Activity	*DMethod*
Parameter	*DParameter*
PrimitiveData	*DPrimitiveType*
ComplexData	*DClass*
SmartGridCommunication	*DApplicationConfig*
CommunicationEntity	*DTopicHandlerPackage*
CommunicationEntity	*DInterface*

Nem todos os elementos do metamodelo de DDS possuem um correspondente direto no metamodelo de *Smart Grids*. Logo, ao gerar o PSM Abstrato, a transformação de modelos deve criar alguns elementos necessários. Os elementos contidos por *DTopicHandlerPackage*, como o *DTopicHandler* por exemplo, não possuem um correspondente direto no metamodelo de *Smart Grids* e, portanto, precisam ser criados. A criação destes elementos pode ser automática (como parte da transformação) ou manual.

O elemento *CommunicationEntity* de *Smart Grids* é mapeado para dois elementos distintos de DDS, como mostra as duas últimas linhas da Tabela 5.1. Na transformação, um elemento do tipo *CommunicationEntity* é transformado para um elemento do tipo *DTopicHandlerPackage* quando o *CommunicationEntity* é contido por *SmartGridCommunication*, o que indica que a comunicação é realizada dentro de *Smart Grids* e, por tanto, o DDS será utilizado para realizar esta comunicação. Caso o elemento do tipo *CommunicationEntity* não seja contido por *SmartGridCommunication*, ele é mapeado para *DInterface*.

Mapeamento entre elementos do metamodelo de XML *Schema* e o metamodelo de DDS

A Tabela tal mostra o mapeamento entre os elementos do metamodelo de XML *Schema* (que descreve os PDMs) e os elementos do metamodelo de DDS.

Tabela 5.2 – Correspondências entre os elementos dos metamodelo de XML *Schema* e DDS.

Elemento de XML *Schema*	Elemento de DDS
Schema	*IDLModule*
ComplexTypeDefinition	*IDLStruct*
ElementDeclaration	*IDLStructMember*

Os elementos dos PDMs, descritos em XML *Schema*, devem ser transformados em elementos da parte do metamodelo de DDS que corresponde aos módulos de IDL (a parte

à direita do metamodelo apresentado na Figura 5.7). Na prática, o que se faz nesta parte desta definição de transformação é um mapeamento entre XML *Schema* e IDL.

Papel do modelo de *weaving* na Definição de Transformação A

Os tipos de dados definidos no PIM (elementos do tipo *Data*) que possuem um *link* no modelo de *weaving* são transformados no elemento correspondente do PSM Abstrato com o nome do elemento do PDM, acessado através do elemento *StandardRepresentationLink* do *weaving*.

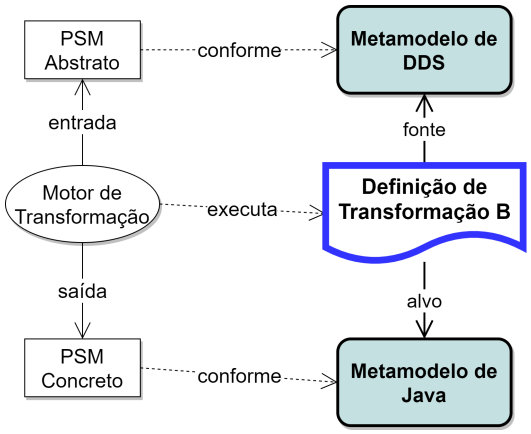
O *weaving* também deve ser utilizado para definir o nome do *TopicHandler*, que deve corresponder ao nome de um elemento do PDM que é ligado, através do *weaving*, a um *data type* que é utilizado por *CommunicationEntity* através do *binding communicates*.

Os elementos do tipo *IDLStructMember*, no PSM Abstrato, gerados a partir de elementos do tipo *ElementDeclaration*, de um PDM, devem ter o atributo *isDDSTopic* inicializado com o valor *true* caso o elemento do PDM correpondente seja referenciado por um *StandardRepresentationLink* do *weaving*.

5.4.2 Definição de Transformação B - PSM Abstrato para PSM Concreto

A Definição de Transformação B, ilustrada na Figura 5.10, especifica uma transformação que tem como entrada o PSM Abstrato - conforme o metamodelo de DDS - e como saída o PSM Concreto, que é conforme o metamodelo de Java.

Figura 5.10 – Seção do FMDE4SGRID que corresponde à Definição de Transformação B.



Fonte: baseado em Junior (2017).

Esta transformação deve levar em conta a parte de pacote de *software* (*DSoftwarePackage*) do PSM Abstrato para gerar os elementos do PSM Concreto, como classes, interfaces e métodos. Além disso, os elementos da API de DDS devem ser gerados durante a transformação. A Tabela 5.3 apresenta o mapeamento entre os elementos do metamodelo de DDS e os elementos do metamodelo Java.

Tabela 5.3 – Correspondências entre os elementos do metamodelo de DDS e os elementos do metamodelo de Java.

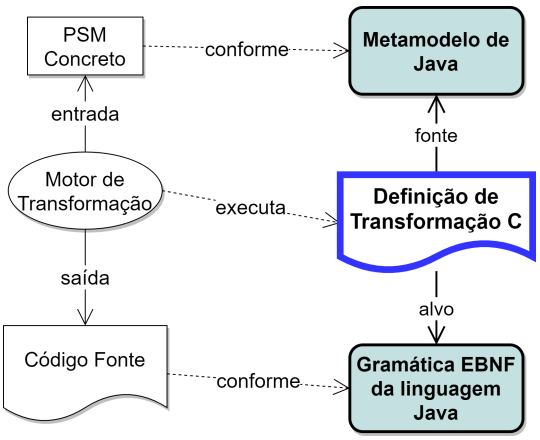
Elemento de DDS	Elemento de Java
DSoftwarePackage	*JPackage*
DClass	*JClass*
DInterface	*JInterface*
DTopicHandler	*JClass*
DTopicHandlerListener	*JClass*
DEventHandler	*JInterface*
DTopicHandlerField	*JField*
DEventHandlerOperation	*JMethod*
DMethod	*JMethod*
DParameter	*JParameter*
DProperty	*JField*

O PSM Concreto representa uma aplicação em linguagem Java. O objetivo, nesta versão do FMDE4SGRID, é obter o esqueleto do código fonte da aplicação e os tipos de dados que serão utilizados. Desta forma, os métodos, parâmetros e campos das classes e intefaces são gerados. Porém, o comportamento dos métodos, contendo os blocos e instruções, não é gerado.

5.4.3 Definição de Transformação C - PSM Concreto para Código Fonte

A Definição de Transformação C (Figura 5.11) especifica a geração de código fonte para a plataforma final que, neste caso, é Java. Esta transformação de modelo-a-texto deve produzir o esqueleto do código das classes e interfaces que compõem a aplicação desenvolvida. Após a geração do código, o mesmo deve ser complementado manualmente para que a aplicação seja executável.

Figura 5.11 – Seção do FMDE4SGRID que corresponde à Definição de Transformação C.



Fonte: baseado em Junior (2017).

5.5 Prototipagem em IDE Eclipse

A implementação do FMDE4SGRID para auxiliar no desenvolvimento e integração de aplicações de *Smart Grids* para a plataforma DDS foi prototipada no IDE Eclipse, utilizando as ferramentas disponíveis no EMF (*Eclipse Modeling Framework*).

Um *workspace* do Eclipse foi organizado e preparado para conter os diferentes projetos do EMF que colocam em prática as diferentes necessidades do FMDE4SGRID. A seguir, as tecnologias e *plugins* utilizados dentro do ambiente Eclipse para criar uma prototipação do FMDE4SGRID são listadas:

- A **linguagem de metamodelagem Ecore** foi utilizada para produzir e editar os metamodelos propostos dentro do FMDE4SGRID, com exceção do metamodelo de XML *Schema*, que já é registrado no repositório de metamodelos do EMF por padrão;
- A ferramenta ***Genmodel* do EMF** foi utilizada para gerar os *plugins* necessários para permitir a edição dos modelos que são conforme os metamodelo criados;
- A ferramenta ***CIMTool***, também baseada em Eclipse, é utilizada para gerar os perfis de CIM em XML *Schema*;
- A **linguagem de transformação QVTo** (*Query-View-Transformation Operational*) foi utilizada para implementar cada transformação de modelo-a-modelo dentro do FMDE4SGRID;
- Por fim, o **projeto Acceleo** foi utilizado para realizar toda a geração de texto necessária dentro do FMDE4SGRID.

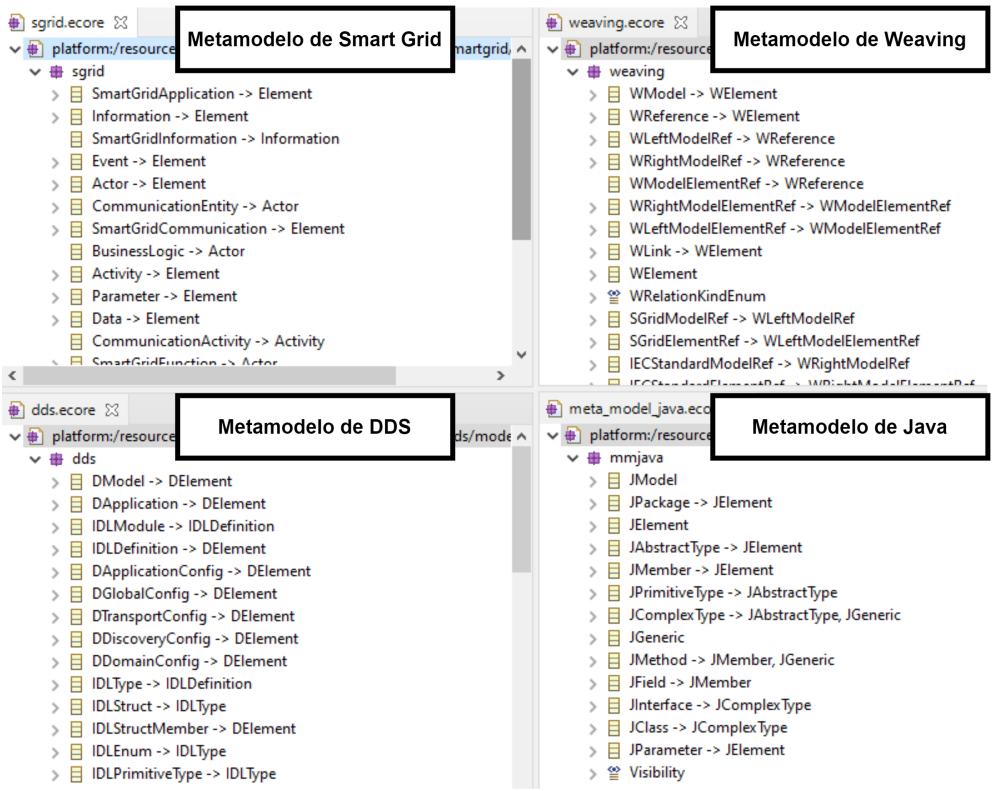
Cada um desses passos para a implementação do FMDE4SGRID no ambiente Eclipse é detalhado nas subseções a seguir. Na subseção 5.5.4, um passo a passo para a utilização da prototipação proposta é fornecido.

5.5.1 Implementação dos metamodelos e modelos

O editor de Ecore disponível no EMF foi utilizado para produzir os metamodelos propostos no FMDE4SGRID. A Figura 5.12 mostra os quatro metamodelos (*Smart Grids*, DDS, Java e *Weaving*) produzidos em linguagem Ecore. O metamodelo de Java foi reaproveitado do trabalho de Junior (2017).

O EMF possui uma ferramenta chamada *GenModel*, que permite gerar *plugins* em Java para criar e editar as instâncias dos metamodelos construídos em Ecore. Estes *plugins* podem ser integrados à instalação do Eclipse ou podem ser inicializados em uma segunda instância do Eclipse.

Figura 5.12 – Metamodelos do FMDE4SGRID produzidos em ambiente Eclipse utilizando a linguagem Ecore.



Fonte: acervo do autor.

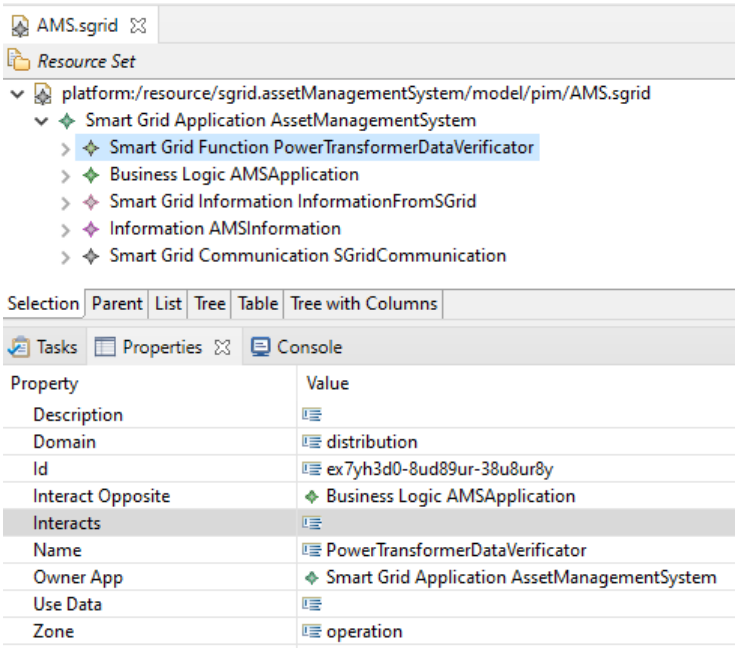
Na implementação realizada neste trabalho, os metamodelos e os *plugins* gerados pelo *GenModel* são criados em um dado *workspace* do Eclipse e, a partir deste *workspace*, uma nova instância do Eclipse é lançada. Para facilitar o entendimento, daqui em diante o *workspace* onde os metamodelos do FMDE4SGRID são criados e os *plugins* são gerados será chamado de *FMDE4SGRID-workspace*. A nova instância do Eclipse que é lançada a partir do *FMDE4SGRID-workspace* será chamada de *FMDE4SGRID-runtime*.

O *FMDE4SGRID-runtime* contém, dentro do repositório interno do EMF, os metamodelos criados. Desta forma, é possível criar e editar as instâncias dos metamodelos propostos, isto é, o PIM, os PDMs, o PSM Abstrato e PSM Concreto, utilizando o editor padrão do EMF, ou um editor de texto. A Figura 5.13 mostra um exemplo de edição de um modelo que é conforme o metamodelo de *Smart Grids* utilizando o editor padrão do EMF. Este editor possui as funcionalidades para a criação e exclusão de elementos e a edição dos atributos dos elementos dos modelos, como demonstra a Figura 5.13.

Dentro do ambiente do EMF, os metamodelos e modelos criados ficam disponíveis no repositório do Eclipse para serem acessados pelas ferramentas de transformação de modelos e geração de código.

Os PDMs, em particular, são criados com o auxílio do *CIMTool*, que é uma

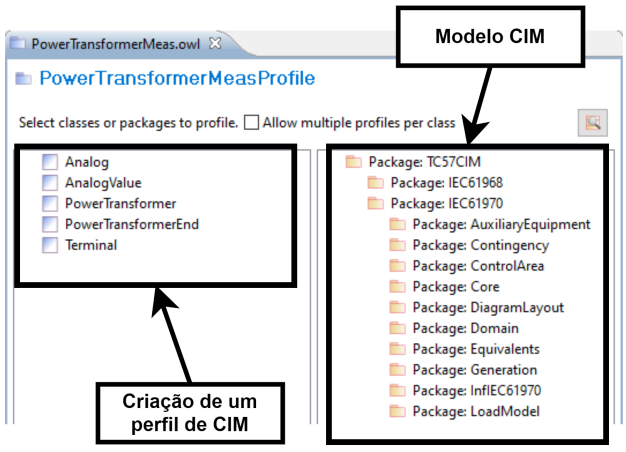
Figura 5.13 – Edição de um modelo conforme o metamodelo de *Smart Grids* utilizando o editor do EMF.



Fonte: acervo do autor.

implementação do Eclipse criada para gerar perfis do modelo CIM. O *CIMTool* possui um ambiente, como mostra a Figura 5.14, onde é possível navegar pelo modelo CIM e selecionar os elementos do modelo que irão compor um perfil de CIM. Este perfil pode ser exportado em vários formatos. Para a utilização dentro do *FMDE4SGRID-runtime*, os perfis de CIM criados no *CIMTool* são exportados em XML *Schema*.

Figura 5.14 – Criação de um perfil de CIM dentro da ferramenta *CIMTool*.



Fonte: acervo do autor.

5.5.2 Implementação das transformações de modelo-a-modelo

As duas transformações entre modelos do FMDE4SGRID foram implementadas utilizando a tecnologia QVTo (OMG, 2016). Também é possível implementar as transformações utilizando a linguagem ATL (JOUAULT et al., 2008), ou outra ferramenta dentro do EMF. A escolha pelo QVTo para este caso específico se deu por conta de que o QVTo é uma linguagem de transformação com uma característica mais imperativa em relação ao ATL, por exemplo, que é uma linguagem híbrida. Como muitos elementos precisavam ser criados dentro das transformações, uma abordagem mais flexível no que diz respeito à criação de elementos alvo sem correspondência com elementos fonte se fez necessária.

As transformações entre modelos correspondentes à *Definição de Transformação A* e à *Definição de Transformação B* foram implementadas no *FMDE4SGRID-runtime* utilizando a linguagem QVTo.

Um trecho da Definição de Transformação A, escrita em QVTo, é listada a seguir, na Listagem 5.1. Os mapeamentos descritos na Listagem 5.1 atuam como funções que produzem os elementos alvo a partir dos elementos fonte de entrada. Por exemplo, o mapeamento *actor2DClass()* produz como saída elementos *DDS::DClass* a partir dos elementos de entrada do tipo *SGRID::Actor*. No corpo dos mapeamentos, como é observado, por exemplo, nas linhas de 3 a 5, os atributos do elemento alvo podem ser inicializados com os valores correspondentes dos atributos do elemento fonte, identificado como *self*, desde que os atributos envolvidos tenham o mesmo tipo.

```
1  -- Mapeamento entre Actor e DClass
2  mapping SGRID::Actor::actor2DClass():DDS::DClass {
3    name := self.name;
4    id := self.id;
5    description := self.description;
6
7    -- Gerando os metodos a partir das activities
8    feature += self.activity->map activity2DMethod();
9  }
10
11 -- Mapeamento SmartGridFunction <-> DClass
12 mapping SGRID::SmartGridFunction::sgridFunction2DClass()
13   :DDS::DClass {
14
15   name := self.name;
16   id := self.id;
17   description := self.description;
18
```

```
19    -- Gerando os metodos a partir das activities
20    feature += self.activity->map activity2DMethod();
21  }
22
23  -- Mapeamento BusinessLogic <-> DCLass
24  mapping SGRID::BusinessLogic::businessLogic2DClass():DDS::DClass{
25    name := self.name;
26    id := self.id;
27    description := self.description;
28
29    -- Gerando os metodos a partir das activities
30    feature += self.activity->map activity2DMethod();
31  }
```

Listing 5.1 – Trecho da Definição de Transformação A, em QVTo

Quando os tipos dos atributos dos elementos fonte e alvo se equivalem, o valor do atributo do modelo fonte pode ser atribuído ao atributo correspondente do modelo alvo, como acontece na linha 15 da Listagem 5.1. Porém, nem sempre os tipos são equivalentes. Na linha 8 da Listagem 5.1, por exemplo, o atributo *feature* de *DDS::DClass* corresponde ao atributo *activity* de *SGRID::Actor*. No entanto, eles não são do mesmo tipo. Um outro mapeamento - neste caso, o mapeamento *activity2DMethod()* - precisa ser chamado de forma imperativa a partir deste ponto para gerar um elemento alvo compatível com *feature*. A mesma situação pode ser observada nas linhas 20 e 30. Isso significa que a ordem em que a transformação acontece é determinada na programação, o que, por um lado, pode gerar mais flexibilidade e, por outro lado, traz mais responsabilidade a quem está desenvolvendo.

Na Listagem 5.2, apresentada a seguir, um exemplo da utilização do *weaving* de modelos na transformação é dado. Esta listagem demonstra, entre as linhas 11 e 26, a criação de um elemento do tipo *DSoftwarePackage* de DDS. Dentro de *DSoftwarePackage*, os tipos complexos (*ComplexData*) de SGRID são transformados em *DClass* de DDS.

```
1  -- Mapeamento entre SmartGridApplication e DApplication
2  mapping SGRID::SmartGridApplication::
      sgridApplication2DApplication()
3       :DDS::DApplication{
4    -- A variavel packName corresponde ao nome que
5    -- sera atribuido ao software package
6    -- Um DSoftwarePackage sera criado para cada DApplication
7    var packName:String = null;
8    packName := self.name.toLower();
9
10   -- UM DSoftwarePackage eh criado dentro de DApplication
```

```
11    var softPack:DDS::DSoftwarePackage = null;
12    package += object softPack:DDS::DSoftwarePackage{
13      -- Recebe o nome designado para o DSoftwarePackage
14      name := packName;
15
16      -- Considerando os tipos complexos (ComplexData)
17      _datatype += pim.objectsOfType(SGRID::ComplexData)
18        ->select(e | e.ownerInfo.oclIsTypeOf(SGRID::Information))
19        ->map complexData2DClass();
20
21      -- Considerando agora apenas aqueles contidos por
           SMARTGRIDINFORMATION
22      _datatype += pim.objectsOfType(SGRID::ComplexData)
23        ->select(e | e.ownerInfo.oclIsTypeOf(SGRID::
             SmartGridInformation))
24        ->map complexData2DClassConsideringWeaving();
25    };
26  }
27
28  -- Mapeamento ComplexData <-> DClass
29  mapping SGRID::ComplexData::complexData2DClass():DDS::DClass{
30    name := self.name;
31    id := self.id;
32    description := self.description;
33  }
34
35  -- Mapeamento ComplexData <-> DClass
36  -- Este mapeamento procura o nome da Classe no link
37  mapping SGRID::ComplexData::
38    complexData2DClassConsideringWeaving()
39    :DDS::DClass{
40
41    if(weaving.objectsOfType(WEAVING::StandardRepresentationLink)
42      ->select(e | e.repOf.reference.endsWith(self.id))
43      ->notEmpty()){
44      name := weaving.objectsOfType(WEAVING::
           StandardRepresentationLink)
45        ->selectOne(e | e.repOf.reference.endsWith(self.id))
46        .standardElem.name;
47    }else{
48      -- Se o id do elemento nao for encontrado no weaving
49      name := self.name;
```

```
50    };
51
52    -- Atribuicao de id e description
53    id := self.id;
54    description := self.description;
55  }
```

Listing 5.2 – Trecho da Definição de Transformação A, em QVTo, na qual o modelo acessado para determinar o nome dos tipos gerados.

Observa-se que, entre as linhas 16 e 24 da listagem 5.2, dois mapeamentos diferentes são chamados para transformar elementos do tipo *SGRID::ComplexData* em elementos do tipo *DDS::DClass*. Na linha 18, apenas os elementos contidos por *SGRID::Information* são selecionados para participar da transformação realizada pelo mapeamento *complexData2DClass()*. Este mapeamento é demonstrado entre as linhas 29 e 33 da Listagem 5.2.

Na linha 23 da Listagem 5.2, apenas os elementos do tipo *ComplexData* contidos por *SGRID::SmartGridInformation* são selecionados para participar da transformação realizada pelo mapeamento *complexData2DClassConsideringWeaving()*, demonstrado entre as linhas 37 e 55. Neste mapeamento, o nome que deverá ser atribuído ao tipo *DClass* gerado é definido em um bloco *if/else*. O *id* do elemento fonte da transformação é procurado entre as referências dos *links* do modelo de *weaving*, como se observa nas linhas 41, 42 e 43 da Listagem 5.2. Caso esta referência seja encontrada, significa que o elemento (um *ComplexType* do PIM) em questão possui uma correspondência em um PDM. Assim, o nome do elemento alvo (um *DClass* do PSM Abstrato) é selecionado a partir do nome da referência ao PDM encontrada. Caso esta referência não seja econtrada, o nome atribuído ao *DClass* será simplesmente o nome do *ComplexType*, ou seja, *self.name*.

Outro desafio relevante dentro das transformações de modelos foi o de criar os elementos da API do *TopicHandler*. Em QVTo, utiliza-se a expressão *object* para criar elementos de modelos sem, necessariamente, haver um elemento correspondente. A Listagem 5.3 a seguir demonstra a criação do elemento *DTopicHandler* de DDS na Definição de Transformação A.

```
1 topicHandler := object tHandler:DDS::DTopicHandler{
2     name := topicTypeName + "TopicHandler";
3     usedBy += self.interactOpposite.resolve(DDS::DClassifier);
4 };
```

Listing 5.3 – Trecho da Definição de Transformação A, em QVTo, na qual o elemento *DDS::DTopicHandler* é criado.

A criação do *TopicHandler* acontece dentro do mapeamento entre

SGRID::CommunicationEntity e *DDS::DTopicHandlerPackage*. O atributo *usedBy*, na linha 3 da Listagem 5.3, deve conter as classes ou interfaces que fazem uso do *TopicHandler*. Essa relação é representada, no metamodelo de *Smart Grids*, pelo *binding* '*interactOpposite*' de *CommunicationEntity*. Assim, na linha 3, o elemento correpsondente a este *binding* é recuperado através da expressão *resolve()* do QVTo.

A Definição de Transformação B, que gera o PSM Concreto a partir do PSM Abstrato, é, em geral, mais simples que a primeira transformação. Lidar com a API de DDS talvez seja o maior desafio da Definição de Transformação B. Nesta implementação, a API de DDS foi carregada a *priori* na transformação. Isto é, existe um modelo externo, em IDL, que contém a API completa de DDS. Os elementos deste modelo são transformados em classes e interfaces conforme Java, em um pacote de Java separado. Esta é a primeira ação que ocorre na Definição de Transformação B implementada em QVT. Portanto, na prática, esta transformação tem dois modelos de entrada: o PSM Abstrato (conforme DDS) e a API de DDS (conforme IDL).

Durante o restante da transformação, os elementos de Java obtidos a partir dos elementos do *TopicHandler* de DDS fazem uma busca na API de DDS gerada sempre que um tipo específico da API é requisitado. A Listagem 5.4 demonstra esta situação.

```
1 mapping DDS::DTopicHandlerField::topicHandlerField2JField():JAVA
     ::JField{
2   name := self.name;
3
4   --Determinando o tipo a partir da API de DDS
5   fieldType := resolveIn(IDL::IDLInterface::
       idlInterface2JInterface)
6     ->select(e | e.name.equalsIgnoreCase(name))->first();
7 }
```

Listing 5.4 – Trecho da Definição de Transformação B, em QVTo, que mostra exemplifica como a API de DDS é levada em consideração.

O código da Listagem 5.4 demonstra o mapeamento entre o campo do *TopicHandler* (*DTopicHandlerField*) e o campo de tipo complexo de Java (*JField*). Na linha 5, uma expressão para obter um tipo correpsondente é implementada. A operação *resolveIn()* obtém todos os elementos gerados a partir do mapeamento *idlInterface2JInterface* - estes são elementos da API de DDS. O primeiro objeto que possui o mesmo nome do campo em questão é selecionado, de acordo com a aplicação do *select* na linha 6. Isso se deve ao fato de que todos os campos criados dentro do *TopicHandler* possuem o mesmo nome de seus respectivos tipos dentro da API de DDS. Por exemplo, os campos *publisher*, *subscriber* e *dataWriter* são dos tipos *Publisher*, *Subscriber* e *DataWriter*, respectivamente.

5.5.3 Implementação das transformações de modelo-a-texto

O projeto Acceleo[2], dentro do ambiente Eclipse, foi utilizado para gerar o código e os arquivos de configuração das aplicações a partir dos modelos. O Acceleo permite especificar como os modelos em EMF podem ser transformados em texto a partir da criação de *templates* do texto alvo.

Nesta implementação do FMDE4SGRID, os arquivos de configuração da plataforma de DDS e os arquivos IDL utilizados pelo *OpenDDS* para gerar a API de DDS são gerados diretamente do PSM Abstrato utilizando o Acceleo. O Acceleo também é utilizado para gerar o esqueleto código fonte das aplicações, em linguagem Java.

A Figura 5.15 mostra fragmentos da geração de texto implementada no FMDE4SGRID-runtime com auxílio do Acceleo.

Figura 5.15 – Um trecho dos *templates* para geração de texto no *FMDE4SGRID-runtime* utilizando o Acceleo.

Fonte: acervo do autor.

Na parte superior da Figura 5.15, parte do *template* utilizado para gerar os arquivos IDL é apresentado. Além dos arquivos IDL, alguns arquivos de configuração são gerados a partir do PSM Abstrato para auxiliar a configuração das aplicações para *OpenDDS*, como:

[2] *Acceleo* é uma ferramenta para geração de código implementada como *plug-in* em IDE-Eclipse. Toda a documentação do *Acceleo* está disponível no endereço <https://www.eclipse.org/acceleo/> (Último acesso em 13/06/2021).

arquivos do *framework* MPC para a geração de *Makefile* para compilação; arquivos de *shell script* utilizados para configurar o ambiente em Linux para a execução das aplicações; e arquivos de extensão '.ini' utilizados para a configuração da comunicação do *OpenDDS*.

Na parte inferior da Figura 5.15, parte do *template* que descreve a geração de código Java é demonstrado.

5.5.4 Passo a passo para a utilização da prototipagem do FMDE4SGRID

O processo para utilizar o FMDE4SGRID foi definido na seção 4.4 do Capítulo 4. No entanto, alguns passos específicos são necessários para pôr em prática o protótipo do FMDE4SGRID apresentado neste capítulo.

O passo a passo para utilizar o protótipo de FMDE4SGRID é listado a seguir. O nome e a numeração de cada passo é igual ao processo ilustrado na Figura 4.5 do Capítulo 4.

1. **Criar/Alterar/Reutilizar o PIM**. O PIM pode ser criado, dentro do *FMDE4SGRID-runtime*, através de um *wizard* do EMF. Se o PIM já estiver criado, ele pode ser alterado por edição de texto (edição do arquivo XMI correpondente ao PIM) ou através do editor do EMF;

2. **Criar/Alterar/Reutilizar os PDMs**. Os PDMs são gerados em uma ferramenta à parte do ambiente onde o FMDE4SGRID é desenvolvido. Esta ferramenta é chamada de *CIMTool*. Os perfis de CIM são gerados no *CIMTool* e são exportados em formado XSD. Dentro do *FMDE4SGRID-runtime*, os arquivos XSD correspondentes aos PDMs devem ser importados dentro do ambiente do Eclipse. Os PDMs podem ser modificados por edição de texto, embora isto não seja aconselhável, pois os PDMs são conforme a norma CIM e qualquer alteração pode descaracterizá-los;

3. **Criar o modelo de Weaving**. Uma pequena transformação em QVTo foi criada para carregar as referências dos PDMs e do PIM ao modelo de *Weaving*. Esta transformação deve, portanto, ser executada. Após isso, o usuário pode editar o modelo de *weaving* para adicionar os *links* entre o PIM e os PDMs manualmente;

4. **Gerar o PSM Abstrato**. A transformação em QVTo que gera o PSM Abstrato deve ser executada neste passo. O PSM Abstrato é gerado automaticamente a partir dos modelos de entrada;

5. **Complementar o PSM Abstrato**. Neste passo, caso necessário, o PSM Abstrato pode ser modificado utilizando o editor do EMF ou através de uma ferramenta de edição de texto;

6. **Gerar o PSM Concreto**. A transformção de modelos em QVTo que gera o PSM Concreto deve ser executada neste passo. O PSM Concreto é gerado automaticamente a partir do PSM Abstrato;

7. **Complementar o PSM Concreto**. O PSM Concreto pode ser modificado e complementado neste passo. Como nos outros casos, o PSM Concreto pode ser editado diretamente no texto do arquivo XMI ou utilizando a ferramenta de edição do EMF;

8. **Gerar o código fonte**. O gerador de texto do Acceleo é executado neste passo para gerar o código em linguagem Java a partir do PSM Concreto. Os arquivos com extensão *.java* serão criados dentro do ambiente do Eclipse, no diretório selecionado. Além disso, os arquivos de configuração da aplicação de DDS também podem ser gerados utilizando o Acceleo. Estes arquivos de configuração são gerados a partir do PSM Abstrato;

9. **Complementar o código fonte**. Neste passo, o código fonte da aplicação deve ser complementado para que a aplicação seja executável.

5.6 Síntese

Neste capítulo, uma implementação do FMDE4SGRID para auxiliar o desenvolvimento de aplicações e integração de aplicações em *Smart Grids* foi apresentada. Uma arquitetura de *software* baseada em *middleware* para compartilhamento de dados conforme o padrão CIM entre aplicações de *Smart Grids* foi escolhida como o estilo de projeto de *software* a ser seguido. O *middleware OpenDDS* foi escolhido como plataforma para implementar as aplicações desenvolvidas com o auxílio do FMDE4SGRID. A linguagem Java foi escolhida para a implementação em plataforma final das aplicações.

Uma camada de *software* que abstrai a API do *OpenDDS* chamada *TopicHandler* foi proposta. Esta camada de *software* fornece uma API simples para publicar e subscrever aos dados publicados em tópicos do *OpenDDS*.

O FMDE4SGRID foi reapresentado contendo os metamodelos específicos de plataforma escolhidos. O metamodelo de XML *Schema* foi escolhido para descrever os PDMs do FMDE4SGRID. O metamodelo de DDS foi apresentado, assim como o metamodelo de Java.

Na Seção 5.4, as definições de transformação entre modelos foram apresentadas. A Definição de Transformação A determina como os elementos do PSM Abstratos são gerados a partir dos modelos de entrada, que são o PIM, os PDMs e o *weaving*. A Definição de Transformação B especifica como os elementos do PSM Concreto são gerados a partir

do PSM Abstrato e, por fim, a Definição de Transformação C descreve a geração de código fonte a partir do PSM Concreto.

Uma prototipagem no ambiente Eclipse foi fornecida para implementar o FMDE4SGRID. A linguagem Ecore foi utilizada para implementar os metamodelos propostos. A ferramenta *GenModel* do EMF foi utilizada para gerar os *plugins* para Eclipse necessários para criar um ambiente de criação e edição de modelos conforme os metamodelos propostos.

A linguagem de tranformação QVTo foi utilizada para implementar as definições de transformação do FMDE4SGRID. A motivação para a escolha de QVTo é o fato de ela ser totalmente imperativa e, por isso, oferece boa flexibilidade para realizar operações dentro das transformações que vão além de mapeamentos simples.

O projeto *Acceleo* do Eclipse foi utilizado para implementar toda a geração de texto necessária dentro do FMDE4SGRID. Existem dois momentos no qual a geração de texto é utilizada na abordagem proposta: na geração dos arquivos de configuração para a plataforma *OpenDDS*, e na geração do código fonte das aplicações.

Por fim, um passo a passo para a utilização da prototipação do FMDE4SGRID foi fornecida na Subseção 5.5.4.

6 Exemplos Ilustrativos de Aplicações Desenvolvidas com o FMDE4SGRID

Este capítulo apresenta o desenvolvimento de exemplos ilustrativos que demonstram a utilização do FMDE4SGRID para auxiliar a atividade de desenvolvimento de *software* para *Smart Grids*. Ao todo, três aplicações de *Smart Grids* foram desenvolvidas com o auxílio do FMDE4SGRID. As aplicações desenvolvidas possuem apenas funcionalidades básicas e servem ao propósito de demonstrar como o FMDE4SGRID faz a separação entre o modelo de dados e o modelo da lógica de negócio desde as primeiras etapas do processo de desenvolvimento de *software*. No final do capítulo, os testes de comunicação entre as aplicações desenvolvidas são apresentados.

6.1 Aplicações desenvolvidas com o auxílio do FMDE4SGRID

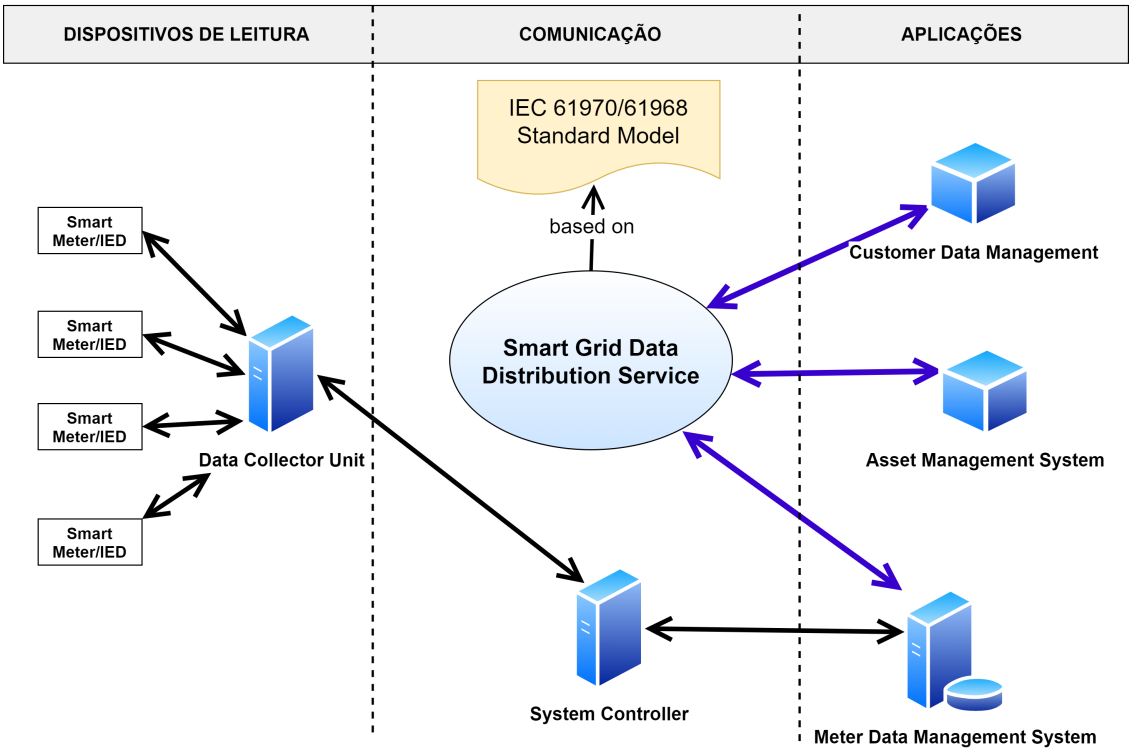
As aplicações de *Smart Grids* desenvolvidas com o auxílio do FMDE4SGRID são sistemas de gerenciamento da informação de sistemas elétricos. O modelo da rede elétrica, que é manipulado pelas aplicações de *Smart Grids*, é uma cópia virtual do sistema elétrico real, ou seja, tudo o que acontece no sistema elétrico real é monitorado pelas aplicações através de um modelo virtual. O modelo virtual é alimentado por dados do sistema elétrico real provenientes de sensores espalhados pela rede elétrica. A Figura 6.1 apresenta uma arquitetura de comunicação para *Smart Grids* baseada em uma arquitetura de AMI proposta por Gungor et al. (2012).

A arquitetura da Figura 6.1 serviu como base para o desenvolvimento das aplicações de *Smart Grids* com o auxílio do FMDE4SGRID. Na parte da esquerda da Figura 6.1, os dispositivos de aquisição de dados são ilustrados. O *Data Collector Unit* é um computador que recebe dados de um ou mais sensores (*Smart Meters* ou IEDs) em uma LAN e os envia ao *System Controller*, que é um sistema que está, possivelmente, em uma WAN e que recebe dados de vários *Data Collectors*. O *System Controller* envia os dados recebidos ao *Meter Data Management System* (MDMS), que faz o tratamento dos dados dos sensores recebidos, os coloca em um formato padrão, e os publica em um tópico de DDS, para que os dados estejam disponíveis para todas as outras aplicações de *Smart Grids*, de tal forma que todas as aplicações tenham acesso ao modelo virtual atualizado da rede elétrica.

As aplicações de *Smart Grids* desenvolvidas com o auxílio do FMDE4SGRID, dispostas à direita da Figura 6.1, são descritas a seguir:

- **Customer Data Management**. Esta aplicação armazena e gerencia uma base de

Figura 6.1 – Esquema de comunicação em *Smart Grids* utilizado para dar suporte à integração das aplicações desenvolvidas com o FMDE4SGRID.



Fonte: baseado em Gungor et al. (2012).

dados com o histórico de consumo dos usuários de energia elétrica. Estes dados podem ser utilizados posteriormente para criar *dashboards* com os dados de consumo e disponibilizá-los em uma página *web* para consulta ou podem ser utilizados para fazer o levantamento do perfil de consumo dos usuários utilizando técnicas como o *machine learning*;

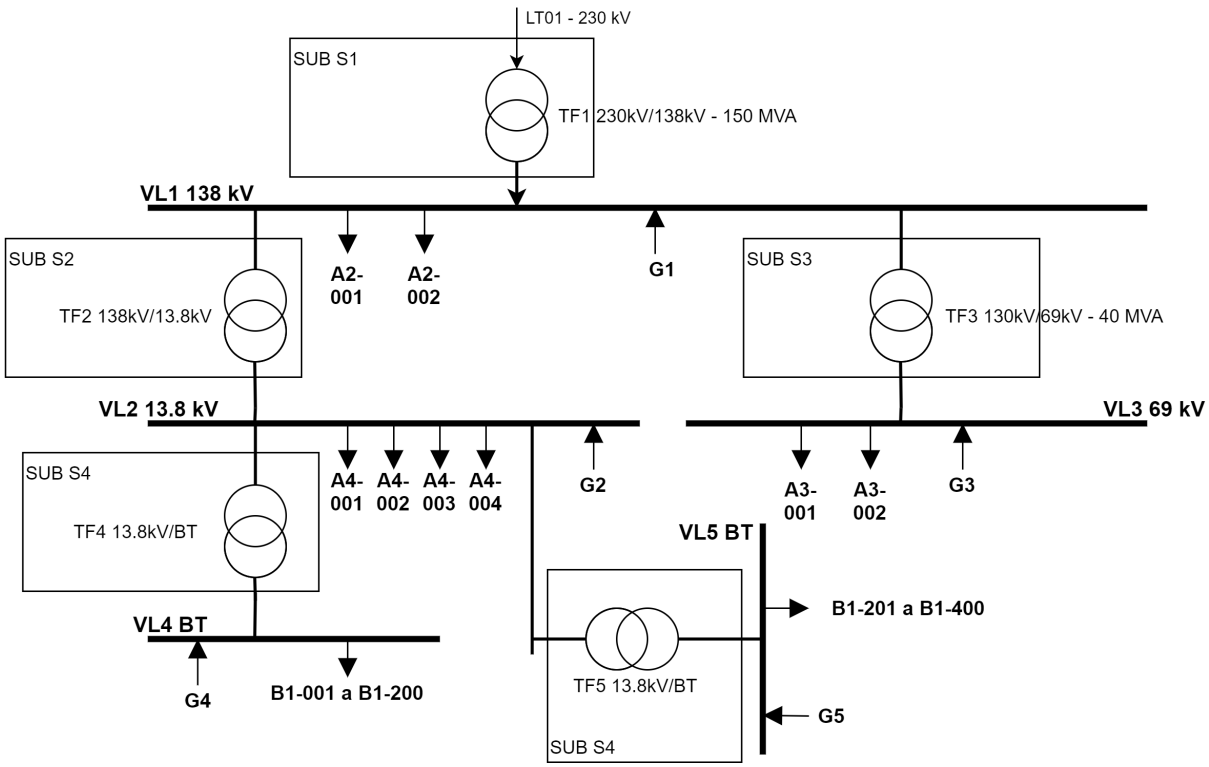
- **Asset Management System**. Esta aplicação monitora o funcionamento dos transformadores conectados à rede elétrica, verificando se os transformadores estão operando fora dos parâmetros normais definidos pelos fabricantes. O *Asset Management* monitora os parâmetros de corrente e tensão no primário e secundário, além do carregamento do transformador e a sua temperatura. As leituras de valores fora dos intervalos considerados normais são registradas para consulta dos gestores da rede elétrica;

- **Meter Data Management System**. O MDMS, como já descrito anteriormente, faz a atualização do tópico de DDS que contém os dados atualizados da rede elétrica monitorada para que as demais aplicações possam acessar os dados atualizados e desempenhar as suas funções.

6.2 Modelagem da rede elétrica utilizada como PDM no desenvolvimento das aplicações de *Smart Grids*

As aplicações de *Smart Grids* atuam sobre um modelo de dados que representa um sistema elétrico. No FMDE4SGRID, este modelo de dados corresponde ao PDM. O sistema elétrico utilizado como exemplo neste trabalho trata-se de uma rede de distribuição simplificada, ilustrada na Figura 6.2.

Figura 6.2 – Modelo simplificado de uma rede de distribuição de energia elétrica utilizado como exemplo para o desenvolvimento das aplicações de *Smart Grids*.



Fonte: baseado em Leão (2012).

O sistema de distribuição da Figura 6.2 contém cinco subestações (identificadas por retângulos) com os seus respectivos transformadores; a entrada do sistema é a linha de transmissão LT01 em 230 kV; existem diferentes níveis de tensão na rede, por exemplo o nível de 138 kV, representado pela simbologia VL1 (*Voltage Level* 1), e o nível de 69 kV, representado pela simbologia VL3 (*Voltage Level* 3); as cargas do sistema são identificadas na Figura 6.2 pelas setas que saem dos níveis de tensão da rede. Então temos, por exemplo, as cargas A2-001, A2-002, A4-001 e assim por diante. As designações A1, A2, A3 e B1 são as classificações de consumidores de energia elétrica segundo a ANEEL (ANEEL, 2010). Por fim, temos os geradores distribuídos, que podem ser painéis fotovoltaicos ou parques eólicos ou outro tipo de fonte distribuída. Os geradores distribuídos são designados na Figura 6.2 por G1, G2, G3, G4 e G5.

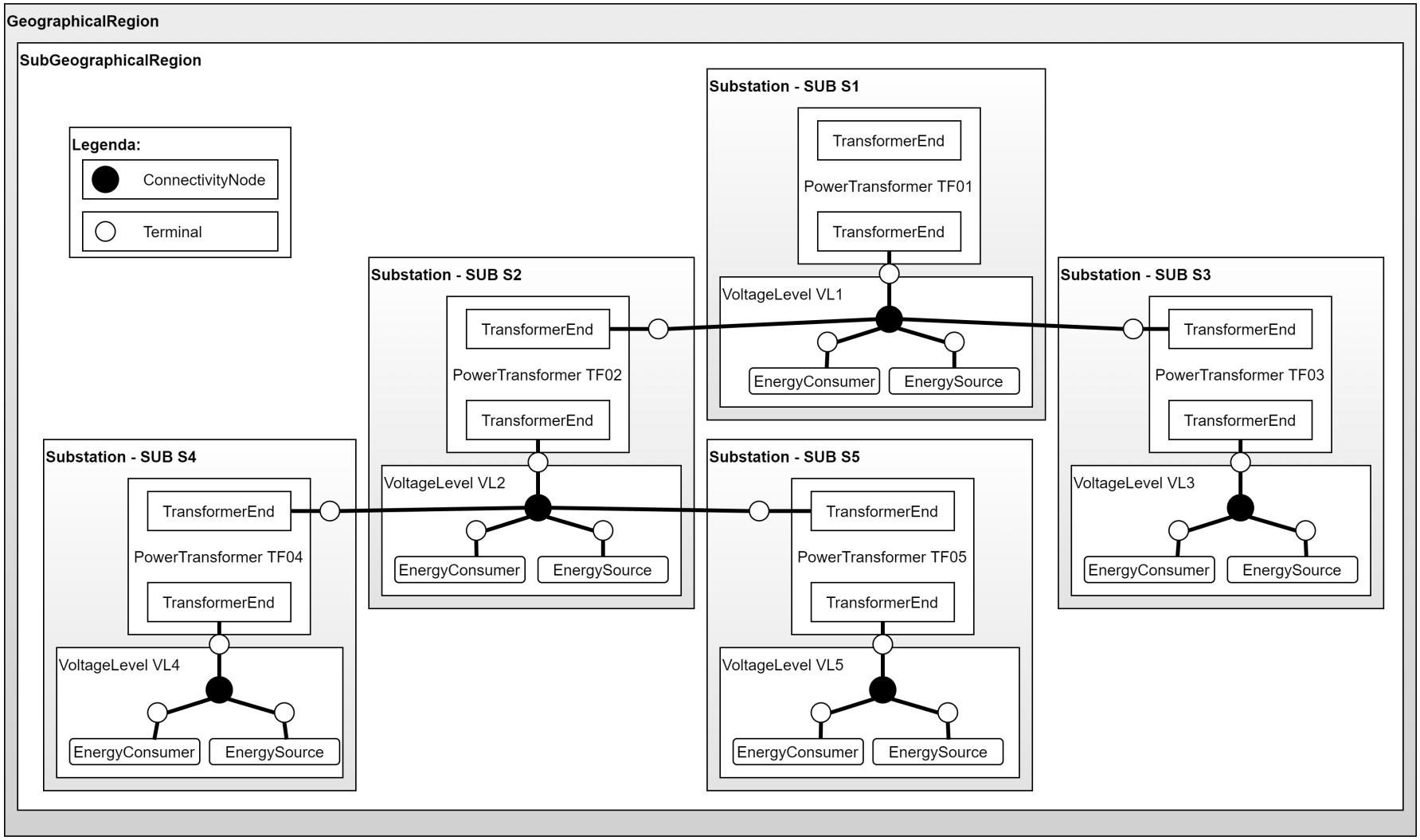
O modelo de sistema de distribuição da Figura 6.2 é um exemplo ilustrativo, não se tratando de um sistema real. Ele possui, portanto, várias simplificações que se verificam, principalmente, pela ausência dos equipamentos de proteção como disjuntores, relés, fusíveis, chaves, entre outros. O objetivo, com a escolha deste modelo de rede elétrica, é verificar a utilização da norma CIM para representá-lo e desenvolver aplicações que manipulem os dados desta rede conforme o CIM.

O perfil de CIM correspondente ao sistema de distribuição apresentado na Figura 6.2 é ilustrado na Figura 6.3. O elemento *GeographicalRegion* representa uma região do sistema de potência que contém uma sub-região, que é representada pelo elemento *SubGeographicalRegion*. As subestações são representadas pelos elementos do tipo *Substation* que, por sua vez contém os transformadores (*PowerTransformer*) e os *VoltageLevels*. Os elementos do tipo *VoltageLevel* são contêineres ou coleções de equipamentos e recursos da rede que operam sob um mesmo nível de tensão. Por exemplo, os elementos *EnergyConsumer* e *EnergySource* contidos por *VoltageLevel* VL2 estão sob o nível de tensão de 13,8 kV, que corresponde ao nível de tensão do lado de baixa do transformador TF02 pertencente à subestação S2.

As conexões elétricas entre os dispositivos condutores são representadas pelas relações entre *Terminal* e *ConnectivityNode*. Os *Terminals* são as representações dos terminais elétricos de cada equipamento ou instalação. Na Figura 6.3, os terminais são representados por um círculo branco. Os elementos do tipo *ConnectivityNode* representam a conexão elétrica entre dois ou mais terminais e são representados por um círculo preto. Pode-se observar que todas as conexões elétricas do modelo de sistema de distribuição da Figura 6.2 são representadas no modelo CIM da Figura 6.3 por meio de *Terminals* e *ConnectivityNodes*.

Alguns elementos do modelo CIM criado foram omitidos na Figura 6.3 para simplificar a ilustração. Destes, vale destacar o *BaseVoltage*, que define o nível de tensão base para um *VoltageLevel*, e o *Location*, que identifica a localização de equipamentos e instalações tanto através de pontos em um sistema de coordenadas quanto pela identificação do endereço da instalação.

Figura 6.3 – Modelo CIM da rede elétrica de distribuição utilizado nos exemplos de aplicações para *Smart Grids*.



Fonte: baseado em McMorran (2007).

6.3 Detalhamento do desenvolvimento de uma aplicação de *Smart Grids* com o FMDE4SGRID

Nesta seção, o desenvolvimento de uma aplicação de *Smart Grids* com o auxílio do FMDE4SGRID será detalhado. A aplicação escolhida para demonstrar a utilização do FMDE4SGRID é o *Asset Management System* – AMS ("Sistema de Gerenciamento de Recursos", em tradução livre).

Segundo Faheem et al. (2018), os AMS são sistemas que "*[...] darão suporte ao gerenciamento do risco de falha, manutenção proativa, performance do sistema, tempo de funcionamento, a saúde geral dos recursos, custos de manutenção e os impactos em confiabilidade.*" Faheem et al. (2018).

Os "*Assets*", ou seja, os recursos gerenciados pelo AMS podem ser praticamente qualquer recurso da rede elétrica incluindo turbinas eólicas, placas fotovoltaicas, veículos elétricos e híbridos, além dos recursos tradicionais como medidores, transformadores, chaves, disjuntores etc., e outros grupos de recursos como os sistemas de comunicação, *software*, *firmware* e dispositivos de armazenamento (FAHEEM et al., 2018).

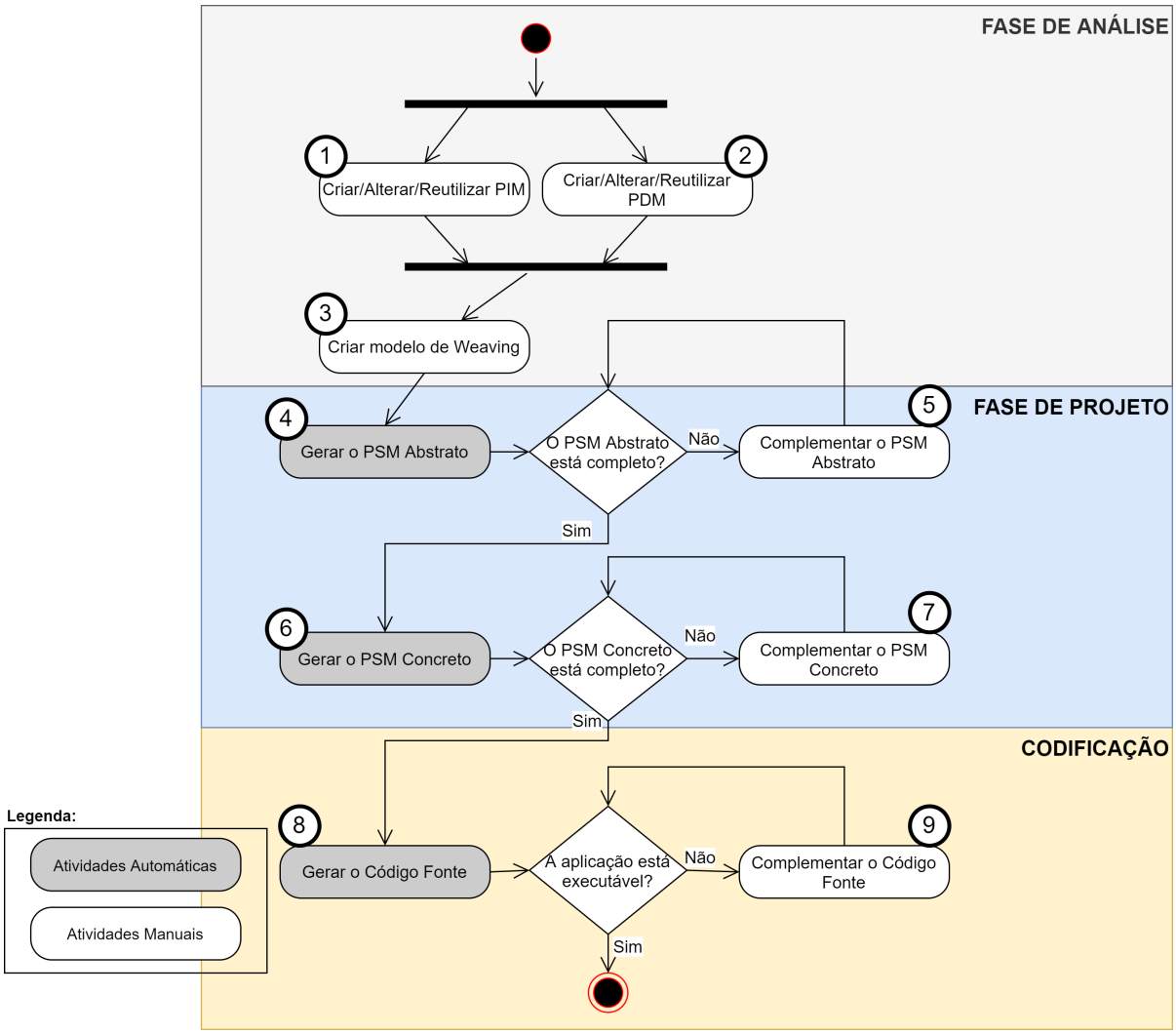
Para os fins do exemplo implementado neste trabalho, o AMS desenvolvido gerencia apenas transformadores. O AMS será responsável por monitorar os dados de sensores dos transformadores de uma rede elétrica e deve registrar os eventos de sobretensão, sobrecorrente, superaquecimento e sobrecarregamento de cada transformador.

O processo de desenvolvimento de *software* utilizado será um processo simples de desenvolvimento em cascata com as fases de Análise, Projeto de *Software*, Codificação e a realização de testes (SOMMERVILLE, 2011). O FMDE4SGRID será utilizado para gerenciar os artefatos de *software* produzidos em cada etapa. A metodologia proposta para o desenvolvimento das aplicações de *Smart Grids* com o auxílio do FMDE4SGRID é apresentado na Figura 6.4. Esta metodologia faz uma adaptação do processo apresentado na Figura 4.5 do Capítulo 4 para a utilização do FMDE4SGRID considerando o processo de desenvolvimento em cascata.

Conforme ilustrado na Figura 6.4, as etapas 1, 2 e 3 da utilização do FMDE4SGRD fazem parte da Fase de Análise, as etapas 4, 5, 6 e 7 fazem parte da Fase de Projeto e, por fim, as etapas 8 e 9 fazem parte da Fase de Codificação. A Fase de Testes não é contemplada na metodologia da Figura 6.4 pois o suporte aos testes de *software* foge do escopo da versão atual do FMDE4SGRID. No entanto, os testes funcionais das aplicações são realizados com o objetivo prático de demonstrar que as aplicações desenvolvidas realizam a comunicação via DDS e que elas realizam as suas funcionalidades básicas.

Nas próximas subseções o desenvolvimento do AMS será detalhado tendo como base as fases e etapas do desenvolvimento apresentadas na Figura 6.4.

Figura 6.4 – Metodologia utilizada para o desenvolvimento das aplicações de *Smart Grids* com o auxílio do FMDE4SGRID.



Fonte: baseado em Junior (2017) e Sommerville (2011).

6.3.1 Fase de Análise

Nesta fase, as funcionalidades da aplicação são definidas. O FMDE4SGRID auxilia na criação, manutenção e/ou reutilização do PIM, PDM e *Weaving*. O PIM contém a descrição das funcionalidades e a definição das informações que serão manipuladas pela aplicação; o PDM é o modelo da rede elétrica que é gerenciado pelas aplicações; o *Weaving* contém as relações entre PIM e PDM.

6.3.1.1 Desenvolvimento do PIM do *Asset Management System* (AMS)

O desenvolvimento do PIM é a primeira Etapa a ser seguida dentro da metodologia apresentada na Figura 6.4. O PIM contém as funcionalidades do *software* e representação dos dados que serão manipulados pela aplicação AMS. Os três principais requisitos funcionais do AMS podem ser colocados de forma bem resumida como segue.

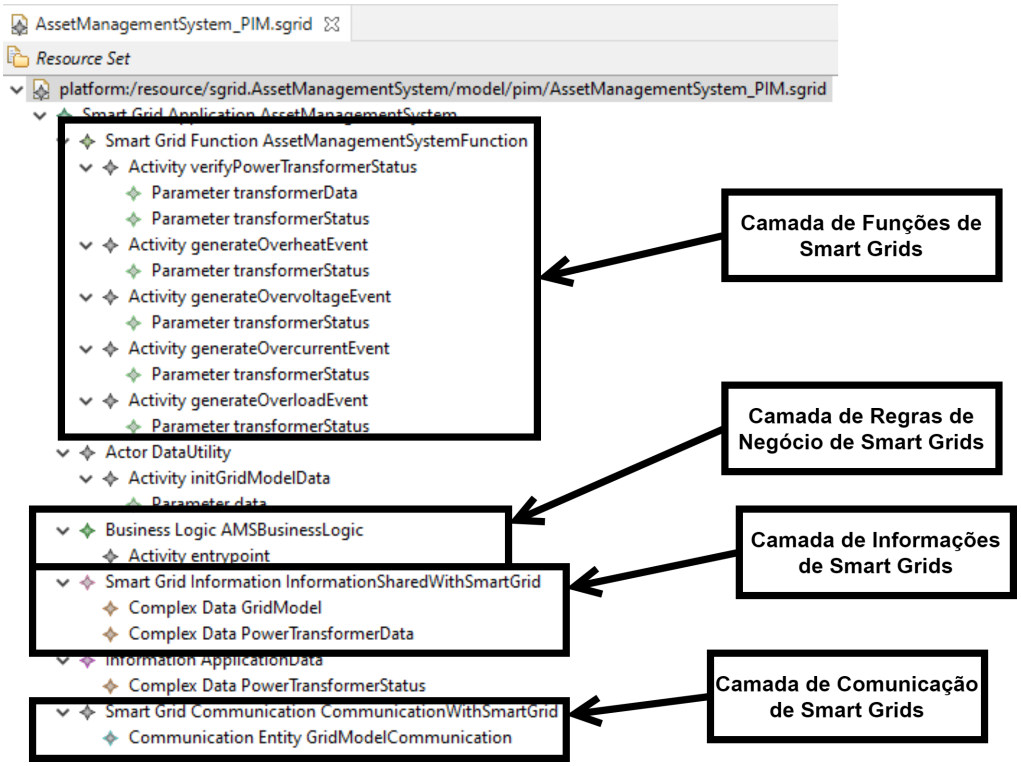
1. **Receber medições relacionadas à operação dos transformadores da rede elétrica**. Receber os dados publicados de tensão e corrente nos terminais, temperatura e carga dos transformadores da rede elétrica;

2. **Identificar o status de operação de cada transformador da rede**. O AMS deverá comparar os dados recebidos de leituras de sensores com valores de referência para a operação de cada transformador. O AMS deverá modificar o *status* de operação de cada transformador em uma tabela ou registro interno de acordo com o resultado da comparação dos dados recebidos com os valores de referência;

3. **Gerar e registrar um evento para cada leitura recebida fora das especificações**. Caso seja identificado que o *status* de operação de um transformador esteja fora das especificações do fabricante, o AMS deverá registrar em banco de dados ou em arquivo a ocorrência identificada.

A partir dos requisitos funcionais enumerados, o FMDE4SGRID é utilizado para produzir o PIM, que é conforme o metamodelo de *Smart Grids*. A Figura 6.5 mostra o PIM produzido com o auxílio do editor de modelos do EMF no ambiente Eclipse.

Figura 6.5 – O PIM desenvolvido para o *Asset Management System* (AMS).



Fonte: acervo do autor.

Na parte superior da Figura 6.5, as funcionalidades do AMS são definidos. Estas funcionalidades correspondem à camada de Funções de *Smart Grids* do *framework* SGAM.

A primeira funcionalidade definida é uma *activity* chamada *verifyPowerTransformerStatus* que verifica o *status* dos parâmetros de tensão, corrente, carregamento e temperatura de um transformador especificado no parâmetro *transformerData*. O *status* do transformador será gravado no parâmetro *transformerStatus*.

As outras funcionalidades definidas dentro das Funções de *Smart Grids* como a *activity generateOverheatEvent* fazem a leitura do *status* do transformador e registram um evento de, por exemplo, superaquecimento.

A Camada de Regras de Negócio de *Smart Grids* contém a lógica de execução da aplicação. Nesta versão do FMDE4SGRID, a modelagem do comportamento da aplicação ou o comportamento dos métodos ainda não é suportado, ou seja, o comportamento dos métodos da aplicação deve ser implementado manualmente no código fonte.

A Camada de Informações de *Smart Grids*, localizada logo abaixo da Camada de Regras de Negócio, contém a especificação dos dados utilizados pelo AMS e que são compartilhados com outras aplicações de *Smart Grids*. No caso do AMS, dois tipos de dados complexos são definidos: *GridModel* e *PowerTransformerData*. *GridModel* contém o modelo completo da rede elétrica monitorada, enquanto que *PowerTransformerData* contém os dados de um transformador. Estes dois tipos devem ser especificados no PDM da aplicação, conforme o modelo CIM para sistemas de energia elétrica. O modelo de *weaving* será utilizado para fazer a relação entre os tipos de dados definidos no PIM e os tipos de dados especificados no PDM.

A Camada de Comunicação, na parte inferior da Figura 6.5, contém a representação do componente de *software* responsável por fazer a comunicação com as outras aplicações de *Smart Grids*. A forma como a comunicação é realizada e os protocolos utilizados, porém, são especificados apenas nos modelos específicos de plataforma (PSM Abstrato e PSM Concreto).

6.3.1.2 Desenvolvimento do PDM para o *Asset Management System* (AMS)

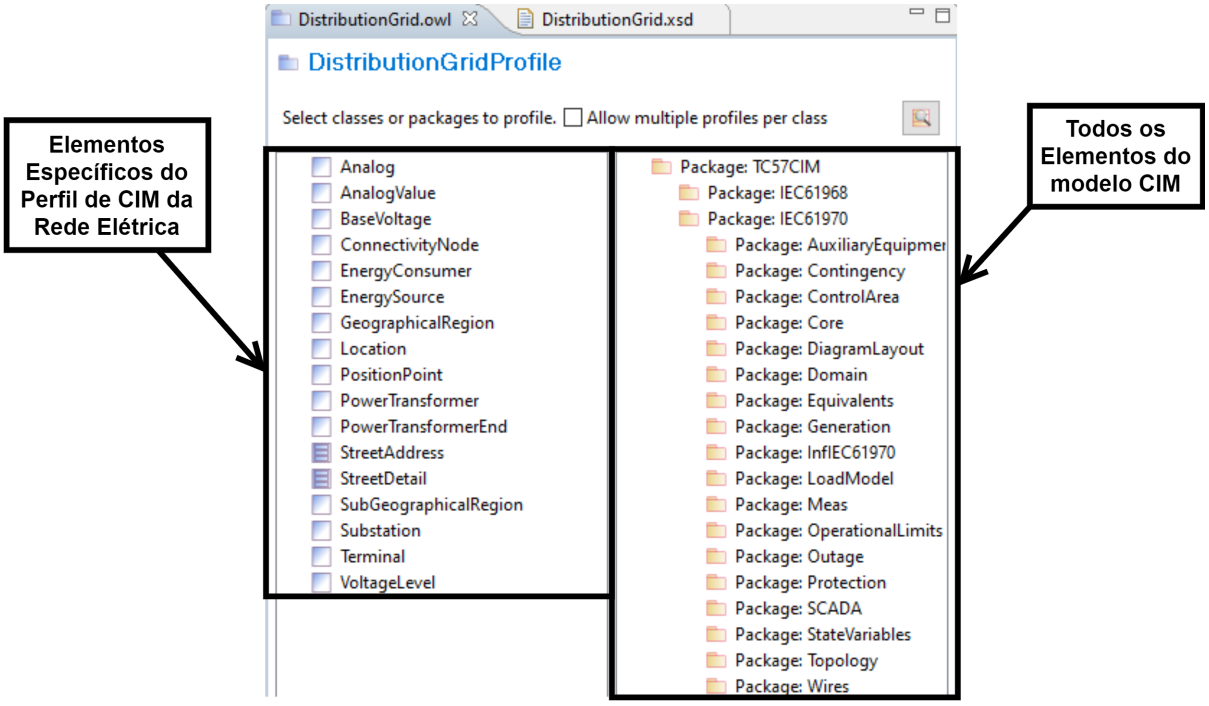
O CIM utilizado pelas aplicações de *Smart Grids* desenvolvidas dentro do escopo deste trabalho foi detalhado na Seção 6.2.

Para que o CIM possa ser utilizado no processo de desenvolvimento de *software* com o auxílio do FMDE4SGRID, ele precisa estar em um formato de serialização padrão, como o *XML Schema*.

A ferramenta *CIMTool*[1] foi utilizada para gerar o *XML Schema* do modelo CIM pretendido. O *CIMTool* oferece um ambiente baseado em Eclipse que permite a criação de um perfil do modelo CIM. A Figura 6.6 mostra o processo de criação de um perfil do CIM para ser utilizado como PDM nas aplicações desenvolvidas.

[1] *Website* da ferramenta *CIMTool*: <https://wiki.cimtool.org/>. Último acesso em 18/05/2021.

Figura 6.6 – Desenvolvimento de um perfil de CIM com o auxílio da ferramenta *CIMTool*.



Fonte: acervo do autor.

Á direita da Figura 6.6, todos os elementos do CIM são disponibilizados ao usuário. A partir disso, o usuário pode selecionar os elementos desejados e movê-los para a esquerda, onde consta o perfil de CIM que está sendo criado. O perfil de CIM é, portanto, um subconjunto do CIM selecionado para atender a uma demanda específica de modelagem. A demanda, no caso presente, consiste em modelar uma topologia de rede elétrica de distribuição simples contendo os elementos essenciais como *Substation*, *Terminal*, *PowerTransformer*, *ConnectivityNode*, *VoltageLevel* etc.

O *CIMTool* permite que o perfil de CIM criado seja exportado em vários formatos. O formato escolhido para que o modelo criado possa servir de entrada no FMDE4SGRID é o *XML Schema*.

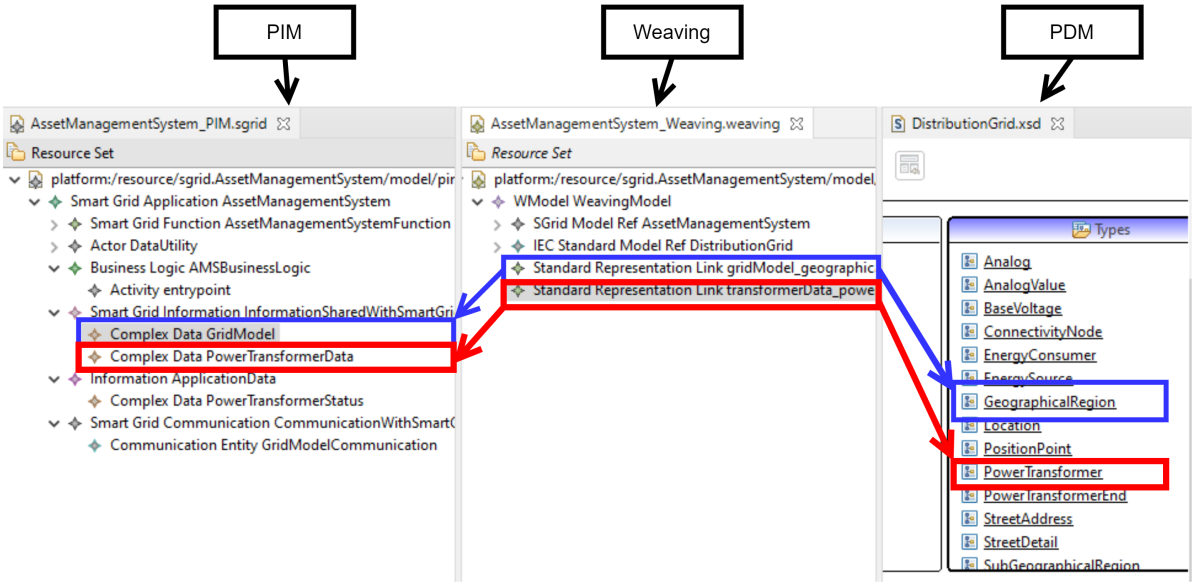
6.3.1.3 Desenvolvimento do modelo de *Weaving* para o AMS

O desenvolvimento do modelo de *Weaving* consiste na Etapa 3 da metodologia especificada na Figura 6.4. O primeiro passo para o desenvolvimento do *Weaving* é carregar os elementos do PIM e do PDM no *Weaving*. Isso foi feito com o auxílio de uma transformação em QVT que tem como entrada o PIM, que é conforme o metamodelo de *Smart Grids* (apresentado na Figura 4.3, no Capítulo 4); o PDM, que é conforme o metamodelo de *XML Schema* (apresentado na Figura 5.6, no Capítulo 4); e tem como saída o modelo de *Weaving* com os elementos do PIM e PDM carregados.

Depois de criado, o modelo de *Weaving* pode ser editado com a ferramenta padrão

de edição de modelos do EMF. A Figura 6.7 mostra o processo de edição do modelo de *Weaving* no ambiente Eclipse.

Figura 6.7 – Edição do modelo de *Weaving* na ferramenta de edição do EMF.



Fonte: acervo do autor.

Como está posto na Figura 6.7, os elementos do tipo *StandardRepresentationLink* (ao centro da Figura 6.7) fazem a relação entre os elementos do PIM e os elementos do PDM. Temos, portanto, que o tipo *GridModel* do PIM corresponde ao tipo *GeographicalRegion* do PDM. Isto significa que nos modelos específicos de plataforma e no código fonte da aplicação o tipo *GridModel* será substituído automaticamente por *GeographicalRegion*, que é baseado no CIM. A mesma relação pode ser observada (destaque em vermelho na Figura 6.7) entre *PowerTransformerData* do PIM e *PowerTransformer* do PDM.

6.3.2 Fase de Projeto

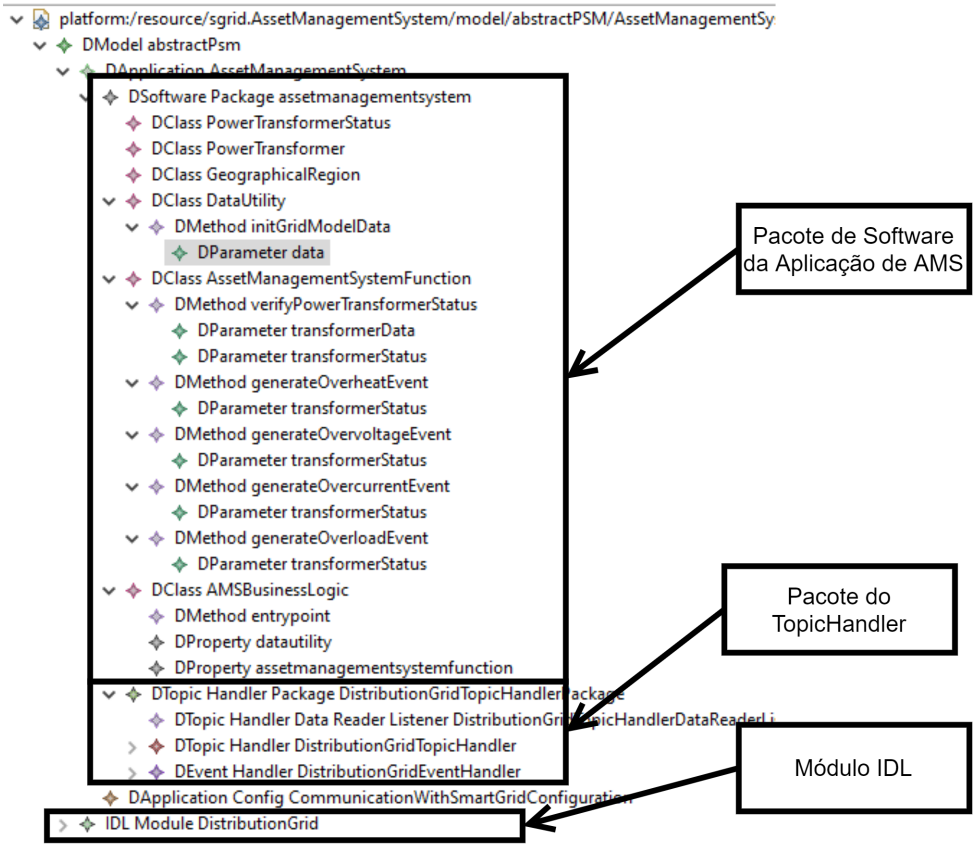
Nesta fase, a aplicação de AMS é projetada para uma plataforma específica. A plataforma escolhida para implementar todas as aplicações de *Smart Grids* desenvolvidas é o *middleware* DDS. O *OpenDDS* é a implementação de DDS escolhida para realizar a comunicação entre as aplicações de *Smart Grids*. Todas as aplicações foram implementadas em linguagem Java.

O FMDE4SGRID auxilia na geração automática dos dois principais artefatos de *software* na Fase de Projeto: o PSM Abstrato e o PSM Concreto. Seguindo a metodologia de desenvolvimento da Figura 6.4, as Etapas 4, 5, 6 e 7 são realizadas durante a Fase de Projeto.

6.3.2.1 Desenvolvimento do PSM Abstrato para o AMS

O desenvolvimento do PSM Abstrato consiste na realização das Etapas 4 ("Gerar o PSM Abstrato") e 5 ("Complementar o PSM Abstrato") da metodologia da Figura 6.4. O PSM Abstrato do AMS foi gerado automaticamente com o auxílio do FMDE4SGRID através da execução de uma definição de transformação que tem como entrada o PIM, o PDM e o *Weaving*. O PSM Abstrato do AMS é apresentado na Figura 6.8.

Figura 6.8 – PSM Abstrato do *Asset Management System* no ambiente de edição do EMF



Fonte: acervo do autor.

O PSM Abstrato do AMS pode ser dividido em três partes principais, como mostra a Figura 6.8. Na parte superior da Figura 6.8, o pacote de *software* da aplicação é definido, onde as classes, métodos, propriedades e parâmetros estão contidos; logo abaixo, temos o pacote do *TopicHandler*, que é uma abstração de *software* para a API de DDS; na parte inferior da Figura 6.8, o módulo IDL "*DistributionGrid*" é definido.

Na plataforma DDS, a linguagem IDL é utilizada para especificar os tipos de dados que são compartilhados nos tópicos do DDS. Os elementos contidos no módulo IDL "*DistributionGrid*" foram obtidos na transformação de modelos diretamente dos elementos do PDM, isto é, este módulo IDL é uma representação do modelo CIM em IDL. O módulo "*DistributionGrid*" é convertido em arquivo no formato IDL para servir de entrada para a

configuração da aplicação na plataforma *OpenDDS*, como será discutido na seção sobre a Fase de Codificação.

Todos os elementos do PSM Abstrato foram gerados corretamente com os seus respectivos tipos de *Property* e *Parameter* correspondendo aos tipos definidos nos modelos de entrada do FMDE4SGRID.

A Etapa 5 ("Complementar o PSM Abstrato") da metodologia da Figura 6.4 não se fez necessária no desenvolvimento do PSM Abstrato do AMS. Foi decidido que as modificações mais importantes na aplicação seriam feitas apenas no PSM Concreto, no intuito de fazer algumas adaptações à linguagem Java.

6.3.2.2 Desenvolvimento do PSM Concreto para o AMS

O desenvolvimento do PSM Concreto corresponde às Etapas 6 e 7 da metodologia da Figura 6.4. Na Etapa 6, com o auxílio do FMDE4SGRID, o PSM Concreto foi gerado automaticamente a partir da execução de uma definição de transformação em QVT cujo modelo de entrada é o PSM Abstrato.

O PSM Concreto, apresentado na Figura 6.9, é conforme o metamodelo de Java e, sendo assim, é um modelo de alto nível da implementação em Java do AMS. O modelo gerado contém todos os elementos de orientação à objetos contidos no pacote de *software* do PSM Abstrato. Além disso, os elementos do *TopicHandler* são transformados nas classes e interfaces correspondentes em Java, como está destacado na Figura 6.9.

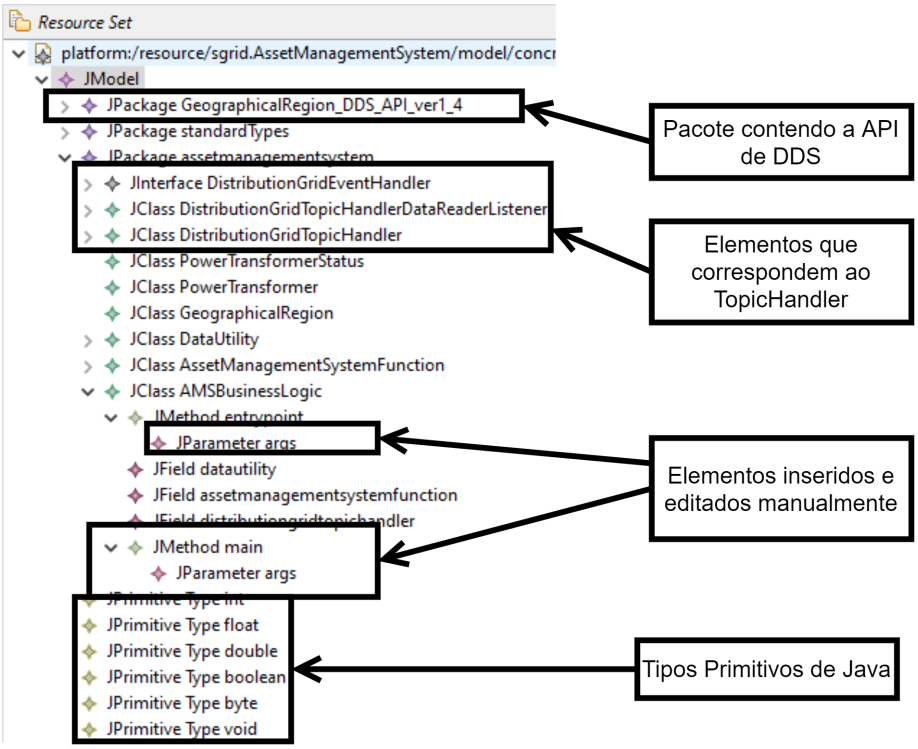
A API de DDS também é gerada na transformação, como está destacado na parte superior da Figura 6.9. Esta API é utilizada como referência para os tipos utilizados dentro do *TopicHandler* no PSM Concreto. No entanto, nenhum código fonte será gerado a partir deste pacote da API de DDS , pois uma API externa obtida da instalação do *OpenDDS* na forma de arquivo *.jar* será utilizada no momento da codificação da aplicação.

A Etapa 7 da metodologia de desenvolvimento utilizada consiste em editar o PSM Concreto caso haja necessidade. Como é indicado na Figura 6.9, alguns elementos foram inseridos no PSM Concreto, com destaque para o método *main* e o *array args* que contém os parâmetros de linha de comando. Na parte inferior da Figura 6.9, os tipos primitivos da linguagem Java estão em destaque. Estes tipos primitivos foram gerados automaticamente na transformação.

6.3.3 Fase de Codificação

Dentro da metodologia de desenvolvimento adotada (Figura 6.4), a Fase de Codificação engloba a Etapa 8, onde o código fonte e os arquivos de configuração da aplicação são gerados com o auxílio do FMDE4SGRID; e a Etapa 9, onde o código fonte é complementado manualmente.

Figura 6.9 – PSM Concreto do *Asset Management System* no ambiente de edição do EMF



Fonte: acervo do autor.

A codificação da aplicação de AMS pode ser dividida entre a codificação do programa em Java que consiste no código executável em si, e a criação de alguns *scripts* e arquivos de configuração que auxiliam a implantação do *software* na plataforma *OpenDDS*.

6.3.3.1 Desenvolvimento do código em Java do AMS

O esqueleto do código[2] em Java foi gerado automaticamente por meio da execução de uma transformação de modelo-a-texto com o projeto *Acceleo*. Esta transformação teve como entrada o PSM Concreto da aplicação.

O código gerado da classe *AssetManagementSystemFunction* é apresentado na Listagem 6.1. O código gerado contém os métodos definidos desde o PIM. Os parâmetros foram gerados com a especificação correta dos tipos, inclusive os tipos que foram definidos no PDM e foram selecionados através do *Weaving*, como o *PowerTransformer*.

```
1 public class AssetManagementSystemFunction {
2   public void verifyPowerTransformerStatus(PowerTransformer
      transformerData, PowerTransformerStatus transformerStatus){
3   }
```

[2] No contexto em que está sendo empregado, o termo "esqueleto de código" significa o código fonte que contém apenas a definição das classes e interfaces com os seus métodos, parâmetros e atributos, mas sem a especificação do comportamento dos métodos, isto é, sem as instruções imperativas do Java como *if*, *else*, *for*, *while*, as atribuições e chamadas de métodos.

```
4
5   public void generateOverheatEvent(PowerTransformerStatus
       transformerStatus){
6   }
7
8   public void generateOvervoltageEvent(PowerTransformerStatus
       transformerStatus){
9   }
10
11  public void generateOvercurrentEvent(PowerTransformerStatus
       transformerStatus){
12  }
13
14  public void generateOverloadEvent(PowerTransformerStatus
       transformerStatus){
15  }
16 }
```

Listing 6.1 – Código da classe *AssetManagementSystemFunction* gerado automaticamente.

Todas as classes do AMS foram geradas corretamente. O código gerado para a classe *TopicHandler* é listado a seguir (Listagem 6.2). Os tipos de alguns atributos como *Publisher*, *Subscriber*, *DataWriter* e *DomainParticipant* são obtidos da API de DDS.

```
1  public class DistributionGridTopicHandler {
2    private Topic topic;
3    private Publisher publisher;
4    private DataWriter dataWriter;
5    private GeographicalRegion instanceDataWriter;
6    private Subscriber subscriber;
7    private DistributionGridTopicHandlerDataReaderListener listener
        ;
8    private DataReader dataReader;
9    private GeographicalRegion instance;
10   private DomainParticipant domainParticipant;
11   private DomainParticipantFactory domainParticipantFactory;
12   private TypeSupport typeSupport;
13
14   public void configureTopicHandler(String topicName, int
        domainId, String[] args){
15   }
16
17   public void enablePublisher(){
```

```
18    }
19
20    public void enableSubscriber(DistributionGridEventHandler
          eventHandler){
21    }
22
23    public void registerInstance(GeographicalRegion instance){
24    }
25
26    public void publishInstance(GeographicalRegion instance){
27    }
28  }
```

Listing 6.2 – Código da classe *TopicHandler* gerado automaticamente.

A Etapa 9 da metodologia empregada para o desenvolvimento da aplicação AMS consiste em complementar manualmente o código fonte gerado. Isto foi feito em todas as classes e interfaces geradas para o AMS. É importante esclarecer que apenas o código da aplicação em si precisou ser complementado manualmente. A API de DDS pode ser gerada através da instalação do *OpenDDS*, com a entrada de alguns arquivos de configuração fornecidos pelo usuário. Estes arquivos de configuração foram gerados automaticamente com o auxílio do FMDE4SGRID, como é detalhado na próxima subseção.

6.3.3.2 Desenvolvimento dos arquivos de configuração para o AMS

A Etapa de geração de código no desenvolvimento da aplicação de AMS também envolveu a geração dos arquivos de configuração que auxiliam na implantação da aplicação no ambiente do *OpenDDS*. É importante notar que todos os arquivos de configuração foram gerados a partir do PSM Abstrato e não do PSM Concreto. A razão para isso é que o PSM Abstrato está mais próximo dos conceitos de DDS do que o PSM Concreto, que está mais próximo dos conceitos de Java.

O principal arquivo entre os arquivos gerados é o arquivo IDL, que contém os tipos de dados definidos no PDM e que são utilizados para representar uma rede elétrica. Na listagem a seguir (Listagem 6.3), um fragmento do código IDL gerado é apresentado.

```
1
2  @topic
3  struct GeographicalRegion{
4    string mRID;
5    string aliasName;
6    string name;
7    SubGeographicalRegionSeq regions;
8  };
```

```
9
10 struct SubGeographicalRegion{
11   string mRID;
12   string aliasName;
13   string name;
14   GeographicalRegionSeq region;
15   SubstationSeq substations;
16 };
17
18 struct Substation{
19   string mRID;
20   string aliasName;
21   string name;
22   PowerTransformerSeq equipments;
23   LocationSeq location;
24   SubGeographicalRegionSeq region;
25   VoltageLevelSeq voltagelevels;
26 };
```

Listing 6.3 – Fragmento do código IDL gerado com o auxílio do FMDE4SGRID

Na Listagem 6.3 vemos que *GeographicalRegion* contém uma sequência de *SubGeographicalRegion* que contém uma sequência de *Substation*, e assim por diante. A *tag* "*@topic*" acima da definição de *GeographicalRegion* indica ao compilador de IDL utilizado pelo *OpenDDS* que este tipo será um tópico de DDS. A partir disso, o compilador irá gerar o código da API de DDS que leva em consideração o tipo *GeographicalRegion* como um tópico.

Outros arquivos foram gerados para a configuração e implantação do AMS na plataforma *OpenDDS*, como os arquivos MPC (*Make Project Creator*[3]) utilizados para configurar a compilação do arquivo IDL e código fonte, e alguns *scripts* para automatizar a compilação e configuração da aplicação.

Os arquivos de configuração não precisam de edição manual para ficarem prontos para o uso, o que facilitou o processo de implantação das aplicações.

6.4 Cenário de testes das aplicações desenvolvidas

Esta seção apresenta um cenário de testes onde as aplicações desenvolvidas foram testadas com o objetivo de verificar a comunicação entre as aplicações de *Smart Grids* via *OpenDDS*.

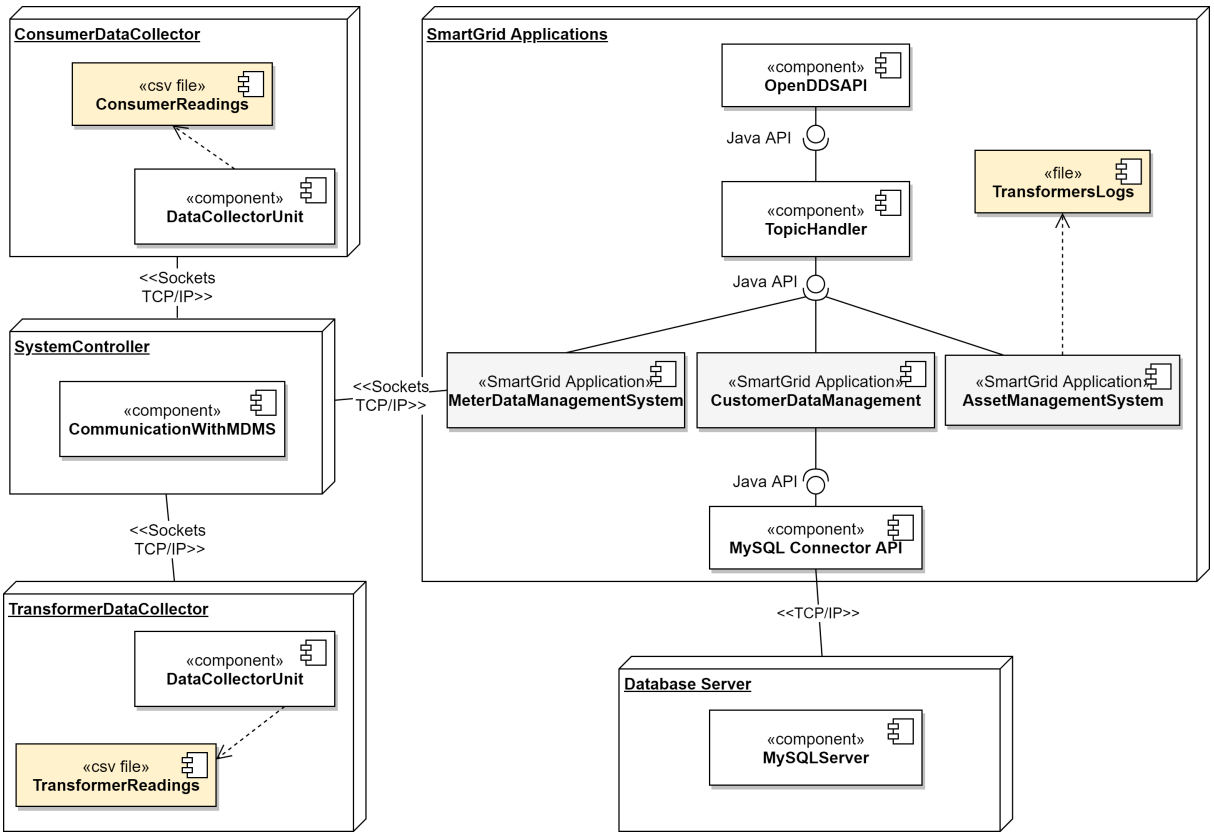
[3] Informações básicas sobre o MPC podem ser encontradas neste *website*: <http://downloads.ociweb.com/MPC/docs/html/MakeProjectCreator.html>. Último acesso em: 19/05/2021.

6.4.1 Sistema distribuído implementado

Um sistema distribuído básico é proposto para se transmitir dados de sensores de uma rede elétrica para as aplicações desenvolvidas para que estas aplicações possam processar e compartilhar entre si os dados fornecidos.

O diagrama de implantação do sistema distribuído proposto é apresentado na Figura 6.10. Neste sistema, os dados provenientes de consumidores de energia elétrica e os dados provenientes de transformadores de potência são gerados por nós chamados *DataCollectors*. Estes dados são enviados via *socket* TCP/IP para um servidor chamado *SystemController*, que encaminha os dados recebidos para o *Meter Data Management System* (MDMS) via *sockets* TCP/IP. O MDMS, por sua vez, possui os objetos em Java que representam a rede elétrica e atualiza estes objetos em um tópico do DDS, para que as outras aplicações de *Smart Grids* tenham acesso aos objetos atualizados.

Figura 6.10 – Diagrama de implantação proposto para o cenário de testes das aplicações de *Smart Grids*



Fonte: baseado em Gungor et al. (2012).

As aplicações *MeterDataManagementSystem*, *CustomerDataManagement* e *AssetManagementSystem* utilizam a API do *TopicHandler* para publicar e subscrever aos dados da rede elétrica via *OpenDDS*.

O *TopicHandler* utiliza a API em Java fornecida pelo *OpenDDS* para utilizar os

serviços de publicação e subscrição de dados no *middleware*. O *middleware OpenDDS* faz a sincronização do estado dos objetos compartilhados para cada aplicação.

O *AssetManagementSystem* (AMS) faz o registro em arquivos (*TransformersLogs*) das ocorrências de mal funcionamento dos transformadores, detectadas a partir de uma análise dos dados atualizados no *OpenDDS* sobre a rede elétrica.

O *CustomerDataManagement* (CDM) utiliza uma API de MySQL *Connector* para registrar em um banco de dados MySQL *Server* o consumo de energia elétrica de cada consumidor da rede elétrica monitorada. Os dados ficam registrados em um banco de dados relacional para que possam ser acessados posteriormente por outra aplicação, como um sistema de faturamento em tempo real dos clientes.

6.4.2 Fontes dos dados utilizadas

Os dados da rede elétrica que são gerados pelas aplicações de *DataCollector* são de duas fontes distintas. Os dados de consumo dos usuários foram obtidos de um *dataset* disponível *online* gratuitamente no portal *Open Energy Data Initiative*[4] (OEDI), vinculado ao Departamento de Energia dos Estados Unidos. O OEDI possui um rico repositório de dados relacionados à energia e ambiente.

O *dataset* do OEDI utilizado contém o perfil horário de consumo de energia elétrica de estabelecimentos comerciais como supermercados, escritórios, escolas e hospitais; além do consumo de energia elétrica em residências. Os dados estão disponíveis em formato CSV (*Comma-Separated Values*).

O *DataCollector* responsável por publicar os dados de consumo de clientes de energia elétrica fazem a leitura dos arquivos CSV e enviam os dados ao *SystemController* como é ilustrado na Figura 6.10.

A outra fonte de dados utilizada foram tabelas em formato CSV criadas manualmente contendo dados de leitura dos transformadores da rede elétrica, como tensão, corrente e temperatura. O objetivo ao se proceder desta forma para gerar estes arquivos é inserir propositalmente valores de leitura que são acima dos valores máximos especificados para os transformadores. Desta forma, a aplicação AMS identifica as anomalias e geram os *logs* das leituras realizadas. Uma tabela em formato CSV foi criada para cada transformador.

6.5 Execução dos testes das aplicações

As aplicações utilizadas para gerar os dados de leitura de consumo de energia elétrica dos usuários e de parâmetros dos transformadores, ou seja, o *ConsumerDataCollector*, o *TransformerDataCollector* e o *SystemController* foram implementados em um computador

[4] O portal OEDI está disponível no link: <https://data.openei.org/>. Último acesso em 23/05/2021.

com sistema operacional Windows 10 e foram monitoradas a partir do *console* do Eclipse-IDE. As aplicações de *Smart Grids* que compartilham dados via *OpenDDS* e fazem a análise destes dados, isto é, o MDMS, o AMS e o CDM foram implementadas em um computador com sistema operacional Ubuntu versão 18.04. A versão de OpenDDS utilizada foi a 3.14.1, embora a versão 3.17.0 já esteja disponível para *download* no site do projeto *OpenDDS*[5].

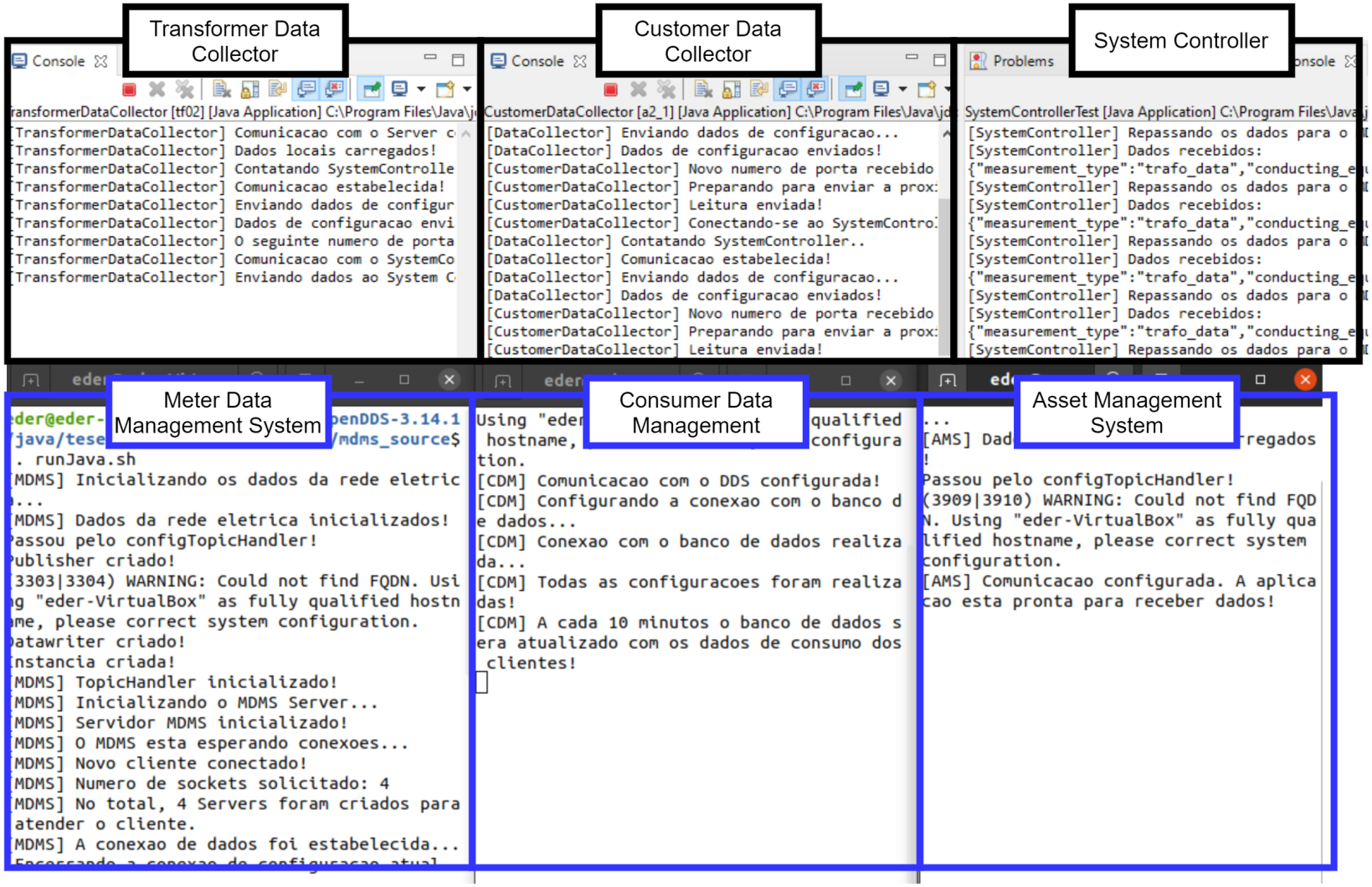
O banco de dados *MySQL* utilizado foi instalado na mesma máquina das aplicações de *DataCollector* e *SystemController*. A versão utilizada do *MySQL Server* foi a 8.0.16. A versão 8.0.25 (versão mais recente encontrada no momento em que este texto é redigido) apresentou alguns problemas com o administrador de banco de dados no *MySQL Workbench* e, por isso, foi decidido que se utilizaria uma versão anterior.

A Figura 6.11 mostra o ambiente de execução das aplicações. Na parte superior da Figura 6.11 temos as aplicações de coleta de dados em execução no console do Eclipse-IDE. Na parte inferior da Figura 6.11, temos a execução das aplicações de *Smart Grids* desenvolvidas com o auxílio do FMDE4SGRID. Estas aplicações compartilham os objetos Java que representam a rede elétrica monitorada via *middleware* DDS. É possível observar, na Figura 6.11, os logs gerados por cada aplicação. O "*Transformer Data Collector*" e o "*Customer Data Collector*" iniciam a comunicação com o servidor "*System Controller*", que abre um socket TCP para cada cliente. Os dados recebidos por "*System Controller*" (enviados por "*Transformer Data Collector*" e "*Customer Data Collector*") são repassados ao MDMS. Os logs do MDMS (*Meter Data Management System*) indicam que o cliente da comunicação (neste caso, o *System Controller*) solicita a quantidade de *sockets* TCP que o MDMS deve gerar para realizar a transferência de dados. Neste caso, a quantidade de *sockets* solicitada foi 4.

O MDMS publica os dados recebidos no DDS, em um tópico chamado "*GridDistributionData*", enquanto que o AMS e o CDM subscrevem a este tópico para receber os dados e fazer o processamento pertinente a cada aplicação.

[5] O *download* da versão mais recente de *OpenDDS* está disponível em: <https://opendds.org/downloads.html>. Último acesso em 23/05/2021.

Figura 6.11 – Aplicações desenvolvidas em execução.



Fonte: acervo do autor.

Os resultados obtidos da execução das aplicações desenvolvidas são apresentados na Figura 6.12. Na parte superior da Figura 6.12 com destaque em preto, a tabela de um banco de dados *MySQL* que é atualizada pela aplicação "*CustomerDataManagement*" é apresentada. Estes dados representam o consumo do usuário, que pode ser comercial ou residencial, na última hora de medição em kWh. O nome do estabelecimento, o endereço, o consumo e o horário em que a medição é feita são armazenados no banco de dados.

Figura 6.12 – Resultados obtidos pela execução das aplicações de *Smart Grids* desenvolvidas.

Fonte: acervo do autor.

Na parte central da Figura 6.12 (destaque em azul) os *logs* gerados pelo AMS são mostrados. Estes *logs* são gerados sempre que o AMS verifica os objetos da rede elétrica que são publicados pelo MDMS e constata que algum dos parâmetros de algum transformador está fora das especificações. Estes *logs* são gravados como arquivos em formato JSON.

Na parte inferior da Figura 6.12 (destaque em vermelho), um arquivo de Log é detalhado. É possível observar que o tipo de anomalia identificada é explicitado. No exemplo indicado na Figura 6.12, a anomalia é do tipo "*OVERLOAD*", ou seja, o transformador esteve em situação de sobrecarga no horário indicado pelo campo "*timestamp*". O registro indica ainda que o valor detectado é de 3.200.000 VA, isto é, 3.2 MVA; e que o valor máximo suportado pelo equipamento é de 3.000.000 VA, ou seja, 3.0 MVA.

Todas as aplicações desenvolvidas compriram o seu propósito, que é o de realizar a sua funcionalidade básica. O FMDE4SGRID auxiliou nas fases de Análise, Projeto e Codificação do desenvolvimento das aplicações para *Smart Grids*, onde as funcionalidades foram definidas, o modelo da rede elétrica (conforme o CIM) monitorada foi definido e os modelos específicos de plataforma foram gerados e editados com as ferramentas de edição disponibilizadas pelo EMF. Todos os arquivos de configuração para configurar a plataforma OpenDDS foram gerados automaticamente, assim como o esqueleto do código em linguagem Java.

6.6 Síntese

Este capítulo apresentou o desenvolvimento e a execução de exemplos de aplicações de *Smart Grids* com o auxílio do FMDE4SGRID. Três aplicações foram desenvolvidas seguindo uma metodologia de desenvolvimento de *software* baseada em cascata adaptada à metodologia de utilização do FMDE4SGRID.

É importante destacar que o FMDE4SGRID não é em si uma metodologia de desenvolvimento e sim um *framework* que propõe gerenciar os artefatos de *software* através da criação, manutenção e transformação de modelos.

O modelo da rede elétrica utilizada como exemplo para a análise de dados das aplicações foi apresentada. Um modelo de rede de uma distribuição simples foi adotada e o CIM equivalente desta rede foi apresentado. O CIM é utilizado dentro do FMDE4SGRID como o PDM.

O desenvolvimento da aplicação *Asset Management System* (AMS) com o auxílio do FMDE4SGRID foi detalhado. O desenvolvimento foi dividido em fases de Análise, Projeto e Codificação.

Na Fase de Análise, o PIM da aplicação foi desenvolvido em conformidade com o metamodelo de *Smart Grids*; o PDM foi desenvolvido com o auxílio da ferramenta *CIMTool* para a criação de perfis de CIM. Um perfil de CIM foi criado de acordo com o modelo da rede de distribuição proposto, e o perfil de criado foi exportado em XML *Schema*; o modelo de *Weaving* foi criado. O *Weaving* contém as relações ou *links* entre os *datatypes* do PIM que representam elementos da rede elétrica e os *datatypes* do PDM que representam os mesmos elementos, mas que estão de acordo com o CIM.

Na Fase de Projeto, os modelos específicos de plataforma foram criados. O PSM Abstrato é gerado automaticamente através da execução de uma definição de transformação que tem como entrada o PIM, o PDM e o *Weaving*; o PSM Concreto, que contém conceitos específicos da linguagem Java, foi gerado automaticamente a partir do PSM Abstrato. O PSM Concreto foi editado e complementado manualmente para inserir mais alguns tipos e

métodos necessários para complementar melhor a aplicação.

Por fim, na Fase de Codificação, o esqueleto do código fonte em Java e os arquivos de configuração da plataforma *OpenDDS* foram gerados automaticamente. O código fonte precisou ser complementado manualmente, o que exigiu um trabalho adicional. Por outro lado, os arquivos de configuração de plataforma foram gerados completamente para cada aplicação, o que facilitou a configuração das aplicações para rodar com o *OpenDDS*.

As aplicações desenvolvidas foram executadas e testadas para aferir se as mesmas realizam a comunicação de maneira eficiente. Um sistema distribuído foi implementado, onde algumas aplicações foram implantadas para simular a leitura de sensores da rede elétrica. Estes dados são enviados às aplicações de *Smart Grids* desenvolvidas via *socket* TCP/IP e estas aplicações compartilham entre si os dados via *OpenDDS* para realizar a análise e gerar informação útil.

7 Análise dos Resultados Obtidos e Comparações com os Trabalhos Relacionados

Neste capítulo, uma análise dos resultados obtidos neste trabalho é fornecida. Os critérios utilizados para avaliar os trabalhos do Estado da Arte apresentados na Seção 3.3 do Capítulo 3 são utilizados para avaliar a abordagem apresentada neste manuscrito. Em seguida, na Seção 7.2, uma Tabela contendo a comparação entre os trabalhos levantados no Estado da Arte e a abordagem apresentada neste trabalho é apresentada.

7.1 Análise dos Resultados Obtidos

Nesta Seção, o trabalho de pesquisa apresentado neste manuscrito é avaliado de acordo com os critérios elaborados na Seção 3.3. Os resultados obtidos na pesquisa científica descrita neste manuscrito permitem afirmar que a abordagem proposta para auxiliar a atividade de desenvolvimento de *software* em *Smart Grids* atendem aos critérios apresentados na Seção 3.3, como é demonstrado a seguir:

1. *É uma abordagem baseada em MDE.* Um *framework* baseado em MDE (FMDE4SGRID) é fornecido para auxiliar a atividade de desenvolvimento de *software* no domínio de *Smart Grids*. Em MDE, o modelo é o principal artefato de desenvolvimento de *software*. Talvez não haja um "*checklist*" preciso e infalível para verificar se uma abordagem é baseada em MDE ou não. Porém, existe um pequeno conjunto de princípios que são bem aceitos na literatura e que podem ser utilizados quando é necessário fazer esse tipo de análise. São seis princípios no total. Três deles são apresentados por Bézivin (2005) (ver a subseção 2.4.3) e consistem em: 1) Tudo é modelo; 2) Conformidade (todo modelo é conforme um metamodelo); e 3) Representação. Os outros três são apresentados por Booch et al. (2004) e consistem nos fundamentos básicos da MDA (ver subseção 2.4.4). São eles: 1) Representação direta; 2) Automação (transformação de modelos) e 3) Utilização de padrões abertos. O FMDE4SGRID atende a todos esses princípios e fundamentos. Todos os artefatos dentro do FMDE4SGRID são modelos e todos os modelos são conforme um metamodelo (princípios da conformidade e "tudo é modelo"). Os modelos utilizados dentro do FMDE4SGRID têm como objetivo representar sistemas, por exemplo, *Smart Grids*, sistemas de distribuição de energia elétrica, sistemas baseados em DDS para comunicação, entre outros (princípios de Representação e Representação direta).

O FMDE4SGRID faz uso das transformações de modelo-a-modelo e modelo-a-texto para automatizar o processo de desenvolvimento (fundamento de Automação). Por fim, o FMDE4SGRID se apoia em padrões abertos que servem como base para a construção dos modelos, metamodelos e da própria arquitetura do FMDE4SGRID, por exemplo, a MDA, o XMI, o SGAM, o DDS e o CIM;

2. *Auxilia na análise e definição das funcionalidades do software.* No FMDE4SGRID, as funcionalidades de *software* podem ser registradas no PIM, que é conforme o metamodelo de *Smart Grids*. Este metamodelo é baseado nas camadas do SGAM. Sendo assim, o usuário pode definir quais são as funcionalidades da aplicação de *Smart Grids* ao criar *activities* dentro da camada de *Smart Grid Function*; pode especificar os tipos de dados que são manipulados pela aplicação ao criar tipos de dados primitivos (*PrimitiveData*) ou complexos (*ComplexData*) dentro da camada de *Smart Grid Information*; o usuário pode também especificar entidades que realizam a comunicação (*CommunicationEntity*) na camada de *Smart Grid Comunication*. Com isso, o FMDE4SGRID fornece ao desenvolvedor uma ferramenta para fazer a especificação do PIM da aplicação, que é a própria DSL proposta.

3. *Auxilia no projeto da arquitetura de software.* Na demonstração da utilização do FMDE4SGRID, uma das fases de desenvolvimento propostas foi a Fase de Projeto (subseção 6.3.2), onde a arquitetura de *software* específica para a plataforma DDS é desenvolvida. A implementação do *software* em plataformas específicas é representada pelo PSM Abstrato e o PSM Concreto. O FMDE4SGRID suporta a geração automática do PSM Abstrato e do PSM Concreto a partir da execução de definições de transformação. Na implementação do FMDE4SGRID apresentada neste trabalho, o PSM Abstrato é conforme o metamodelo de DDS e o PSM Concreto é conforme o metamodelo de Java. O metamodelo de DDS proposto neste trabalho se mostrou eficiente ao capturar não apenas os aspectos de orientação a objetos das aplicações, como as classes, os métodos, as interfaces e os atributos, mas também aspectos pertinentes à configuração da plataforma de DDS utilizada (*OpenDDS*), com a especificação das configurações de descoberta de participantes de domínio e transporte de dados, além dos módulos IDL que definem os tópicos de DDS utilizados pelas aplicações. Além disso, o metamodelo de DDS proposto também carrega a definição do *TopicHandler* (ver subseção 5.1.2), que é uma API que abstrai as funcionalidades disponíveis na API de DDS;

4. *Auxilia na implementação do software.* O FMDE4SGRID suporta a geração automática do esqueleto do código fonte das aplicações e dos arquivos de configuração utilizados pela plataforma *OpenDDS*. Durante a implementação das aplicações, houve a necessidade de se complementar o código fonte manualmente, pois na implementação do FMDE4SGRID apresentada neste manuscrito, a geração automática de código

fonte compreende apenas as definições das classes, interfaces e a assinatura dos métodos, com a identificação dos tipos. Por outro lado, a geração automática dos arquivos de configuração para a plataforma *OpenDDS*, especialmente o arquivo IDL, se mostrou muito eficaz e útil, já que a transformação do modelo CIM (descrito em XML *Schema*) em IDL pode ser um processo muito dispendioso e propenso a erro se for realizado manualmente;

5. *Utiliza transformações de modelo-a-modelo.* O FMDE4SGRID, proposto neste trabalho, especifica que uma transformação de modelo-a-modelo deve ser utilizada para gerar o PSM Abstrato a partir dos modelos de entrada, isto é, o PIM, o *Weaving* e o PDM; também é especificado que uma transformação de modelo-a-modelo deve ser utilizada para gerar automaticamente o PSM Concreto a partir do PSM Abstrato. Na implementação do FMDE4SGRID, a linguagem de transformação de modelos QVTo foi utilizada para escrever as definições de transformação. No entanto, outras linguagens de transformação podem ser utilizadas no contexto do FMDE4SGRID, como a ATL;

6. *Fornece uma metodologia para implementar a abordagem proposta.* Uma metodologia foi fornecida em forma de um diagrama de atividades (ver Seção 4.4) para a correta utilização do FMDE4SGRID. Na experimentação do FMDE4SGRID, a metodologia proposta foi adaptada a um processo simplificado de desenvolvimento de *software* baseado no modelo em cascata. Assim, as etapas que fazem parte do processo de aplicação do FMDE4SGRID foram inseridas dentro das fases de Análise, Projeto e Codificação, como mostra a Figura 6.4, do Capítulo anterior;

7. *Auxilia o desenvolvimento de aplicações completas.* O FMDE4SGRID foi pensado para ser utilizado para auxiliar o desenvolvimento de aplicações completas. Isso quer dizer que as aplicações desenvolvidas reúnem os componentes necessários para que elas sejam consideradas sistemas independentes. Exemplos ilustrativos de aplicações utilizadas no contexto de *Smart Grids* como o MDMS, o AMS e o CDMS foram desenvolvidos com o auxílio do FMDE4SGRID. As aplicações desenvolvidas contém apenas os seus componentes básicos de comunicação e acesso a arquivos e bancos de dados. Porém, tanto a arquitetura do FMDE4SGRID quanto os metamodelos utilizados podem ser estendidos para incluir outros aspectos de *software* ou considerar outras plataformas de implementação. Com isso, aplicações com mais recursos aplicadas a sistemas elétricos reais podem ser desenvolvidas futuramente;

8. *Utiliza padrões internacionais do domínio de sistemas de energia elétrica para a representação dos dados.* Na abordagem apresentada neste trabalho, o CIM para sistemas de energia elétrica foi utilizado para descrever a rede elétrica que é monitorada pelas aplicações desenvolvidas com o auxílio do FMDE4SGRID. O CIM

da rede elétrica escolhida para o desenvolvimento dos exemplos foi desenvolvido com a ferramenta *CIMTool*, que cria perfis do modelo e exporta estes perfis em vários formatos. O CIM criado foi exportado em formato XML *Schema* e, no FMDE4SGRID, o CIM foi importado como um XML *Schema* que é conforme o metamodelo de XML *Schema*. Desta forma, os tipos de dados gerados com a execução das transformações de modelos são obtidos a partir do CIM, o que facilita a interoperabilidade entre as aplicações;

9. *Possui uma clara separação entre a modelagem de dados e a lógica de negócio a nível de modelos*. O FMDE4SGRID proporciona uma clara separação entre o modelo da rede elétrica conforme o padrão CIM (capturado pelo PDM) e a modelagem da lógica de negócio das aplicações (capturada pelo PIM) que é conforme o metamodelo de *Smart Grids*. Estes dois modelos são independentes, porém relacionados. Parte dos tipos de dados que são utilizados pelo PIM, principalmente aqueles que descrevem os elementos dos sistemas de energia elétrica, são definidos no PDM. As relações entre os elementos do PIM e os tipos de dados definidos no PDM são especificadas formalmente através do modelo de *Weaving*. Nas experimentações realizadas, o mesmo PDM foi utilizado para todas as aplicações desenvolvidas. No modelo de *Weaving* de cada aplicação, as correspondências entre os tipos de dados no PIM e os tipos de dados contidos no PDM foram definidas. Nos modelos específicos de plataforma, código fonte e arquivos de configuração gerados pelas transformações, os *datatypes* obtidos foram precisamente aqueles definidos no PDM. Ou seja, todas as aplicações desenvolvidas utilizam a mesma estrutura de dados para representar a rede elétrica. A nível de modelo, o CIM utilizado continua independente do PIM da aplicação, e vice-versa. No contexto do FMDE4SGRID, outro modelo de dados da rede elétrica poderia ser utilizado, até mesmo um modelo conforme outra norma, bastando, para isso, alterar o modelo de *Weaving*.

10. *Fornece uma abordagem para suportar a integração de aplicações em Smart Grids*. Na implementação proposta para o FMDE4SGRID, uma arquitetura para integração de aplicações baseada em *middleware* DDS foi considerada. A plataforma escolhida para o desenvolvimento das aplicações foi uma implementação em código aberto de DDS, chamada *OpenDDS*. Nesta plataforma, as aplicações trocam informações ao modificar os valores de um objeto cujo estado é mantido pelo *middleware*. Por exemplo, se o tópico de DDS é definido por um objeto do tipo "*Data*", quer dizer que todos as aplicações que subscrevem a este tópico recebem as modificações no objeto "*Data*" feitas por aplicações que publicam neste mesmo tópico. A ideia proposta neste trabalho é que, em *Smart Grids*, o objeto "*Data*" contém a topologia de uma rede elétrica, com os seus dispositivos, equipamentos e instalações. Desta forma, o objeto que contém todas as informações da rede elétrica é compartilhado por todas

as aplicações de *Smart Grids*. O *middleware* OpenDDS é o responsável por manter o estado deste objeto compartilhado. Nos exemplos ilustrativos implementados, o MDMS publica no *OpenDDS* as medições dos parâmetros de transformadores e instalações de consumidores comerciais e residenciais, enquanto o AMS e o CDMS subscrevem ao mesmo tópico no *OpenDDS*, para que ambos recebam estas atualizações.

7.2 Comparação com os Trabalhos Relacionados

A Tabela 7.1 contém uma comparação entre a abordagem proposta neste trabalho para suportar o desenvolvimento de *software* para *Smart Grids* e as abordagens propostas nos trabalhos relacionados. Os mesmos critérios apresentados na Seção 3.3 são retomados aqui para elaborar as comparações. A coluna mais à direita da Tabela 7.1 com as informações em negrito contém a análise da proposta do FMDE4SGRID apresentada neste trabalho.

Baseado nas comparações apresentadas na Tabela 7.1, é possível afirmar que várias pesquisas na literatura têm investigado a aplicação de MDE para auxiliar o desenvolvimento de *software* no domínio de *Smart Grids* nos últimos anos. Muitos avanços foram realizados na aplicação de MDE para *Smart grids*, especialmente no que se refere ao tratamento das normas do domínio de sistemas de energia elétrica (como o CIM e o IEC 61850) como modelos que podem ser processados e manipulados por computadores e que são conforme um metamodelo. O trabalho de Andrén et al. (2014), por exemplo, apresenta um estudo sobre a transformação de modelos conforme o IEC 61850 em modelos conforme o IEC 61499, que descrevem aplicações de automação para sistemas de energia elétrica. O trabalho de Andrén et al. (2014), no entanto, apresenta limitações principalmente com relação ao escopo dos modelos utilizados, que não representam aplicações completas de *Smart Grids*.

Os trabalhos de Dänekas et al. (2014), Ebeid et al. (2016), Andrén et al. (2016) e Fischinger et al. (2019) propõem abordagens e *frameworks* baseados em MDE que auxiliam o desenvolvimento de aplicações completas para *Smart Grids*, dando suporte à análise e definição das funcionalidades, ao projeto da arquitetura e à implementação de *software*; dentre estas abordagens, apenas a abordagem apresentada por Fischinger et al. (2019) não utiliza transformações automáticas de modelo-a-modelo ou não deixa claro como as transformações são realizadas. Uma limitação encontrada em todos estes trabalhos, incluindo o trabalho de Andrén et al. (2014), é que eles não fazem uma clara separação entre o modelo da rede elétrica e o modelo da lógica de negócio das aplicações desenvolvidas. Por "clara separação" entende-se que os modelos são desenvolvidos e mantidos independentemente um do outro e que existe uma abordagem ou metodologia

Tabela 7.1 – Comparação da abordagem proposta com os trabalhos relacionados.

Critérios de avaliação	Andrén et al. (2014)	Dänekas et al. (2014)	Ebeid et al. (2016)	Andrén et al. (2016)	Souvent et al. (2019)	Fischinger et al. (2019)	**Abordagem Proposta**
É uma abordagem baseada em MDE?	Sim	Sim	Sim	Sim	Não	Sim	**Sim**
Auxilia na análise e definição das funcionalidades do *software*?	Não	Sim	Sim	Sim	Não	Sim	**Sim**
Auxilia no projeto da arquitetura de *software*?	Não	Sim	Sim	Sim	Sim	Sim	**Sim**
Auxilia a implementação do *software*?	Sim	Sim	Sim	Sim	Não	Sim	**Sim**
Utiliza transformações de modelo-a-modelo?	Sim, QVT	Não	Sim	Sim, QVT	Não	Não	**Sim, QVT**
Uma metodologia para implementar a abordagem proposta é fornecida?	Sim	Sim	Sim	Sim	Não	Sim	**Sim**
Auxilia o desenvolvimento de aplicações completas?	Não	Sim	Sim	Sim	Não	Sim	**Sim**
Utiliza padrões internacionais para representação dos dados?	Sim	Sim	Não	Sim	Sim	Não	**Sim**
Possui uma clara separação entre a modelagem de dados e a lógica de negócio?	Não	Não	Não	Não	Sim	Não	**Sim**
Fornece uma abordagem para suportar a integração de aplicações em *Smart Grids*?	Não	Não	Não	Não	Sim, em teoria	Sim, na prática	**Sim, na prática**

para estabelecer relações entre os dois modelos sem modificá-los.

Dentre todos os trabalhos analisados, o trabalho de Souvent et al. (2019) é o único que aborda esta questão da separação de preocupações com uma metodologia bem definida. Souvent et al. (2019) propõem uma arquitetura de sistemas na qual a definição, o desenvolvimento e a manutenção do modelo de dados da rede elétrica (conforme o CIM) é feita independentemente das aplicações de *Smart Grids*. Souvent et al. (2019) propõem que as aplicações de *Smart Grids* utilizem interfaces ou *wrappers* para acessar os modelos definidos conforme CIM. No entanto, Souvent et al. (2019) não demonstram como os elementos das aplicações fazem referência aos elementos do CIM na prática e, além disso, uma abordagem baseada em MDE para auxiliar no desenvolvimento de *software* para *Smart Grids* não é proposta. Como foi demonstrado na implementação do FMDE4SGRID, no Capítulo 5, e no desenvolvimento dos exemplos ilustrativos no Capítulo 6, o trabalho apresentado nesta dissertação fornece uma abordagem baseada em MDE para auxiliar no desenvolvimento de aplicações completas para *Smart Grids*, fornecendo um mecanismo formal baseado em *Weaving* de modelos para fazer a separação entre o modelo da lógica de negócio (PIM) e o modelo da rede elétrica (PDM).

Outra questão que pode ser levantada a partir da leitura da Tabela 7.1 é que a maioria dos trabalhos analisados não propõem uma abordagem para realizar a *integração de várias aplicações para Smart Grids*. A integração de aplicações é fundamental no contexto de *Smart Grids* devido à grande quantidade de dados e à heterogeneidade das diferentes aplicações, equipamentos, dispositivos e modelos de rede elétrica. Souvent et al. (2019) demonstram uma arquitetura beseada em ESB para integração de aplicações, porém a implementação desta arquitetura não é demonstrada na prática; Fischinger et al. (2019) afirmam que, em sua abordagem, um *middleware* é utilizado para integrar as aplicações desenvolvidas. Porém, Fischinger et al. (2019) não deixam claro como as aplicações representam a informação ou se algum padrão é utilizado para servir como base para a representação dos dados, ou seja, a interoperabilidade de dados não é garantida. No trabalho apresentado neste manuscrito, o FMDE4SGRID auxilia na implementação de aplicações para *Smart Grids* na plataforma *OpenDDS*, que facilita a integração de aplicações e fornece o compartilhamento da informação por meio de um *middleware* centrado em dados. Ao mesmo tempo, o padrão CIM é utilizado para representar os dados do sistema de energia elétrica monitorado, facilitando a interoperabilidade de dados.

7.3 Síntese

Neste capítulo, uma análise dos resultados obtidos com a realização deste trabalho é feita. A abordagem proposta nesta dissertação para auxiliar o desenvolvimento de *software* e a integração de aplicações em *Smart Grids* foi submetida aos critérios elaborados no

Capítulo 3 para comparação com os trabalhos relacionados.

Baseado nos resultados obtidos durante a definição, a prototipação e implementação prática do FMDE4SGRID, pode-se afirmar que a abordagem proposta neste trabalho atende aos critérios de comparação levantados.

Dentre os trabalhos da literatura que foram analisados, duas limitações foram destacadas: 1) a falta de uma separação clara entre a modelagem da lógica de negócio das aplicações de *Smart Grids* e a modelagem dos dados da rede elétrica; 2) e a falta de abordagens para realizar a integração de aplicações em *Smart Grids*.

A realização do presente trabalho buscou, portanto, encontrar soluções para as limitações encontradas. O FMDE4SGRID é um *framework* baseado nos princípios da MDE que auxilia de forma semiautomática (transformações de modelo-a-modelo e modelo-a-texto são utilizadas) na análise, no projeto e na implementação de aplicações completas para *Smart Grids*. O FMDE4SGRID suporta a utilização do padrão CIM para sistemas de energia elétrica para a representação dos dados do sistema elétrico e faz uma clara separação, por meio da técnica de *Weaving* de modelos, entre a modelagem das aplicações e a modelagem da rede elétrica, o que facilita a interoperabilidade. Além disso, a integração de aplicações é suportada na implementação do FMDE4SGRID, através da escolha de um *middleware* centrado em dados – o *OpenDDS* – para fazer a distribuição da informação.

8 Conclusões e Trabalhos Futuros

Neste trabalho, um *framework* baseado em MDE e *weaving* de modelos para auxiliar o desenvolvimento de aplicações de *software* no domínio de *Smart Grids* – o FMDE4SGRID – foi apresentado. O FMDE4SGRID contém metamodelos, transformações entre modelos, geração de código e uma arquitetura definida para a organização destes elementos. Uma prototipagem do FMDE4SGRID no EMF (*Eclipse Modeling Framework*) foi implementada e foi validada com o desenvolvimento de aplicações simplificadas para o domínio de *Smart Grids*. A aplicações desenvolvidas foram integradas por meio do uso do *middleware OpenDDS*. Os dados compartilhados entre as aplicações são conforme o padrão CIM para sistemas de energia elétrica.

A seguir, na Seção 8.1, os objetivos alcançados com o desenvolvimento deste trabalho são apontados. Em seguida, na Seção 8.2, as contribuições científicas e tecnológicas são apresentadas e, por fim, a Seção 8.3 e a Seção 8.4 tratam das limitações da abordagem proposta e das sugestões para trabalhos futuros, respectivamente.

8.1 Objetivos Alcançados

Nesta seção, os objetivos alcançados com a realização deste trabalho são apresentados. Os meios utilizados para alcançar cada objetivo específico deste trabalho (ver Seção 1.7) são discutidos na lista a seguir.

- *Criar um framework baseado em MDE e weaving que dê suporte ao desenvolvimento de software para Smart Grids.* Este objetivo específico foi atingido com a criação do FMDE4SGRID, um *framework* desenvolvido de acordo com os princípios da MDE e que foi definido por meio de uma arquitetura que define quais metamodelos são utilizados e como os modelos e definições de transformação se relacionam. Os metamodelos de *Smart Grids* e *Weaving* foram propostos para apoiar a abordagem do FMDE4SGRID;

- *Criar uma metodologia para servir como guia para a utilização do framework proposto.* Uma metodologia especificando uma sequência de etapas para a correta utilização do FMDE4SGRID foi proposto. A etapas elaboradas consistem, primeiramente, na elaboração manual ou reutilização do PIM, PDM e *weaving*. O PIM e o PDM são elaborados em etapas distintas. O *weaving* é construído após o PIM e o PDM estarem prontos. O processo criado contém também as etapas de geração automática do PSM Abstrato, do PSM Concreto e do código fonte e arquivos de configuração da aplicação. Algumas etapas de edição manual dos modelos são intercaladas com

as etapas de geração automática dos modelos e do código fonte. Desta forma, a abordagem proposta é considerada *semiautomática*;

- *Criar um metamodelo de Smart Grids para a definição de modelos de lógica de negócio para aplicações no domínio de Smart Grids.* Um metamodelo de *Smart Grids* foi desenvolvido para apoiar a abordagem do FMDE4SGRID. Este metamodelo se baseia no modelo de arquitetura do SGAM e, desta forma, tem como objetivo especificar aplicações para *Smart Grids* a partir da definição de camadas de interoperabilidade. Portanto, o metamodelo de *Smart Grids* proposto contém elementos correspondentes às camadas do SGAM. Por exemplo, o elemento *Smart Grid Function* contém a especificação das funcionalidades da aplicação com os seus respectivos parâmetros; o elemento *Smart Grid Information* contém os tipos de dados que são compartilhados com outras aplicações de *Smart Grids*; o elemento *Smart Grid Communication* também é definido dentro do metamodelo proposto. O metamodelo de *Smart Grid* se mostrou eficaz para a especificação rápida de aplicações. Porém, talvez uma maior expressividade seja necessária para considerar mais aspectos das aplicações desenvolvidas. Por exemplo, o elemento *Smart Grid Communication* foi pouco explorado, assim como o elemento *BusinessLogic*, que é inspirado na camada de *Business Layer* do SGAM;

- *Criar um metamodelo de DDS para a definição de modelos de aplicações para plataformas baseadas em DDS.* Um metamodelo de DDS é fornecido neste trabalho. O metamodelo de DDS proposto pode ser dividido em quatro partes: um pacote de *software* contendo os conceitos necessários para descrever aplicações orientadas à objeto; um pacote contendo os elementos do *TopicHandler*; um conjunto de configurações de DDS para a descoberta de participantes e de transporte de dados; e, por fim, a definição de módulos IDL que são utilizados para especificar as estruturas de dados utilizadas pelos tópicos de DDS. Dentre estas quatro partes apresentadas, a parte de configurações de descoberta e transporte foi pouco explorada neste trabalho, já que apenas as configurações padrão da plataforma foram utilizadas;

- *Desenvolver um metamodelo de weaving que atenda às necessidades de separação de preocupações dentro do domínio de Smart Grids, principalmente a separação entre o CIM da rede elétrica e o PIM da aplicação.* Este objetivo foi alcançado por meio do desenvolvimento de um metamodelo de *weaving* como uma extensão dos metamodelos de *weaving* apresentados por Junior (2017) e Fabro et al. (2006). O metamodelo de *weaving* proposto neste trabalho suporta a especificação de relações entre os tipos de dados definidos no PIM e os tipos de dados definidos no PDM. A transformação de modelos que gera o PSM Abstrato leva em considerações as relações especificadas no PIM para construir os tipos de dados correspondentes;

- *Desenvolver uma arquitetura de middleware baseada em DDS que dê suporte à integração de aplicações para Smart Grids.* Uma arquitetura estilo ESB baseada em DDS e CIM foi implementada neste trabalho. Esta arquitetura de comunicação permite o compartilhamento de informações de objetos da rede elétrica entre as aplicações de *Smart Grids*. O *middleware OpenDDS* foi utilizado para implementar o esquema de comunicação proposto;

- *Criar uma prototipação do framework proposto em ambiente Eclipse.* O FMDE4SGRID foi prototipado em ambiente Eclipse por meio da utilização das ferramentas disponíveis no EMF, como: *Ecore*, *Genmodel*, QVTo e o *Acceleo*. O *Ecore* foi a linguagem de metamodelagem utilizada para desenvolver os metamodelos propostos no contexto do FMDE4SGRID; o *Genmodel* é uma ferramenta do EMF que gera o código Java que habilita os editores de modelos no EMF; O QVTo foi a linguagem de transformação de modelos utilizada para o desenvolvimento das definições de transformação; o *Acceleo* foi a ferramenta utilizada para desenvolver a geração de código;

- *Projetar e implementar, com auxílio do framework proposto, aplicações para análise de dados em Smart Grids.* Para alcançar este objetivo específico, versões simplificadas de três aplicações conhecidas do domínio de *Smart Grids* foram idealizadas. As aplicações foram desenvolvidas utilizando uma metodologia de desenvolvimento de *software* baseada no modelo em cascata. O FMDE4SGRID auxiliou no desenvolvimento das aplicações nas fases de Análise, Projeto e Codificação. O modelo da topologia de uma rede elétrica de distribuição utilizada como exemplo foi elaborado conforme o padrão CIM. Um sistema distribuído foi elaborado para testar e validar as aplicações desenvolvidas. Dados de consumo de energia elétrica de usuários residenciais e comerciais, além de dados de monitoramento dos transformadores da rede elétrica foram gerados e publicados no *middleware OpenDDS* e as aplicações desenvolvidas fizeram a leitura e o tratamento destes dados conforme o especificado.

8.2 Contribuições Científicas e Tecnológicas

A abordagem proposta nesta dissertação apresenta as seguintes contribuições científicas:

- Proposta de um abordagem para auxiliar a atividade de desenvolvimento de *software* para *Smart Grids* que tem como premissa a separação entre a modelagem da lógica de negócio das aplicações e a modelagem da rede elétrica;

- Implementação de uma abordagem, com auxílio do *framework* proposto, para realizar a integração de aplicações de *Smart Grids* através do uso do *middleware* DDS;

- Uma extensão aos metamodelos de *weaving* propostos por Junior (2017) e Fabro et al. (2006) para especificar a relação entre os *data types* do modelo da lógica de negócio das aplicações de *Smart Grids* e os *data types* definidos em um modelo conforme um padrão do IEC.

As contribuições tecnológicas são as seguintes:

- Proposta de um *framework* para auxiliar a atividade de desenvolvimento de *software* e integração de aplicações para *Smart Grids*;
- Proposta de uma DSL na forma de um metamodelo para descrever aplicações de *Smart Grids*.
- Definição e implementação de uma API chamada *TopicHandler* que abstrai a API de DDS;
- Proposta de um metamodelo de DDS que leva em conta vários aspectos como o *TopicHandler*, as configurações da plataforma e os módulos IDL que descrevem os tópicos.

8.3 Limitações

As principais limitações deste trabalho são listadas a seguir.

- O metamodelo de *Smart Grids* proposto ainda precisa evoluir. Este metamodelo é baseado nas camadas de interoperabilidade do SGAM. No entanto, nem todas as camadas estão representadas, por exemplo, a camada de componentes (*Component Layer*). Além disso, os elementos do metamodelo que representam as camadas de comunicação (*Communication Layer*) e de regras de negócios (*Business Layer*) possuem pouca expressividade e foram pouco utilizadas durante o desenvolvimento das aplicações;
- Ainda não há uma ferramenta que inclua todas as funcionalidades do FMDE4SGRID. Todos os elementos do FMDE4SGRID foram implementados utilizando projetos separados dentro do *workspace* do Eclipse, o que gera um trabalho adicional de configurar manualmente os diretórios e os caminhos dos arquivos dos modelos;
- A abordagem proposta ainda carece de uma investigação sobre a usabilidade do FMDE4SGRID e os ganhos potenciais que a utilização da abordagem pode trazer em relação ao tempo necessário para desenvolver as aplicações e a qualidade do *software* produzido;

- Como a implementação atual do FMDE4SGRID não suporta a especificação do comportamento dos métodos, a geração de código fonte se limita a gerar o esqueleto do código apenas;
- O escopo das aplicações desenvolvidas é bem limitado, pois o objetivo das aplicações desenvolvidas neste trabalho foi dar um exemplo da utilização do FMDE4SGRID na prática;
- As aplicações desenvolvidas nos exemplos ilustrativos não utilizam dados reais de redes elétricas obtidos em tempo real. Os dados utilizados para o teste das aplicações são gerados a partir de arquivos que contém os dados de consumo de energia elétrica dos consumidores e os dados de monitoramento dos transformadores.

8.4 Sugestões para Trabalhos Futuros

Para trabalhos futuros, as seguintes sugestões são feitas:

- Estender o metamodelo de *Smart Grids* para que ele possa suportar a representação de mais requisitos de comunicação e para que ele se aproxime mais dos conceitos do SGAM;
- Desenvolvimento de um *plugin* para Eclipse contendo: todos os metamodelos definidos no FMDE4SGRID e as definições de transformação; uma interface gráfica para o usuário interagir com mais facilidade com as atividades do FMDE4SGRID, inclusive com botões para carregar ou importar modelos e executar as transformações de modelos;
- Utilizar o FMDE4SGRID para implementar as aplicações em outras plataformas, não apenas as plataformas de comunicação como o *OpenDDS*. Seria interessante, por exemplo, desenvolver implementações do FMDE4SGRID para plataformas de *Big Data*, como o *Spark* ou o *Hadoop*;
- Conduzir uma pesquisa de um estudo de caso para investigar o quanto o FMDE4SGRID pode melhorar o processo de desenvolvimento de *software* comparando com outras abordagens da literatura;
- Indroduzir aos metamodelos do FMDE4SGRID a capacidade de representar o comportamento dos métodos e operações, para que o código fonte possa ser gerado de forma mais completa;
- Desenvolver aplicações de *software* maiores em *Smart Grids* para resolver problemas reais do setor dos sistemas de energia elétrica, inclusive testando as aplicações desenvolvidas em instalações elétricas reais;

- Avaliação do *framework* proposto por parte dos desenvolvedores de aplicações para o domínio de *Smart Grids*.

8.5 Publicações

Este trabalho rendeu a publicação de um artigo em uma conferência internacional. O artigo intitulado "*A Framework Based on Model Driven Engineering and Model Weaving to Support Data-Driven Interoperability for Smart Grid Application*" foi publicado em "*Proceedings of the 2020 European Symposium on Software Engineering* - ESSE 2020". O artigo está disponível na biblioteca digital da ACM, no link: <https://dl.acm.org/doi/10.1145/3393822.3432341>.

Referências

ALABEYI, B. *Data Centricity vs. Message Centricity*. 2015. Disponível em: <https://www.rti.com/blog/data-centricity-vs-message-centricity/>. Acesso em: 11/12/2020. Citado na página 43.

ALI, H.; MAMUN, A. A.; ANWAR, S. A comprehensive study of advancement of electrical power grid and middleware based smart grid communication platform. *International Journal of Advancements in Technology*, v. 7, n. 2, p. 4, 2016. Citado na página 24.

ALMEIDA, J. P.; DIJKMAN, R.; SINDEREN, M. V.; PIRES, L. F. On the notion of abstract platform in mda development. In: IEEE. *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004.* [S.l.], 2004. p. 253–263. Citado na página 86.

AMBROSE, J. Renewable energy breaks UK record in first quarter of 2020. *The Guardian*, 2020. Acesso em 11/12/2020. Citado na página 21.

ANDERSON, R. N.; BOULANGER, A.; POWELL, W. B.; SCOTT, W. Adaptive stochastic control for the smart grid. *Proceedings of the IEEE*, IEEE, v. 99, n. 6, p. 1098–1115, 2011. Citado na página 27.

ANDRÉN, F.; STRASSER, T.; KASTNER, W. Model-driven engineering applied to smart grid automation using iec 61850 and iec 61499. In: IEEE. *2014 Power Systems Computation Conference*. [S.l.], 2014. p. 1–7. Citado 10 vezes nas páginas 9, 17, 24, 25, 67, 68, 79, 80, 150 e 151.

ANDRÉN, F.; STRASSER, T.; KASTNER, W. From textual programming to IEC 61499 artifacts: Towards a model-driven engineering approach for smart grid applications. In: IEEE. *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. [S.l.], 2015. p. 1524–1530. Citado 2 vezes nas páginas 23 e 25.

ANDRÉN, F. P.; STRASSER, T.; KASTNER, W. Applying the sgam methodology for rapid prototyping of smart grid applications. In: IEEE. *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. [S.l.], 2016. p. 3812–3818. Citado 10 vezes nas páginas 9, 17, 25, 27, 72, 73, 79, 80, 150 e 151.

ANDRÉN, F. P.; STRASSER, T. I.; KASTNER, W. Engineering smart grids: Applying model-driven development from use case design to deployment. *Energies*, Multidisciplinary Digital Publishing Institute, v. 10, n. 3, p. 374, 2017. Citado 2 vezes nas páginas 25 e 39.

ANEEL. *Resolução Normativa Nº 414*. [S.l.], 2010. Citado na página 124.

ANTON, S. D.; FRAUNHOLZ, D.; LIPPS, C.; POHL, F.; ZIMMERMANN, M.; SCHOTTEN, H. D. Two decades of scada exploitation: A brief history. In: IEEE. *2017 IEEE Conference on Application, Information and Network Security (AINS)*. [S.l.], 2017. p. 98–104. Citado na página 37.

ARNOLD, G. W. Challenges and opportunities in smart grid: A position article. *Proceedings of the IEEE*, IEEE, v. 99, n. 6, p. 922–927, 2011. Citado 5 vezes nas páginas 20, 21, 33, 36 e 37.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. Citado na página 75.

BELANGOUR, A.; BÉZIVIN, J.; FREDJ, M. Towards a new software development process for mda. *Milestones, Models and Mappings for Model-Driven Architecture*, p. 1, 2006. Citado 4 vezes nas páginas 25, 27, 59 e 60.

BÉZIVIN, J. Model driven engineering: An emerging technical space. In: SPRINGER. *International Summer School on Generative and Transformational Techniques in Software Engineering*. [S.l.], 2005. p. 36–64. Citado 9 vezes nas páginas 24, 47, 48, 49, 53, 54, 61, 63 e 146.

BÉZIVIN, J.; BOUZITOUNA, S.; FABRO, M. D. D.; GERVAIS, M.-P.; JOUAULT, F.; KOLOVOS, D.; KURTEV, I.; PAIGE, R. F. A canonical scheme for model composition. In: SPRINGER. *European Conference on Model Driven Architecture-Foundations and Applications*. [S.l.], 2006. p. 346–360. Citado na página 60.

BOEHM, B. A view of 20th and 21st century software engineering. In: *Proceedings of the 28th international conference on Software engineering*. [S.l.: s.n.], 2006. p. 12–29. Citado 2 vezes nas páginas 45 e 46.

BOOCH, G.; BROWN, A.; IYENGAR, S.; RUMBAUGH, J.; SELIC, B. *The IBM MDA Manifesto The MDA Journal, May 2004*. 2004. Citado 5 vezes nas páginas 47, 50, 51, 52 e 146.

BOX, D.; EHNEBUSKE, D.; KAKIVAYA, G.; LAYMAN, A.; MENDELSOHN, N.; F, N. H.; THATTE, S.; WINER, D. *Simple Object Access Protocol (SOAP) 1.1*. 2000. Disponível em: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Acesso em 20/12/2020. Citado na página 50.

BROOKS, F. P. No silver bullet essence and accidents of software engineering. *Computer*, v. 20, n. 4, p. 10–19, 1987. Citado na página 46.

CARVALHO, M. V.; LOPES, D.; ABDELOUAHAB, Z. A framework based on model driven engineering to support schema merging in database systems. In: *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*. [S.l.]: Springer, 2015. p. 397–405. Citado na página 60.

CEN-CENELEC-ETSI. *Smart Grid Reference Architecture*. [S.l.], 2012. Acesso em 11/12/2020. Citado 5 vezes nas páginas 25, 39, 40, 87 e 88.

CORRÊA, R. d. M. *Avaliação do Impacto da Conexão de Veículos Elétricos em Sistemas de Potência*. 2018. Monografia. Universidade Federal do Maranhão. Citado na página 35.

CZARNECKI, K.; HELSEN, S. Classification of model transformation approaches. In: USA. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*. [S.l.], 2003. v. 45, n. 3, p. 1–17. Citado na página 58.

DAHL, O.-J.; DIJKSTRA, E. W.; HOARE, C. A. R. *Structured programming*. [S.l.]: Academic Press Ltd., 1972. Citado na página 46.

DÄNEKAS, C.; NEUREITER, C.; ROHJANS, S.; USLAR, M.; ENGEL, D. Towards a model-driven-architecture process for smart grid projects. In: *Digital enterprise design & management*. [S.l.]: Springer, 2014. p. 47–58. Citado 10 vezes nas páginas 9, 17, 27, 68, 69, 70, 79, 80, 150 e 151.

DIJKSTRA, E. W. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, ACM New York, NY, USA, v. 11, n. 3, p. 147–148, 1968. Citado na página 46.

DIJKSTRA, E. W. The humble programmer. *Communications of the ACM*, ACM New York, NY, USA, v. 15, n. 10, p. 859–866, 1972. Citado 2 vezes nas páginas 46 e 47.

DOE, U. D. of E. *The Smart Grid*. 2020. Disponível em: <https://www.smartgrid.gov/the_smart_grid/smart_grid.html>. Acesso em 26/11/2020. Citado na página 35.

DUPONT, F. H.; GRASSI, F.; ROMITTI, L. Energias renováveis: buscando por uma matriz energética sustentável. *Revista Eletrônica em Gestão, Educação e Tecnologia Ambiental*, v. 19, p. 70–81, 2015. Citado 3 vezes nas páginas 21, 34 e 35.

EBEID, E.; VALOV, M.; JACOBSEN, R. H. Model-driven design approach for building smart grid applications. In: IEEE. *2016 Euromicro Conference on Digital System Design (DSD)*. [S.l.], 2016. p. 260–267. Citado 10 vezes nas páginas 9, 17, 25, 70, 71, 72, 79, 80, 150 e 151.

EC - European Commission. *Standardization Mandate to European Standardization Organizations (ESOs) to Support European Smart Grid Deployment*. 2011. Disponível em: <https://ec.europa.eu/growth/tools-databases/mandates/index.cfm?fuseaction=search.detail&id=475#>. Acesso em 01/03/2021. Citado na página 39.

ECLIPSE. *Eclipse Modeling Framework - EMF*. 2020. Disponível em: <https://www.eclipse.org/modeling/emf/>. Acesso em 28/12/2020. Citado na página 62.

FABRO, M. D. D.; BÉZIVIN, J.; JOUAULT, F.; BRETON, E.; GUELTAS, G. Amw: a generic model weaver. In: *1 ere Journées sur l'Ingénierie Dirigée par les Modèles (IDM05)*. [S.l.: s.n.], 2005. p. 105–114. Citado 5 vezes nas páginas 24, 25, 28, 59 e 60.

FABRO, M. D. D.; BÉZIVIN, J.; VALDURIEZ, P. Weaving models with the eclipse amw plugin. In: *Eclipse Modeling Symposium, Eclipse Summit Europe*. [S.l.: s.n.], 2006. v. 2006, p. 37–44. Citado 10 vezes nas páginas 24, 59, 60, 61, 62, 89, 90, 94, 155 e 157.

FAHEEM, M.; SHAH, S. B. H.; BUTT, R. A.; RAZA, B.; ANWAR, M.; ASHRAF, M. W.; NGADI, M. A.; GUNGOR, V. C. Smart grid communication and information technologies in the perspective of industry 4.0: Opportunities and challenges. *Computer Science Review*, Elsevier, v. 30, p. 1–30, 2018. Citado na página 127.

FANG, X.; MISRA, S.; XUE, G.; YANG, D. Smart grid—The new and improved power grid: A survey. *IEEE communications surveys & tutorials*, IEEE, v. 14, n. 4, p. 944–980, 2011. Citado 2 vezes nas páginas 21 e 36.

FELIX, E. *Uma Unidade de Medição Fasorial Sincronizada de Baixo Custo com Raspberry Pi 3*. 2018. Monografia. Universidade Federal do Maranhão. Citado na página 22.

FERREIRA, M. R. *Desenvolvimento de um Protótipo de Sistema Microprocessado capaz de Realizar Leituras Sincronizadas de uma PMU*. 2017. Monografia. Universidade Federal do Maranhão. Citado na página 22.

FISCHINGER, M.; EGGER, N.; BINDER, C.; NEUREITER, C. Towards a model-centric approach for developing dependable smart grid applications. In: IEEE. *2019 4th International Conference on System Reliability and Safety (ICSRS)*. [S.l.], 2019. p. 1–9. Citado 11 vezes nas páginas 10, 17, 25, 27, 75, 76, 79, 80, 150, 151 e 152.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design patterns: Abstraction and reuse of object-oriented design. In: SPRINGER. *European Conference on Object-Oriented Programming*. [S.l.], 1993. p. 406–431. Citado na página 46.

GHARAVI, H.; GHAFURIAN, R. *Smart grid: The electric energy system of the future*. [S.l.]: IEEE, 2011. Citado 2 vezes nas páginas 21 e 35.

GOLDEMBERG, J.; LUCON, O. Energias renováveis: um futuro sustentável. *Revista USP*, n. 72, p. 6–15, 2007. Citado na página 34.

GUNGOR, V. C.; SAHIN, D.; KOCAK, T.; ERGUT, S.; BUCCELLA, C.; CECATI, C.; HANCKE, G. P. Smart grid technologies: Communication technologies and standards. *IEEE transactions on Industrial informatics*, IEEE, v. 7, n. 4, p. 529–539, 2011. Citado 2 vezes nas páginas 22 e 40.

GUNGOR, V. C.; SAHIN, D.; KOCAK, T.; ERGUT, S.; BUCCELLA, C.; CECATI, C.; HANCKE, G. P. A survey on smart grid potential applications and communication requirements. *IEEE Transactions on industrial informatics*, IEEE, v. 9, n. 1, p. 28–42, 2012. Citado 5 vezes nas páginas 37, 82, 122, 123 e 139.

GWAC, G. A. C. *GridWise Interoperability Context-Setting Framework*. 2008. Disponível em: <https://www.gridwiseac.org/pdfs/interopframework_v1_1.pdf>. Acesso em 09/12/2020. Citado 2 vezes nas páginas 23 e 39.

IEA, I. E. A. *World Energy Outlook Executive Summary*. 2018. Disponível em: <https://webstore.iea.org/download/summary/190?fileName=English-WEO-2018-ES.pdf>. Acesso em 24/11/2020. Citado na página 33.

IEA, I. E. A. *World Energy Outlook Executive Summary*. 2019. Disponível em: <https://iea.blob.core.windows.net/assets/1f6bf453-3317-4799-ae7b-9cc6429c81d8/English-WEO-2019-ES.pdf>,. Acesso em 11/12/2020. Citado na página 21.

IEC. *Function Blocks – Part 1: Architecture*. [S.l.], 2012. Citado na página 23.

IEC. *Energy Management System Application Program Interface (EMS-API) – Part 301: Common Information Model (CIM)*. [S.l.], 2016. Citado 4 vezes nas páginas 22, 40, 41 e 42.

IEC. *Communication Networks and Systems for Power Utility Automation – Part 6: Configuration Description Language for Communication in Power Utility Automation Systems Related to IEDs*. [S.l.], 2018. Citado na página 23.

IEC. *Application integration at electric utilities - System interfaces for distribution management - Part 4: Interfaces for records and asset management*. [S.l.], 2019. Citado 2 vezes nas páginas 22 e 40.

IVANOV, C.; SAXTON, T.; WAIGHT, J.; MONTI, M.; ROBINSON, G. Prescription for interoperability: Power system challenges and requirements for interoperable solutions. *IEEE Power and Energy Magazine*, IEEE, v. 14, n. 1, p. 30–39, 2015. Citado 6 vezes nas páginas 22, 23, 24, 27, 37 e 82.

JOUAULT, F. Loosely coupled traceability for atl. In: CITESEER. *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany*. [S.l.], 2005. v. 91, p. 29–37. Citado na página 60.

JOUAULT, F.; ALLILAIRE, F.; BÉZIVIN, J.; KURTEV, I. Atl: A model transformation tool. *Science of computer programming*, Elsevier, v. 72, n. 1-2, p. 31–39, 2008. Citado 2 vezes nas páginas 57 e 113.

JOY, B.; STEELE, G.; GOSLING, J.; BRACHA, G. *The Java language specification*. [S.l.]: Addison-Wesley Reading, 2000. Citado na página 46.

JUNIOR, O. S. S. *Um Framework para Suportar de Forma Semiautomática a Atividade de Desenvolvimento de Software para MapReduce Utilizando MDE*. Tese (Doutorado) — Universidade Federal do Maranhão, 2017. Citado 19 vezes nas páginas 25, 28, 60, 61, 84, 85, 89, 90, 93, 94, 101, 105, 106, 108, 109, 110, 128, 155 e 157.

KAITOVIC, I.; LUKOVIC, S. Adoption of model-driven methodology to aggregations design in smart grid. In: IEEE. *2011 9th IEEE International Conference on Industrial Informatics*. [S.l.], 2011. p. 533–538. Citado na página 25.

KEZUNOVIC, M. Translational knowledge: From collecting data to making decisions in a smart grid. *Proceedings of the IEEE*, IEEE, v. 99, n. 6, p. 977–997, 2011. Citado na página 22.

KIM, D.-K.; ALAERJAN, A.; LU, L.; YANG, H.; JANG, H. Toward interoperability of smart grids. *IEEE Communications Magazine*, IEEE, v. 55, n. 8, p. 204–210, 2017. Citado 6 vezes nas páginas 22, 24, 25, 27, 83 e 95.

KIM, D.-K.; LEE, B.; KIM, S.; YANG, H.; JANG, H.; HONG, D.; FALK, H. Qvt-based model transformation to support unification of iec 61850 and iec 61970. *IEEE transactions on power delivery*, IEEE, v. 29, n. 2, p. 598–606, 2014. Citado na página 25.

KLEPPE, A. G.; WARMER, J.; WARMER, J. B.; BAST, W. *MDA explained: the model driven architecture: practice and promise*. [S.l.]: Addison-Wesley Professional, 2003. Citado 8 vezes nas páginas 24, 52, 53, 54, 55, 57, 58 e 59.

KRUCHTEN, P. *The rational unified process: an introduction*. [S.l.]: Addison-Wesley Professional, 2004. Citado na página 46.

LEãO, R. *Geração, Transmissão e Distribuição de Energia Elétrica*. 2012. Notas de aula. Universidade Federal do Ceará. Citado na página 124.

LIMA, A. A. S. *Estudo Teórico e Experimental da Medição Fasorial Sincronizada com Aplicação de PIC e GPS*. 2015. Monografia. Universidade Federal do Maranhão. Citado na página 22.

LOPEZ, R.; MOORE, A.; GILLERMAN, J. A model-driven approach to smart substation automation and integration for comision federal de electricidad. In: IEEE. *IEEE PES T&D 2010*. [S.l.], 2010. p. 1–8. Citado na página 25.

MAHONEY, M. S. What makes the history of software hard. *IEEE Annals of the History of Computing*, IEEE, v. 30, n. 3, p. 8–18, 2008. Citado na página 46.

MARLEN, A.; MAXIM, A.; UKAEGBU, I. A.; NUNNA, H. K. Application of big data in smart grids: Energy analytics. In: IEEE. *2019 21st International Conference on Advanced Communication Technology (ICACT)*. [S.l.], 2019. p. 402–407. Citado na página 27.

MATOS, P. L. C. *Um Framework de Segurança Baseado em Engenharia Dirigida por Modelos para Plataformas de Computação em Nuvem: Uma Abordagem para Modelos SaaS*. Dissertação (Mestrado) — Universidade Federal do Maranhão, 2015. Citado 3 vezes nas páginas 25, 28 e 84.

MCMORRAN, A. W. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model*. [S.l.], 2007. Citado 4 vezes nas páginas 40, 42, 43 e 126.

MICROSOFT. *What is .NET?* 2020. Disponível em: <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet>. Acesso em 20/12/2020. Citado na página 50.

MOHAMMAD, R. Ami smart meter big data analytics for time series of electricity consumption. In: IEEE. *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. [S.l.], 2018. p. 1771–1776. Citado na página 22.

MOLINA, M. J.; MOLINA, L. T. Megacities and atmospheric pollution. *Journal of the Air & Waste Management Association*, Taylor & Francis, v. 54, n. 6, p. 644–680, 2004. Citado na página 34.

NEIS, P.; WEHRMEISTER, M. A.; MENDES, M. F. Model driven software engineering of power systems applications: literature review and trends. *IEEE Access*, IEEE, v. 7, p. 177761–177773, 2019. Citado 3 vezes nas páginas 25, 95 e 96.

NIST. *NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0*. 2014. Disponível em: <http://dx.doi.org/10.6028/NIST.SP.1108r3>. Acesso em 01/03/2021. Citado 2 vezes nas páginas 37 e 38.

OCI. *OpenDDS Developer's Guide - OpenDDS version 3.14*. [S.l.], 2020. Acesso em 11/12/2020. Citado 5 vezes nas páginas 26, 45, 96, 103 e 104.

OMG. *Common Object Request Broker Architecture Specification Version 3.3*. 2012. Disponível em: <https://www.omg.org/spec/CORBA/3.3>. Acesso em 20/12/2020. Citado na página 50.

OMG. *Business Process Model and Notation - version 2.0.2*. 2013. Disponível em: <https://www.omg.org/spec/BPMN/2.0.2/PDF>. Acesso em 15/03/2021. Citado 3 vezes nas páginas 87, 88 e 94.

OMG. *Object Constraint Language - version 2.4*. 2014. Disponível em: <https://www.omg.org/spec/OCL/2.4/PDF>. Acesso em 27/12/2020. Citado na página 56.

OMG. *Data Distribution Service (DDS) - version 1.4*. [S.l.], 2015. Acesso em: 11/12/2020. Citado 4 vezes nas páginas 27, 43, 45 e 96.

OMG. *XML Metadata Interchange Specification - version 2.5.1*. 2015. Disponível em: <https://www.omg.org/spec/XMI/2.5.1/PDF>. Acesso em 27/12/2020. Citado na página 56.

OMG. *Meta Object Facility (MOF) Query/View/Transformation Specification - version 1.3*. 2016. Disponível em: <https://www.omg.org/spec/QVT/1.3/PDF>. Acesso em 27/12/2020. Citado 2 vezes nas páginas 57 e 113.

OMG. *Unified Modeling Language Specification - version 2.5.1*. 2017. Disponível em: <https://www.omg.org/spec/UML/2.5.1/PDF>. Acesso em 26/12/2020. Citado na página 55.

OMG. *DDS Security - version 1.1*. [S.l.], 2018. Acesso em: 02/03/2021. Citado na página 45.

OMG. *Meta Object Facility Core Specification - version 2.5.1*. 2019. Disponível em: <https://www.omg.org/spec/MOF/2.5.1/PDF>. Acesso em 26/12/2020. Citado na página 56.

OMG. *The Real-Time Publish-Subscribe Protocol DDS Interoperability Wire Protocol Specification (DDSI-RTPS) - version 2.3*. [S.l.], 2019. Acesso em: 02/03/2021. Citado na página 45.

OMG. *What is DDS?* 2021. Disponível em: <https://www.dds-foundation.org/what-is-dds-3/>. Acesso em: 02/03/2021. Citado 5 vezes nas páginas 27, 43, 44, 97 e 98.

ORACLE. *What is an enterprise bean? The Java EE 5 Tutorial*. 2010. Disponível em: <https://docs.oracle.com/javaee/5/tutorial/doc/bnblt.html>. Acesso em 20/12/2020. Citado na página 50.

PACHECO, F. Energias renováveis: breves conceitos. *Conjuntura e Planejamento*, v. 149, p. 4–11, 2006. Citado na página 35.

PINHEIRO, N. *Microrredes: Conceitos e Atualidade*. 2015. Monografia. Universidade Federal do Maranhão. Citado na página 20.

RAMCHURN, S. D.; VYTELINGUM, P.; ROGERS, A.; JENNINGS, N. R. Putting the'smarts' into the smart grid: a grand challenge for artificial intelligence. *Communications of the ACM*, ACM New York, NY, USA, v. 55, n. 4, p. 86–97, 2012. Citado na página 27.

REHMANI, M. H.; DAVY, A.; JENNINGS, B.; ASSI, C. Software defined networks-based smart grid communication: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, IEEE, v. 21, n. 3, p. 2637–2670, 2019. Citado na página 27.

REHMANI, M. H.; REISSLEIN, M.; RACHEDI, A.; EROL-KANTARCI, M.; RADENKOVIC, M. Integrating renewable energy resources into the smart grid: Recent developments in information and communication technologies. *IEEE Transactions on Industrial Informatics*, IEEE, v. 14, n. 7, p. 2814–2825, 2018. Citado na página 21.

REKA, S. S.; DRAGICEVIC, T. Future effectual role of energy delivery: A comprehensive review of internet of things and smart grid. *Renewable and Sustainable Energy Reviews*, Elsevier, v. 91, p. 90–108, 2018. Citado na página 27.

RENTSCH, T. Object oriented programming. *ACM Sigplan Notices*, ACM New York, NY, USA, v. 17, n. 9, p. 51–57, 1982. Citado na página 46.

RITCHIE, D. M. The development of the c language. *ACM Sigplan Notices*, v. 28, n. 3, p. 201–208, 1993. Citado na página 46.

SCHMIDT, D. C. Model-driven engineering. *Computer-IEEE Computer Society-*, Citeseer, v. 39, n. 2, p. 25, 2006. Citado 4 vezes nas páginas 24, 47, 49 e 57.

SCHOFIELD, M.; COULTER, D.; JACOBS, M.; SATRAN, M. *COM+ - Component Services*. 2018. Disponível em: <https://docs.microsoft.com/en-us/windows/win32/cossdk/component-services-portal>. Acesso em 20/12/2020. Citado na página 50.

SHI, K.; BI, Y.; JIANG, L. Middleware-based implementation of smart micro-grid monitoring using data distribution service over ip networks. In: IEEE. *2014 49th International Universities Power Engineering Conference (UPEC)*. [S.l.], 2014. p. 1–5. Citado na página 95.

SIEGEL, J. *Object Management Group Model Driven Architecture (MDA) MDA Guide rev. 2.0*. 2014. Disponível em: <http://www.omg.org/cgi-bin/doc?ormsc/14-06-01>. Acesso em 11/12/2020. Citado 6 vezes nas páginas 24, 47, 48, 49, 52 e 86.

SOLEY, R. *Model Driven Architecture White Paper*. [S.l.], 2000. Acesso em 20/12/2020. Citado 3 vezes nas páginas 24, 50 e 51.

SOMMERVILLE, I. *Engenharia de Software*. PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>. Citado 2 vezes nas páginas 127 e 128.

SOUVENT, A.; KODEK, T.; SULJANOVIĆ, N. Cim-based integration in smart grids: Slovenian use cases. In: IEEE. *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*. [S.l.], 2019. p. 1–6. Citado 11 vezes nas páginas 9, 17, 24, 74, 75, 78, 79, 95, 96, 151 e 152.

STANTON, N. K.; GIRI, J. C.; BOSE, A. Energy Management. In: _____. [S.l.]: CRC Press, 2007. cap. 17, p. 289–298. Citado na página 22.

STEFANELLO, D. R. *Um Framework Baseado em MDE e Weaving Para Suporte ao Desenvolvimento de Sistemas de Software*. Dissertação (Mestrado) — Universidade Federal do Maranhão, 2017. Citado 7 vezes nas páginas 25, 28, 60, 61, 84, 85 e 101.

STRASSER, T.; ANDRÉN, F.; KATHAN, J.; CECATI, C.; BUCCELLA, C.; SIANO, P.; LEITAO, P.; ZHABELOVA, G.; VYATKIN, V.; VRBA, P. et al. A review of architectures and concepts for intelligence in future electric energy systems. *IEEE Transactions on Industrial Electronics*, IEEE, v. 62, n. 4, p. 2424–2438, 2014. Citado na página 21.

UJVAROSI, A. Evolution of scada systems. *Bulletin of the Transilvania University of Brasov. Engineering Sciences. Series I*, Transilvania University of Brasov, v. 9, n. 1, p. 63, 2016. Citado na página 36.

W3C. *XML Schema Definition Language - version 1.1*. 2012. Disponível em: <https://www.w3.org/TR/xmlschema11-1/>. Acesso em 21/03/2021. Citado na página 102.

WALLACE, C. *The semantics of the C++ programming language*. [S.l.]: Citeseer, 1993. Citado na página 46.

WAN, C.; ZHAO, J.; SONG, Y.; XU, Z.; LIN, J.; HU, Z. Photovoltaic and solar power forecasting for smart grid energy management. *CSEE Journal of Power and Energy Systems*, CSEE, v. 1, n. 4, p. 38–46, 2015. Citado na página 21.

WIDERGREN, S.; LEVINSON, A.; MATER, J.; DRUMMOND, R. Smart grid interoperability maturity model. In: IEEE. *IEEE PES General Meeting*. [S.l.], 2010. p. 1–6. Citado na página 24.

YU, X.; CECATI, C.; DILLON, T.; SIMOES, M. G. The new frontier of smart grids. *IEEE Industrial Electronics Magazine*, IEEE, v. 5, n. 3, p. 49–63, 2011. Citado 5 vezes nas páginas 27, 32, 33, 34 e 35.

A Código completo do *TopicHandler*

Este anexo contém o código completo do *TopicHandler* utilizado nas aplicações desenvolvidas. O código fonte das classes *DistributionGridTopicHandler* e *DistributionGrid TopicHandlerDataReaderListener* e da interface *DistributionGridEventHandler* são apresentados.

A.1 Código da classe *DistributionGridTopicHandler*

O código da classe *DistributionGridTopicHandler* é apresentado na Listagem A.1.

```
1
2  import DDS.*;
3  import OpenDDS.DCPS.*;
4  import DistributionGrid.*;
5  import org.omg.CORBA.StringSeqHolder;
6
7  public class DistributionGridTopicHandler {
8
9    private Topic topic;
10   private Publisher publisher;
11   private DataWriter dataWriter;
12   private GeographicalRegionDataWriter instanceDataWriter;
13   private Subscriber subscriber;
14   private DistributionGridTopicHandlerDataReaderListener listener
       ;
15   private DataReader dataReader;
16   private GeographicalRegion instance;
17   private int instance_handle;
18   private DomainParticipant domainParticipant;
19   private DomainParticipantFactory domainParticipantFactory;
20   private GeographicalRegionTypeSupportImpl typeSupport;
21
22   public void configureTopicHandler(GeographicalRegion _instance,
23       String topicName, int domainId, String[] args){
24
25     //Criacao da fabrica do participante
26     domainParticipantFactory = TheParticipantFactory
27         .WithArgs(new StringSeqHolder(args));
28
29     if (domainParticipantFactory == null) {
```

```
30        System.err.println("ERROR: Domain Participant Factory not
              found");
31        System.exit(1);
32      }
33
34      //Criacao do participante do dominio
35      this.domainParticipant = domainParticipantFactory
36          .create_participant(domainId,
37          PARTICIPANT_QOS_DEFAULT.get(),
38          null, DEFAULT_STATUS_MASK.value);
39
40      if (this.domainParticipant == null) {
41        System.err.println("ERROR: Domain Participant creation
              failed");
42        System.exit(1);
43      }
44
45      //Registrar o tipo
46      this.typeSupport = new GeographicalRegionTypeSupportImpl();
47      if(this.typeSupport
48        .register_type(this.domainParticipant, "")!=RETCODE_OK.
              value) {
49
50        System.err.println("ERROR: register_type failed");
51        System.exit(1);
52      }
53
54      //Criar o topico DDS
55      this.topic = this.domainParticipant.create_topic(topicName,
56          this.typeSupport.get_type_name(),
57          TOPIC_QOS_DEFAULT.get(), null, DEFAULT_STATUS_MASK.value)
                ;
58      if (this.topic == null) {
59                System.err.println("ERROR: Topic creation failed");
60                System.exit(1);
61      }
62
63      //Registrar a instancia
64      setInstance(_instance);
65
66    }
67
```

```
68   public void enablePublisher(){
69
70     //Criacao do Publisher
71     this.publisher = this.domainParticipant
72         .create_publisher(PUBLISHER_QOS_DEFAULT.get(),
73         null,
74         DEFAULT_STATUS_MASK.value);
75
76     if (this.publisher == null) {
77             System.err.println("ERROR: Publisher creation failed")
                ;
78             System.exit(1);
79     }
80
81     //Criacao do DataWriter
82     this.dataWriter = this.publisher.create_datawriter(this.topic
          ,
83       DATAWRITER_QOS_DEFAULT.get(),
84       null, DEFAULT_STATUS_MASK.value);
85     if (this.dataWriter == null) {
86         System.err.println("ERROR: DataWriter creation failed");
87         System.exit(1);
88     }
89
90     //Criacao do DataWriter especifico do tipo
91     instanceDataWriter = GeographicalRegionDataWriterHelper
92       .narrow(this.dataWriter);
93
94     //Registrando instancia
95           this.instance_handle = instanceDataWriter
96             .register_instance(instance);
97   }
98
99   public void enableSubscriber(DistributionGridEventHandler
        eventHandler){
100    //Criacao do Subscriber
101    this.subscriber = this.domainParticipant
102        .create_subscriber(SUBSCRIBER_QOS_DEFAULT.get(),
103        null,
104        DEFAULT_STATUS_MASK.value);
105
106    if (this.subscriber == null) {
```

```
107             System.err.println("ERROR: Subscriber creation failed"
                    );
108             System.exit(1);
109     }
110
111     //Criacao do listener
112           if(eventHandler==null){
113                 this.listener =
114                 new
                        DistributionGridTopicHandlerDataReaderListener(
                        instance);
115           }else{
116                 this.listener =
117                 new
                        DistributionGridTopicHandlerDataReaderListener(
                        instance,
118                      eventHandler);
119           }
120
121
122     //Criacao do DataReader
123     this.dataReader = this.subscriber
124       .create_datareader(this.topic,
125       DATAREADER_QOS_DEFAULT.get(),
126       this.listener, DEFAULT_STATUS_MASK.value);
127
128     if(this.dataReader == null) {
129         System.err.println("ERROR: DataReader creation failed");
130         System.exit(1);
131     }
132   }
133
134   private void setInstance(GeographicalRegion _instance){
135     this.instance = _instance;
136
137   }
138
139   public void publishInstance(){
140     int ret = RETCODE_TIMEOUT.value;
141
142     if(instance == null){
143       System.out.println("The instance is not set!");
```

```
144         return;
145       }
146
147       //Enviar dados para o DDS
148       ret = instanceDataWriter.write(instance, instance_handle);
149       if (ret != RETCODE_OK.value) {
150                   System.err.println("ERROR " + " write() returned
                        " + ret);
151                   System.exit(1);
152             }
153     }
154 }
```

Listing A.1 – Código completo da classe *DistributionTopicHandler*.

A.2 Código da classe *DistributionGridTopicHandler DataReaderListener*

O código completo da classe *DistributionGridTopicHandlerDataReaderListener* está disponível na listagem A.2.

```
1
2  import DDS.*;
3  import DistributionGrid.*;
4
5  public class DistributionGridTopicHandlerDataReaderListener
6    extends _DataReaderListenerLocalBase
7    {
8
9    private DistributionGridEventHandler eventHandler = null;
10   private GeographicalRegion instance = null;
11
12   //Construtor
13   public DistributionGridTopicHandlerDataReaderListener(
14     GeographicalRegion _instance) {
15     super();
16     instance = _instance;
17   }
18
19   //Construtor com o event Handler
20   public DistributionGridTopicHandlerDataReaderListener(
21         GeographicalRegion _instance,
```

```
22        DistributionGridEventHandler _eventHandler){
23      super();
24            instance = _instance;
25      eventHandler = _eventHandler;
26    }
27
28    @Override
29    public void on_data_available(DataReader reader) {
30
31      //Criacao do DataReader especifico do tipo
32      GeographicalRegionDataReader dataReader =
33        GeographicalRegionDataReaderHelper.narrow(reader);
34
35      if (dataReader == null) {
36        System.err.println("ERROR: read: narrow failed.");
37        System.exit(1);
38      }
39
40      GeographicalRegionHolder holder =
41        new GeographicalRegionHolder(instance);
42      SampleInfoHolder sih =
43        new SampleInfoHolder(new SampleInfo(0, 0, 0,
44        new DDS.Time_t(), 0, 0, 0, 0, 0, 0, 0, false, 0));
45
46      int status = dataReader.take_next_sample(holder, sih);
47
48      //Acessando os dados
49      if(status == RETCODE_OK.value){
50        if(sih.value.valid_data){
51
52        if(eventHandler!=null){
53                          //Contactando o event handler
54          this.eventHandler
55          .newDistributionGridData(holder.value);
56                    }
57
58
59        }
60        else if (sih.value.instance_state ==
61                          NOT_ALIVE_DISPOSED_INSTANCE_STATE.value)
                             {
62              System.out.println("instance is disposed");
```

```
63        }
64        else if (sih.value.instance_state ==
65                            NOT_ALIVE_NO_WRITERS_INSTANCE_STATE.value
                               ) {
66              System.out.println("instance is unregistered");
67        }
68        else {
69              System.out.println("DataReaderListenerImpl::" +
70                "on_data_available: " +
71                "ERROR: received unknown instance state " +
72                sih.value.instance_state);
73        }
74
75      }
76      else if (status == RETCODE_NO_DATA.value) {
77        //Acao para o caso de que nao ha dados disponiveis
78      }
79      else {
80      }
81    }
82  }
```

Listing A.2 – Código completo da classe *DistributionGridTopicHandlerDataReaderListener*.

A.3 Código da interface *DistributionGridEventHandler*

O código completo da interface *DistributionGridEventHandler* está disponível na Listagem A.3.

```
1  import DistributionGrid.*;
2
3  public interface DistributionGridEventHandler{
4    public void newDistributionGridData(GeographicalRegion data);
5
6  }
```

Listing A.3 – Código completo da interface *DistributionGridEventHandler*.