UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ISRAEL LIMA DIAS

TADEM: UM FRAMEWORK PARA TRANSFORMAÇÃO DE MODELOS COMPUTACIONAIS ENTRE FERRAMENTAS DE MODELAGEM

MOSSORÓ

2021

ISRAEL LIMA DIAS

TADEM: UM FRAMEWORK PARA TRANSFORMAÇÃO DE MODELOS COMPUTACIONAIS ENTRE FERRAMENTAS DE MODELAGEM

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação - associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Lenardo Chaves e Silva- UFERSA

Coorientador: Prof. Dr. Sebastião Emidio Alves Filho- UERN

MOSSORÓ

2021

ISRAEL LIMA DIAS

TADEM: UM FRAMEWORK PARA TRANSFORMAÇÃO DE MODELOS COMPUTACIONAIS ENTRE FERRAMENTAS DE MODELAGEM

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação - Associação ampla entre a Universidade do Estado do Rio Grande do Norte e a Universidade Federal Rural do Semi-Árido, para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: ___ / ___ / _____.

BANCA EXAMINADORA

Prof. Dr. Lenardo Chaves e Silva
Orientador e Presidente da Banca

Prof. Dr.
Examinador Externo -

Prof. Dr.
Examinador Interno - UFERSA

AGRADECIMENTOS

Agradeço à Deus, pois sinto que não há nada que possa ser feito sem sua benção.

Agradeço à minha família. Minha mãe Josilene, que sempre veio a mim com palavras de incentivo, inspiração e superação. Ao meu pai, Francisco Israel, que sempre me deu suporte e descontrações para seguir em frente, um ser humano em que busco me espelhar por sua dedicação e comprometimento.

Agradeço à minha namorada Lohanna, por estar comigo durante todo esse percurso, me ajudando não só psicologicamente, mas também tornando possível a conclusão desse trabalho.

Agradeço aos meus orientadores e amigos, professor Lenardo Silva e professor Sebastião Alves, pois não mediram esforços para ajudar no desenvolvimento deste trabalho, sempre pacientes, trazendo conselhos e incentivos.

Agradeço aos meus amigos conterrâneos de Amarante, que mesmo durante um período tão difícil, não me deixaram esquecer o quanto devemos lutar para alcançar nossos objetivos.

Agradeço aos amigos que fiz na UFERSA, onde encontrei apoio e companheirismo, junto com todo aprendizado que tivemos juntos.

Agradeço ao Programa de Pós-Graduação em Ciência da Computação das universidades UERN e UFERSA, e à CAPES, por terem disponibilizado estrutura e apoio financeiro durante a execução deste trabalho.

Agradeço às pessoas que Deus me apresentou em outras épocas da minha vida e que me ajudaram a percorrer o caminho para eu pudesse chegar até aqui.

“Eu posso não ter ido para onde eu pretendia ir, mas eu acho que acabei terminando onde eu pretendia estar.”

Douglas Adams

RESUMO

Construir sistemas que integram dispositivos médicos com o intuito de prover tomada de decisão automatizada, os denominados Sistemas Médicos Físico-Cibernéticos (SMFCs), demandam um intenso esforço em sua concepção (SILVA et al., 2014). Os SMFCs têm como características a tomada de decisão automática, segura e inteligente realizada pelo próprio sistema médico. No entanto, exige-se planejar, codificar e testar com bastante cautela todas as suas funcionalidades, pois em caso de falha do sistema o maior prejudicado é o paciente. A partir da perspectiva cibernética, os desenvolvedores de SMFC devem lidar com a heterogeneidade e a confiança no funcionamento dos dispositivos. Entretanto, devido às mudanças do estilo de desenvolvimento e das metodologias utilizadas, a construção de uma mesma solução para diferentes ferramentas torna-se massante, prolongando o tempo para se criar outro modelo ou mapear modelos diferentes, uma vez que diferentes ferramentas podem realizar tarefas equivalentes por meios distintos. A Engenharia Dirigida por Modelos, mais especificamente a Transformação de Modelos, torna mais eficiente o desenvolvimento de um *software*, buscando melhorar a manutenção e atualização, evitando erros e evitando a perca de consistência entre os diversos artefatos relacionados. Desta maneira, este trabalho apresenta o TADEM, um *framework* que contribui com um mapeamento aprimorado entre elementos de linguagem, um mecanismo para mapear de forma automática os componentes presentes nos modelos de Sistemas Físico-Cibernéticos (SFC). O uso desse *framework* permitiu aumentar a velocidade na concepção de CPS modelos no contexto clínico, uma vez que cada ferramenta de modelagem possui suas nuances que impossibilitam uma interação direta entre elas. Neste documento serão apresentados os detalhes sobre sua arquitetura e as tecnologias utilizadas, além de demonstrar a aplicação do TADEM em modelos clínicos criados no Ptolemy II® para os componentes com funcionalidades similares da ferramenta Simulink®.

Palavras-chave: Modelos, Transformação de Modelos, Engenharia Dirigida por Modelos, Mapeamento de Componentes, Sistemas (Médicos) Físico-Cibernéticos

ABSTRACT

Building systems that integrate medical devices in order to provide automated decision making, the so-called Medical Cyber-Physical Systems (MCPSs), demands an intense effort in their design (SILVA et al., 2014). MCPSs are characterized by automatic, safe and intelligent decision making performed by the medical system itself. However, it is necessary to plan, code, and test all their functionalities with great care, because in case of system failure, the patient will be the most affected. From the cybernetic perspective, MCPS developers must deal with heterogeneity and reliability in the operation of the devices. However, due to changes in development style and methodologies used, building the same solution for different tools becomes massive, prolonging the time to create another model or map different models, since different tools can perform equivalent tasks by different means. Model Driven Engineering, more specifically Model Transformation, makes software development more efficient, seeking to improve maintenance and updating, avoiding errors and avoiding the loss of consistency among the various related artifacts. In this way, this paper presents TADEM, a framework that contributes with an improved mapping between language elements, a mechanism for automatically mapping the components present in the models of Cyber-Physical Systems (CPS). The use of this framework allowed to increase the speed in designing CPS models in the clinical context, since each modeling tool has its nuances that make a direct interaction between them impossible. This paper will present details about its architecture and the technologies used, and demonstrate the application of TADEM in clinical models created in Ptolemy II® for components with similar functionalities of the Simulink® tool.

Keywords: Models, Model Transformation, Model Driven Engineering, Component Mapping, Physical-Cyber (Medical) Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AADL	*Architecture Analysis & Design Language*
DOM	Desenvolvimento Orientado a Modelos
GME	*Generic Modeling Environment*
JSON	*JavaScript Object Notation*
MDE	Engenharia Dirigida por Modelos
MDL	*Simulink Model Format*
MIC	*Model-Integrated Computing*
MoCs	*Model of Computations*
OWL	*Ontology Web Language*
PID	Proporcional Integral Derivativo
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
QC	Questão Conceitual
QGP	Questão Geral de Pesquisa
QP	Questão Prática
QT	Questão Tecnológica
RDF	*Resource Description Framework*
RSL	Revisão Sistemática da Literatura
SFC	Sistema Físico-Cibernético
SLX	*MathWorks Simulink Model Format*
SMFC	Sistema Médico Físico-Cibernético
Tcl	*Tool Command Language*
TEL	Tempo de Execução Lógico
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*
XPath	*XML Path Language*

SUMÁRIO

1 INTRODUÇÃO

A integração das tecnologias computacionais com processos físicos em meio ao ambiente no qual vivemos é possibilitada pelos componentes cibernéticos, onde a comunicação e integração entre esses meios precisam ser coerentes e controladas de maneira a regular o seu comportamento. Segundo Baheti e Gill (2011), o termo Sistema Físico-Cibernético (SFC), do inglês *Cyber-Physical System*, refere-se a "sistemas computacionais e físicos integrados que podem interagir com os seres humanos através de muitas modalidades, com capacidade de expandir as possibilidades do mundo físico através da computação, comunicação, e controle como um facilitador chave para desenvolvimentos tecnológicos".

Os sistemas físico-cibernéticos fazem cada vez mais parte do cotidiano das pessoas, indo de robôs de limpeza e sistemas inteligentes de iluminação a carros autônomos e espaçonaves capazes de levar uma tripulação à órbita terrestre. Por meio dessas tecnologias, tem-se o potencial de melhorar significativamente o bem-estar e a saúde das pessoas. Dessa forma, quando bem aplicadas ao contexto médico, pode-se chegar ao ponto de salvar vidas, como acontece com a utilização de marca-passos, bombas de insulina, próteses inteligentes, robôs de assistência pessoal, estendendo-se a uma gama de possibilidades.

1.1 Motivação

Há uma busca cada vez maior para se construir SFCs na área da Saúde com o objetivo de melhorar a qualidade de vida da população. Os sistemas computacionais que integram dispositivos médicos, com o intuito de prover tomada de decisão automatizada, são denominados Sistemas Médicos Físico-Cibernéticos (SMFCs), do inglês *Medical Cyber-Physical Systems* (SILVA *et al.*, 2014).

Os SMFCs tratam-se de sistemas de malha fechada, onde um conjunto de dispositivos mecânicos ou eletrônicos regula automaticamente uma variável de processo para um estado (ponto de ajuste desejado) sem interação humana. Por serem sistemas críticos, onde falhas podem significar perigos à vida humana, há a necessidade de um esforço árduo em suas etapas de projeto, desenvolvimento e validação, requerendo alta exatidão para garantir confiabilidade e robustez.

Uma das principais técnicas utilizadas para o projeto de sistemas críticos é a construção de modelos (TUAN *et al.*, 2010). Para que esses modelos sejam validados,

especialmente quando tratam-se de modelos de simulação, é importante testá-los em diferentes ferramentas de modelagem. A motivação deste trabalho parte do fato de que as ferramentas de modelagem existentes são muito especializadas, o que dificulta a transformação de um modelo construído em uma ferramenta para outra. Assim, com este trabalho, busca-se facilitar o trabalho de projetistas de SMFCs com uma solução para a transformação (semi-)automática de modelos entre ferramentas distintas.

1.2 Justificativa

O desenvolvimento de modelos sempre requer casos de teste cuidadosamente projetados para cobrir todos os caminhos de execução possíveis, mas é difícil garantir que o projeto do caso de teste cubra completamente todos os caminhos. Normalmente, não é possível saber se todos os erros existentes no sistema foram detectados após o teste. Assim, outras técnicas, como a verificação prévia de modelos, podem ser utilizadas em conjunto para melhorar este processo. A verificação de modelos é uma técnica de verificação formal que permite explorar todo o espaço de estado do sistema a fim de encontrar o defeito ou falha (TUAN *et al.*, 2010).

É de crucial importância proporcionar aos projetistas e desenvolvedores de SMFCs, a viabilidade de realizar testes em múltiplas ferramentas para garantir a efetividade desses sistemas, uma vez que pacientes podem ter comportamentos distintos e não completamente previsíveis. No entanto, o trabalho de refazer o modelo em outra ferramenta leva tempo e pode gerar diferenças conceituais. Assim, uma solução para a transformação (semi-)automática de modelos pode facilitar e acelerar os testes de um modelo.

Devido ao tamanho e complexidade dos SMFCs serem maiores em relação aos sistemas médicos tradicionais, a viabilidade a longo prazo desses modelos exige enfrentar desafios através do desenvolvimento de um novo projeto, composição, verificação e técnicas de validação (LEE *et al.*, 2011). Para minimizar o esforço no desenvolvimento e no processo de validação, dispensando de início a utilização de pacientes reais com o intuito de identificar comportamentos indesejados com base no modelo computacional do sistema, é possível explorar novas oportunidades e soluções relevantes para o desenvolvimento de SMFCs, em geral, sistemas embarcados e SFCs.

1.3 Problemática

Devido às vantagens das ferramentas de simulação, com relação ao nível de complexidade para projeto e desenvolvimento, diversos pesquisadores têm adotado a abordagem de co-simulação, onde os diferentes subsistemas que formam um problema acoplado são modelados e simulados de forma distribuída (ZHAO *et al.*, 2014). A modelagem desses sistemas distribuídos, baseada em componentes que estariam se comunicando e coordenando ações entre si através de computadores interligados em rede, leva em conta a concorrência de componentes, ausência de um relógio global e falhas de componentes independentes (COULOURIS *et al.*, 2013).

Um dos processos para a construção de um SMFC é colocar a modelagem no centro do processo de desenvolvimento do software, que se beneficia de meios gráficos para descrevê-lo. Assim, por meio de simulações e diagrama de fluxo, o comportamento de um sistema pode ser concebido de acordo com a entrada de dados, processamento e consequentemente uma saída. Entretanto, mesmo sendo disponibilizadas diversas ferramentas de construção e simulação de modelos, há um problema com relação à compatibilidade entre elas por utilizarem meios particulares para elaborar o componente, ou o conjunto de componentes, que se fazem necessários para a construção de um sistema específico.

Alguns tipos de componentes e interações entre eles podem ser mapeados diretamente, por se tratarem de regras matemáticas e computacionais genéricas, como cálculos básicos e atribuições de variáveis. Com alguma adaptação, há componentes que podem ser relacionados entre si. Entretanto, há casos em que esses componentes são incompatíveis ou simplesmente não existem mutuamente. Dessa forma, os sistemas modelados podem apresentar diferenças em sua concepção, como pode ser verificado nos trabalhos de:

- Derler *et al.* (2008): que descreve como a abstração de Tempo de Execução Lógico (TEL) permite a modelagem independente da plataforma de acordo com o comportamento de temporização do software, com o intuito de comparar e destacar as diferenças e semelhanças da simulação de TEL entre as ferramentas PtolemyII® e Simulink®;
- SILVA *et al.* (2015): que utilizou uma arquitetura baseada em modelos para validação de SMFC, com o objetivo de promover o reúso e a produtividade, sendo seu potencial avaliado com a aplicação a três diferentes cenários clínicos;
- Melo *et al.* (2015): descreve a utilização dos ambientes PtolemyII® e Simulink®

como ferramentas de aprendizagem na implementação de modelos de simulação, para que sejam discutidas e avaliadas por estudantes de engenharia, representando o sistema de controle *multi-loop* descentralizado, usado para controlar processo em sistemas que precisam de mais de uma malha de controle dentro de um único sistema.

A fim de garantir a efetividade de um mesmo sistema mapeado entre duas ferramentas de modelagem e simulação distintas e obter resultados semelhantes, levando em conta a particularidade de cada ferramenta, primeiro é importante entender o funcionamento isolado em cada uma delas. Para isso, as ferramentas PtolemyII®[1] e Simulink®[2] foram escolhidas para este estudo.

1.4 Objetivos

Diante do exposto, o principal objetivo deste trabalho é definir um mecanismo para mapear ou transformar componentes entre diferentes ferramentas de modelagem computacional mediante a aplicação de um conjunto de regras de transformação (semi-)automática de modelos. Com isso, busca-se melhorar a eficiência na construção de um sistema médico físico-cibernético e garantir a eficácia e a qualidade no projeto, minimizando erros e aumentando a segurança do seu funcionamento. Assim, para empresas e desenvolvedores de sistemas que utilizam a modelagem como a base do seu desenvolvimento, manutenção e evolução, é dada a possibilidade de minimizar erros. Dessa forma, a consistência dos sistemas pode ser mantida, de modo a reduzir o esforço de desenvolvimento e preservar a similaridade entre modelos, código e documentação.

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- definir um mecanismo geral para transformar, de forma automática ou semi-automática, os componentes presentes nos modelos de simulação de cenários clínicos entre diferentes ferramentas de modelagem;
- garantir que o mecanismo proposto para o mapeamento (semi-)automático dos componentes dos modelos de cenários clínicos de simulação seja geral o suficiente para ser utilizado como solução única para diversos cenários clínicos;
- validar empiricamente a solução, demonstrando que os modelos de cenários

[1] Disponível em: ptolemy.berkeley.edu/ptolemyII/index.html
[2] Disponível em: mathworks.com/products/simulink.html

clínicos de simulação gerados a partir do mecanismo de mapeamento proposto, e com base nas ferramentas de modelagem em estudo, são compatíveis e mantêm a conformidade comportamental durante as simulações de cada cenário clínico em particular.

1.5 Questões de Pesquisa

As questões de pesquisa deste trabalho foram definidas segundo a metodologia *Design Science*, que é voltada para o projeto e a investigação de artefatos em um contexto (WIERINGA, 2014). O foco deste trabalho está na especificação e definição de um *framework* para o mapeamento de componentes entre diferentes ferramentas de modelagem computacional, mais especificamente, aquelas utilizadas para construir modelos de simulação de cenários clínicos para validações e projetos de sistemas aplicados à Saúde. Assim, a Questão Geral de Pesquisa (QGP) definida é a seguinte:

Como realizar o mapeamento de modelos de cenários clínicos entre diferentes ferramentas de modelagem computacional para os SMFCs?

De acordo com Wieringa (2009), na *Design Science* a QGP pode ser decomposta em grupos de Questões Conceituais (QCs), Questões Tecnológicas (QTs) e Questões Práticas (QPs), constituindo o conjunto-problema da pesquisa. Dessa forma, os grupos de questões mais específicas são os seguintes:

- QC: Qual a abordagem que melhor se adéqua aos paradigmas e às ferramentas de modelagem utilizadas em questão?
  - QC1: Qual o foco das ferramentas de modelagem?
  - QC2: Quais as especificações destas ferramentas utilizadas?
- QT: Quais as abordagens ou os mecanismos que possuem potencial de serem utilizados para mapear, de forma automática ou semiautomática, os componentes presentes nos modelos?
  - QT1: Como os modelos são estruturados no arquivo gerado em cada umas das ferramentas?
- QP: Como garantir que o mecanismo proposto para o mapeamento seja geral o suficiente para ser utilizado como solução para vários cenários clínicos utilizados como base para sua concepção?
  - QP1: É possível utilizar a solução de forma generalista, independentes do

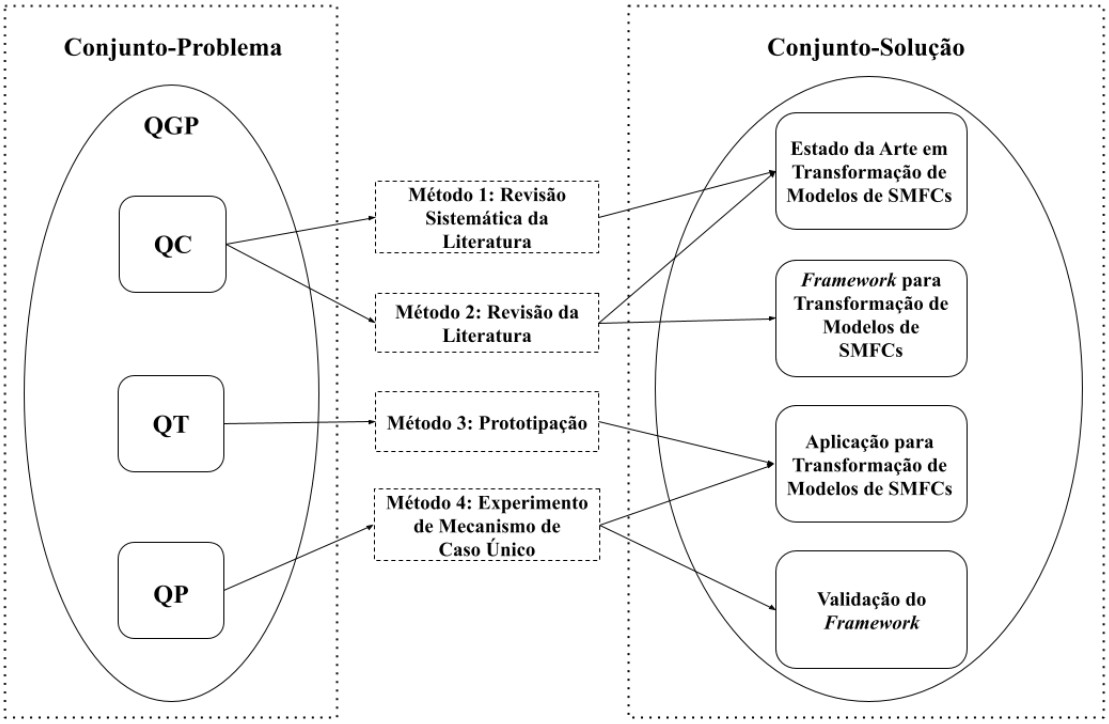
cenário clínico modelado nas ferramentas de modelagem computacional empregadas?

1.6 Metodologia

Neste trabalho, foram definidos um conjunto-problema (questões de pesquisa) e um conjunto-solução a fim de alcançar os objetivos definidos na **Seção 1.4**. Para chegar ao conjunto-solução, foram definidos alguns métodos, conforme mostrado na **Figura 1** e listado a seguir:

1. **Revisão Sistemática da Literatura (RSL)**: levantamento do estado da arte por meio de um processo sistemático e formal. O objetivo com a aplicação deste método foi encontrar abordagens de Transformação de Modelos específicas entre diferentes ferramentas de modelagem e identificar as principais soluções que pudessem ser utilizadas no contexto do trabalho;
2. **Revisão da Literatura**: realização de pesquisas por artigos e/ou relatórios técnicos, em motores de busca *Web*, com o objetivo de obter as principais informações que norteiam o estudo da Engenharia Dirigida por Modelos com foco em Transformação de Modelos;
3. **Prototipação**: com a prototipação foi pretendido especificar um *framework* utilizável que permitisse auxiliar no desenvolvimento de ferramentas para transformação automática de modelos. A partir do *framework* foi desenvolvida uma aplicação para a transformação de modelos de SMFC entre as ferramentas PtolemyII® e Simulink® para fins de validação;
4. **Experimento de Mecanismo de Caso Único**: o objetivo com este método foi testar, descrever e explicar o comportamento de causa-efeito do objeto de estudo (WIERINGA, 2014). Com o intuito de validar o *framework*, foi aplicado um modelo gerado na ferramenta PtolemyII® ao protótipo desenvolvido com o intuito de avaliar cenários de testes dos componentes lidos para observar as respostas com relação à leitura e execução pela ferramenta Simulink®.

Figura 1 – Metodologia de Pesquisa



Fonte: Autoria própria (2021).

1.7 Organização do Trabalho

Após este capítulo introdutório, dar-se-á continuidade à estruturação do trabalho com o **Capítulo 2**, no qual é apresentado o Referencial Teórico, onde os conhecimentos básicos para o processo de pesquisa são explorados, junto aos conceitos necessários para compreender e executar este trabalho. Para encontrar os trabalhos relacionados, uma pesquisa aprofundada foi realizada através de uma RSL, que é mostrada no **Capítulo 3**. Além da RSL, também é apresentados trabalhos complementares, onde foi possível identificar pesquisas realizadas por diversos autores que se equiparavam à deste trabalho. Após a análise de todo o conteúdo teórico, um *framework* pôde ser estabelecido e sua abordagem é concebida no **Capítulo 4**. No **Capítulo 5** é mostrado como se deu o desenvolvimento do *framework* na prática, nomeado pelos autores como TADEM, um acrônimo para "Transformação Automática DE Modelos". Ainda no **Capítulo 5**, com o propósito de validação do *framework*, um experimento de mecanismo de caso único é descrito. Por fim, no **Capítulo 6** são discutidas as conclusões do trabalho, explorando suas contribuições, descrevendo as limitações encontradas e as possibilidades para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo é apresentada a base teórica necessária para o entendimento das tecnologias utilizadas no decorrer da pesquisa. Conceitos que fundamentam a solução são expostos, sendo eles: sistemas médicos físico-cibernéticos, desenvolvimento baseado em modelos, as ferramentas Ptolemy II® e Simulink®, utilizadas para criação de modelos, engenharia dirigida por modelos e transformação de modelos.

2.1 Sistemas Físico-Cibernéticos

Segundo Lee (2015), um Sistema Físico-Cibernético (SFC) é representado por dois componentes cruciais, sendo o primeiro uma conectividade que assegura a comunicação de dados em tempo real do mundo físico, ao mesmo tempo em que há um *feedback* de informações do sistema cibernético, e o segundo, um gerenciamento otimizado de dados com uma análise computacional que constrói o sistema cibernético.

Poovendran *et al.* (2011) divide os estudos sobre SFC em dois campos: os fundamentos teóricos e as aplicações, onde seu uso tem o objetivo de facilitar e aumentar a implementação de sistemas de grande escala, com a possibilidade de melhorar a autonomia, funcionalidade, adaptabilidade, confiabilidade, usabilidade e segurança desses sistemas.

A academia, a indústria e até mesmo vários governos têm se interessado pelos SFCs por conta do potencial impacto significativo na economia e na sociedade (XU *et al.*, 2014) e (GÜRDÜR *et al.*, 2016).

O termo *Cyber-Physical System* tem sido usado desde os anos 70 com o surgimento dos microprocessadores, mas foi apenas em 2006 que esse termo foi atualizado e utilizado para descrever sistemas que comunicam o mundo físico com o digital (LEE, 2015), (WOLF, 2009) e (GILL, 2006).

2.2 Sistemas Médicos Físico-Cibernéticos

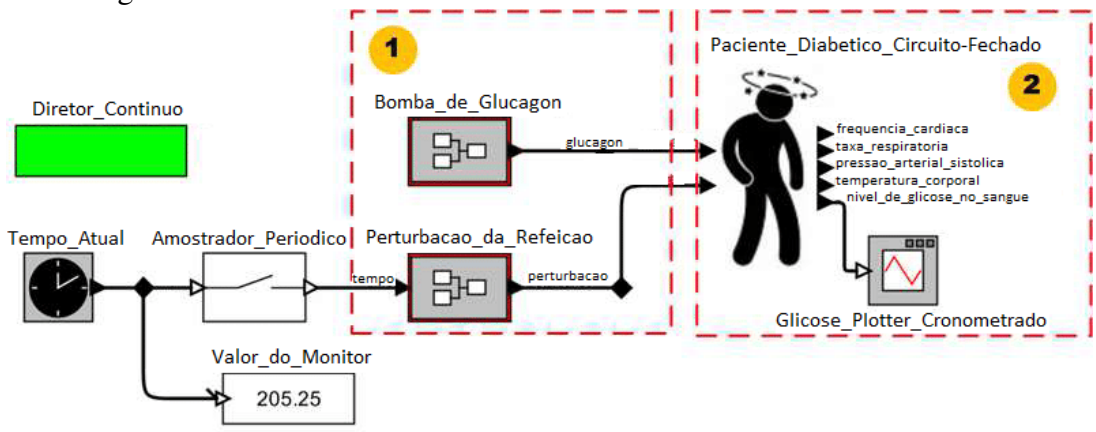
Sistemas Médicos Físico-Cibernéticos, são sistemas interconectados e inteligentes, críticos de segurança, voltados aos dispositivos médicos. Alguns cenários clínicos mais tradicionais podem ser vistos como sistemas de malha fechada, onde os cuidadores são os controladores, os dispositivos médicos são os sensores e atuadores, enquanto os pacientes são vistos como as “plantas”, se referindo a uma função de transferência que indica a

relação entre um sinal de entrada e o sinal de saída de um sistema (LEE *et al.*, 2011).

O suporte à decisão é automatizado, tornando possível que os dados recebidos dos dispositivos de monitoramento sejam analisados por um elemento controlador para estimar o estado de saúde do paciente e prontamente iniciar um tratamento (SILVA *et al.*, 2015).

Na **Figura 2** é exibido de maneira conceitual um modelo de SMFC usado como demonstração de um cenário clínico envolvendo um modelo de paciente diabético, onde SILVA *et al.* (2015) descreveram que o comportamento do modelo é determinado por equações diferenciais que representam a forma como é feito o controle da absorção, distribuição e eliminação da insulina no organismo de um paciente. Observando o modelo geral, os modelos de componentes funcionais **(1)** são usados para a ajustar os respectivos valores para cada sinal vital representado no modelo de paciente **(2)**, utilizando a glicose como variável preditora. Foi adotada a estratégia de que um controlador Proporcional Integral Derivativo (PID) é projetado para o propósito de controlar o nível de glicose sanguínea, sendo este embarcado no modelo de paciente (intrusiva), como um sistema em malha-fechada.

Figura 2 – Modelo de SMFC usando um Modelo de Paciente Diabético.



Fonte: SILVA *et al.* (2015).

Devido ao aumento do tamanho e complexidade dos SMFCs por conta das demandas que surgem, novas técnicas e tecnologias voltadas ao projeto e desenvolvimento desses sistemas tendem a ser criadas, buscando sempre atender aos requisitos de segurança e eficácia, verificação e validação (LEE; SOKOLSKY, 2010).

2.3 Desenvolvimento Baseado em Modelos

Os modelos de *software* são componentes cruciais para uma simulação, pois com eles é possível executar testes separadamente para cada componente de um sistema, assim como um todo, sem a necessidade de investir um custo alto para o teste "em campo" desse sistema (KLEE; ALLEN, 2018). Um "modelo", de maneira geral, é uma entidade conceitual ou física que se assemelha, descreve, prevê ou veicula informações sobre o comportamento de algum processo ou sistema com o benefício de poder explorar o comportamento desse sistema de uma forma econômica e segura.

O Desenvolvimento Orientado a Modelos (DOM), do inglês *Model Driven Development*, considera, dentro do desenvolvimento de software, a modelagem do problema mais importante, que é o desenvolvimento do código fonte. Com isto, a atenção fica mais concentrada em construir uma aplicação final que funcione de acordo com os requisitos especificados no modelo do sistema.

Ao invés de exigir que os desenvolvedores expliquem todos os detalhes da implementação de um sistema usando uma linguagem de programação, ele permite que eles modelem qual funcionalidade é necessária e que arquitetura geral o sistema deve ter (ATKINSON; KUHNE, 2003). Ao aumentar os níveis de abstração, o DOM visa automatizar muitas das tarefas de programação complexas como fornecer suporte para persistência, interoperabilidade e distribuição do sistema (ATKINSON; KUHNE, 2003).

No entanto, não existe uma definição universalmente aceita de exatamente o que é DOM e qual o suporte para isso, existindo várias maneiras para sua concepção (PASTOR *et al.*, 2008). Assim que os desenvolvedores dominam uma nova tecnologia, outra aparece para substituí-la. Para aumentar a vida útil dos artefatos de software principais, deve-se protegê-los contra alterações no nível da plataforma, ou seja, deve-se automatizar o processo de obtenção de artefatos de software específicos da plataforma, aplicando mapeamentos predefinidos (ATKINSON; KUHNE, 2003).

Com o modelo já projetado e compreendido, há uma tentativa da engenharia dirigida por modelos de utilizá-lo de forma automatizada ou semi-automatizada para a construção em outras ferramentas seja por motivos de testes ou limitações da ferramenta primária. Assim, é possível o desenvolvimento de um mesmo sistema em outra ferramenta sem a necessidade de se construir todo o sistema do início. Isto possibilita realizar um *upgrade* (ou *downgrade*) de acordo com as características de cada ferramenta.

2.4 Ferramentas de Modelagem

Neste trabalho, a ferramenta Ptolemy II® foi utilizada para criar os modelos de simulação referentes aos cenários clínicos, utilizados como base para definir o mecanismo (semi-)automático de mapeamento de componentes para o Simulink®. Ambos os ambientes de modelagem que fizeram parte do estudo se caracterizam como um conjunto de ferramentas disponíveis, utilizadas para simular sistemas de controle e também para gerar códigos.

Ptolemy II®

Dentre várias ferramentas de modelagem computacional, neste trabalho é abordado o Ptolemy II®, uma ferramenta de código aberto que utiliza do conceito de projeto orientado a atores, onde componentes de software são executados simultaneamente e se comunicam por meio de mensagens enviadas através de portas interconectadas, com a finalidade de construir sistemas informatizados experimentais (XIONG *et al.*, 2005).

Eker *et al.* (2003) descreveram o Ptolemy II® como um ambiente de desenvolvimento que possibilita a modelagem, simulação e projeção de grandes sistemas simultâneos em tempo real, objetivando um alto nível de confiança e comportamento, de maneira que a própria ferramenta dê suporte para combinar uma grande variedade de sistemas computacionais permitindo uma rede de alinhamento hierárquico de modelos.

Ptolemy II® tem seu ambiente de simulação que serve como *playground* para experimentar diferentes modelos de computação e sua combinação em modelos heterogêneos, com seus arquivos definidos em *eXtensible Markup Language* (XML).

Simulink®

Outra ferramenta utilizada no estudo foi o Simulink®, uma extensão do Matlab® que permite a construção de modelos computacionais dinâmicos utilizando notação de diagrama de blocos. A ferramenta possibilita uma fácil modelagem de sistemas não lineares complexos, contando com componentes de uso contínuo e de tempo discreto. Com o Simulink® é possível demonstrar visualmente, através de animações gráficas, o comportamento da simulação, acrescentando significativamente o entendimento do comportamento do sistema (DABNEY; HARMAN, 2004).

A ferramenta permite uma rapidez e precisão no desenvolvimento de modelos de sistemas dinâmicos, simplificando a modelagem de sistemas não-lineares complexos (DABNEY; HARMAN, 2004). O formato único de texto *Simulink Model Format* (MDL) já foi o padrão da ferramenta, até ser atualizado para o formato de pacotes compactados *MathWorks Simulink Model Format* (SLX) em 2012.

2.5 Transformação de Modelos

A transformação de modelos é um importante quesito dentro da abordagem de Engenharia Dirigida por Modelos (MDE), do inglês *Model Driven Engineering*, que visa tornar mais eficiente o processo de desenvolvimento de software, incluindo suas atualizações e manutenção. O conceito de modelo dentro da Engenharia Dirigida por Modelos vai além de um simples esboço de um projeto para que vários desenvolvedores possam interconectar suas ideias, seu ponto central é prover artefatos primários que possam guiar todo o processo de desenvolvimento (JOUAULT *et al.*, 2008).

A literatura vem tratando os termos modelo e meta-modelo há algum tempo, caracterizando um modelo como a representação de um sistema, descrevendo as características do mesmo e provendo algum conhecimento acerca do mesmo. A base conceitual de um modelo é trabalhada como um meta-modelo, sendo considerados uma definição da sintaxe abstrata das linguagens de modelagem (JOUAULT; BÉZIVIN, 2006).

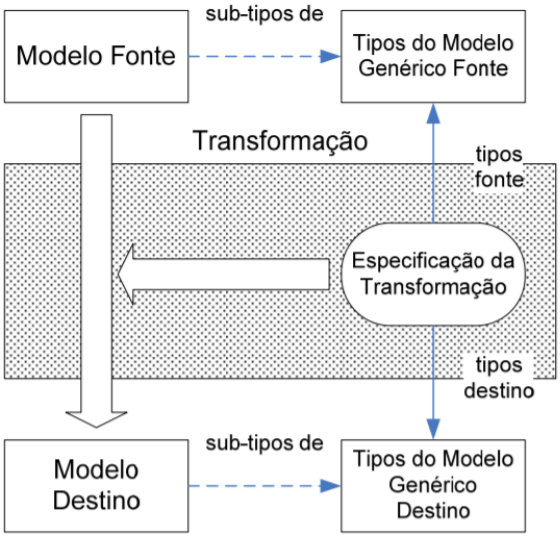
As transformações possibilitam o mapeamento e a transformação de modelos para outros modelos. Estas transformações podem ser de um outro modelo (modelo-modelo) ou de um modelo específico de plataforma, artefatos da plataforma especificada (modelo-plataforma). Contudo, as transformações resultantes em um código-fonte podem ser consideradas modelo-modelo desde que a linguagem de programação tenha um meta-modelo (CZARNECKI; HELSEN, 2003).

Para uma transformação ser efetivada, deve-se definir quais elementos serão transformados e qual o mapeamento entre os modelos. A partir dos elementos do modelo fonte, o mapeamento indicará quais elementos serão criados no modelo destino (TENÓRIO, 2004).

Com a transformação de meta-modelos, tem-se a definição dos meta-modelos referentes aos modelos fonte e destino, e a transformação dá-se com base em elementos dos meta-modelos. Para a transformação de modelos, os elementos dos respectivos

modelos fonte e destino são sub-tipos de modelos genéricos existentes e a transformação é especificada com base nos tipos dos modelos genéricos, como pode ser observado na **Figura 3**.

Figura 3 – Transformação de Modelos.



Fonte: Tenório (2004).

Para a transformação modelo-plataforma, o meta-código contendo as informações de um modelo fonte serve como base para que, a partir destas informações, seja selecionado ou criado um novo código-fonte, servindo de entrada para um *template* onde é dada a representação de um modelo.

2.5.1 Modelo Específico de Plataforma

Um Modelo Específico de Plataforma, do inglês *Platform Specific Model* (PSM), é um modelo genérico de uma tecnologia ou sistema de organização que é intrinsecamente dependente de uma tecnologia específica. Modelos específicos de plataforma têm sido indispensáveis na implantação de sistemas tecnicamente complexos (XIAO; FAN, 2012). Isso se deve ao fato de que qualquer informação, dado, conhecimento e funcionalidade devem ser processados e comunicados de maneira consistente com as tecnologias específicas que estão sendo usadas. Consequentemente, um modelo dependente de plataforma fornece soluções para todos os tipos de infraestruturas, principalmente para SFCs. Por fim, um PSM é um modelo que representa o sistema em relação às escolhas implementadas em uma determinada plataforma (BéVAN *et al.*, 2012).

2.5.2 Modelo Independente de Plataforma

Um Modelo Independente de Plataforma, do inglês *Platform Independent Model* (PIM), também chamado de modelo genérico na Engenharia de Software, é um modelo simples e reduzido de um sistema de software ou aplicativo específico que não depende de nenhuma plataforma tecnológica específica que representa o sistema para apoiar os processos (KHERRAF *et al.*, 2008).

Um conceito importante de um modelo independente é que este modelo em si não deve depender de nenhum componente do sistema de origem. Em vez disso, todos os dados e funcionalidades necessários devem ser obtidos de uma fonte externa.

2.5.3 Considerações Finais do Capítulo

Neste capítulo é apresentada a base teórica necessária para o entendimento das tecnologias utilizadas no decorrer da pesquisa. Conceitos que fundamentam a solução são expostos, sendo eles: sistemas médicos físico-cibernéticos, desenvolvimento baseado em modelos, as ferramentas Ptolemy II® e Simulink®, utilizadas para criação de modelos,engenharia dirigida por modelos e transformação de modelos. Com isso, é possível dar continuidade a pesquisa, aprofundando o tema através da RSL e trabalhos complementares.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma Revisão Sistemática da Literatura - RSL, realizada a fim de identificar o estado da arte na temática proposta, avaliar e interpretar pesquisas relevantes para o conjunto problema deste trabalho de forma imparcial. Também são apresentados trabalhos complementares que foram consultados para auxiliar na compreensão do tema estudado e ajudar a alcançar os objetivos da pesquisa. Os trabalhos abordados demonstram mecanismos para a transformação de modelos e estão disponíveis na literatura.

3.1 Revisão Sistemática da Literatura

Nesta seção é apresentado o protocolo e a execução de uma RSL que buscou trabalhos relacionados ao tópico de transformação de modelos relacionados a sistemas (médicos) físicos cibernéticos.

3.1.1 Definição do Objetivo e das Questões de Pesquisa

O objetivo desta RSL foi identificar e analisar soluções computacionais desenvolvidas e disponíveis na literatura que possibilitam a transformação de modelos entre diferentes ferramentas de modelagem computacional, no contexto de SMFC. Mais especificamente, o objetivo está relacionado à avaliação das abordagens, dos métodos, dos algoritmos, das ferramentas e das formas de validação destas soluções que auxiliam no mapeamento automático ou semiautomático de componentes de modelos de cenários clínicos de simulação.

As soluções buscadas devem possibilitar a transformação de modelos criados a partir de uma ferramenta de modelagem para uma segunda ferramenta distinta. Dessa forma, de maneira geral, podem ser utilizados como solução única em diferentes cenários. Considerando o que foi descrito, para alcançar o objetivo da RSL, as três questões de pesquisa definidas, e apresentadas a seguir, devem ser respondidas:

Questão de Pesquisa 1: Quais as soluções computacionais disponíveis na literatura que possibilitam o mapeamento de componentes entre diferentes ferramentas de modelagem computacional?

Questão de Pesquisa 2: Qual a abordagem que melhor se adequa aos paradigmas e às ferramentas de modelagem Ptolemy II e Simulink sob estudo, utilizadas para construir

os modelos de simulação de cenários clínicos no contexto da Saúde?

Questão de Pesquisa 3: Quais formas de validação foram empregadas para o propósito de analisar a eficácia dessas soluções de transformação de modelos computacionais entre diferentes ferramentas?

3.1.2 Estratégia de Busca e Seleção dos Estudos

Nesta seção é apresentado o protocolo que foi definido para a busca e a seleção dos estudos que respondam às questões de pesquisa delimitadas. A busca foi realizada em bibliotecas digitais relevantes na área de Ciência da Computação com a restrição de trabalhos publicados a partir de 2009, com texto completo do estudo disponível. As fontes de busca escolhidas são apresentadas no **Quadro 1**.

Quadro 1 – Fontes de Busca

Nome da Base	Link de Acesso
IEEE Xplore Digital Library	ieeexplore.ieee.org
ACM Digital Library	dl.acm.org
Science Direct	sciencedirect.com
Scopus	scopus.com
Web Of Science	webofknowledge.com

Fonte: Autoria própria (2019).

3.1.2.1 Elaboração da String de Busca

Algumas palavras-chave foram utilizadas para identificar trabalhos durante as buscas nas fontes de buscas, sendo algumas relevantes testadas previamente para auxiliar na identificação de sinônimos, a fim de definir a *string* de busca. Na busca prévia, tanto nas fontes definidas para a pesquisa, como em fontes secundárias, as palavras-chave do trabalho foram submetidas em português. No entanto, não foram encontrados trabalhos para os termos. Por este motivo, o idioma de pesquisa utilizado para a busca nas principais fontes foi o inglês. Devido às buscas com palavras mais específicas não retornarem trabalhos razoáveis acerca da pesquisa, os termos foram adaptados para abranger o conteúdo de maneira mais geral. As seguintes palavras-chave, apresentadas no **Quadro 2**, foram utilizadas:

Para obter os resultados esperados, foi definida a seguinte *string*: "***("model transformation") AND ("medical cyber-physical systems" OR "cyber-physical systems")***".

Quadro 2 – Palavras-chave para a *string*

Palavra-chave	Sinônimos
model transformation	-
medical cyber-physical systems	*cyber-physical systems*

Fonte: Autoria própria (2019).

Uma vez que os motores das fontes de busca têm características diferentes, a *string* teve de ser adaptada de acordo com a respectiva entrada de cada biblioteca.

3.1.2.2 Especificação do Escopo

Além do objetivo e das questões de pesquisa a serem respondidas, também foi especificado o seguinte escopo, que auxiliou na definição dos critérios analisados na seleção dos estudos:

- **Intervenção:** soluções computacionais voltadas ao mapeamento de componentes de modelos computacionais;
- **População:** estudos científicos que abordem soluções de transformação de modelos entre diferentes ferramentas;
- **Resultados:** identificar métodos, técnicas ou ferramentas que automatizem ou simplifiquem a transformação de modelos, bem como suas formas de validação;
- **Controle:** estudos considerados fundamentais para alcançar os objetivos da RSL.

3.1.2.3 Critérios de Inclusão

Para esta pesquisa, foram estabelecidos três critérios de inclusão que permitiram delimitar quais estudos atendem aos resultados esperados com a revisão. Os critérios definidos são apresentados a seguir:

1. estudos da área de Informática/Computação que apresentem mapeamento de componentes entre diferentes ferramentas de modelagem;
2. estudos que apresentem tecnologias apoiadas em quaisquer abordagens que se adéquem aos paradigmas e às ferramentas de modelagem Ptolemy II e Simulink;
3. estudos que relatem a(s) metodologia(s) de validação utilizada(s) para avaliar a eficácia dessas soluções de transformação de modelos computacionais entre diferentes ferramentas.

3.1.2.4 Critérios de Exclusão

A fim de remover os estudos que não apresentam os resultados esperados, durante o processo de seleção, também foram definidos os seguintes critérios de exclusão:

1. estudos da área de Informática/Computação que não contenham os interesses da pesquisa;
2. estudos publicados anteriormente ao ano de 2009;
3. estudos duplicados;
4. estudos que não estejam em português ou em inglês;
5. estudos que não são artigos completos (apresentações em slides, resumos expandidos ou pôsteres).

3.1.2.5 Critérios de Seleção Preliminar

Para a seleção preliminar, foi conferida a elegibilidade dos estudos conforme os quesitos a seguir:

- Estudos cujas as palavras-chave da *string* de busca estejam presentes (parcialmente ou totalmente) em seu título, resumo, palavras-chave e/ou termos indexados;
- Estudos que relatem características, forneçam algum tipo de revisão, formalismo, abordagem ou solução para transformação de modelos.

3.1.3 Extração de Dados dos Estudos

Para a extração dos dados a partir dos trabalhos finais obtidos na revisão foi definido o formulário de questões listado a seguir:

01. Qual a contribuição principal do trabalho?
    - Arquitetura (AR);
    - Abordagem (AB);
    - Ferramenta (FE);
    - Outras (OU).
02. Quais ferramentas de apoio foram utilizadas no trabalho?
03. Qual o nível de automação?
    - Automático (A);
    - Semiautomático (S);

Nenhum (N).

04. A contribuição do trabalho se adequa às ferramentas Ptolemy II e Simulink?

Sim;

Não.

3.1.3.1 Critérios de Qualidade dos Estudos

Para medir e determinar um *ranking* baseado no quanto cada um dos estudos finais, definidos na fase de extração, se adequam ao objetivo desta pesquisa, foram especificados os seguintes critérios de qualidade:

1. trabalhos científicos que estejam em conformidade com os interesses da pesquisa obtendo valores: **Pouco** (1 ou 2 pontos) ou **Muito** (3 ou 4 pontos);
2. detalhes da solução proposta para transformação de modelos computacionais, obtendo um nível de detalhamento com os seguintes valores: **Muito** (3), **Pouco** (1) e **Nenhum** (0);
3. estudos realizados nos últimos 3 anos (1 ponto), uma vez que a tecnologia vem evoluindo com o tempo, as ferramentas computacionais mais recentes tendem a utilizar tecnologias atualizadas.

3.1.4 Execução

A etapa de execução da revisão deu-se por meio da identificação das fontes de pesquisa, para então selecionar os trabalhos que serão submetidos à avaliação de qualidade seguida da extração dos dados referentes a cada trabalho e, por fim, a elaboração de uma síntese desses dados. O período de execução da busca e da seleção dos estudos ocorreu durante o mês de setembro de 2019.

3.1.4.1 Procedimento para Seleção dos Estudos

Após a definição dos objetivos de pesquisa, e com a estruturação do protocolo de RSL, o processo de seleção de estudos foi organizado em três etapas distintas, descritas a seguir:

1. Submeter a *string* de busca aos motores de busca *Web* listados na **Seção 3.1.2**. Em seguida, os artigos duplicados ou incompletos foram excluídos como determinado

nos critérios de exclusão;

2. Realizar a leitura dos títulos, resumos e palavra-chave dos trabalhos, para assim selecionar ou rejeitar os trabalhos com base nos critérios de inclusão e exclusão especificados no protocolo;
3. Realizar a leitura na íntegra dos trabalhos selecionados na etapa anterior com o intuito de extrair as informações acerca dos estudos dos mesmos (especificado na **Seção 3.1.3**) e selecionar para próxima etapa de avaliação ou excluí-los de acordo com os critérios de inclusão e exclusão.

3.1.4.2 Identificação dos Estudos Primários

De início, foram encontrados 525 estudos (subtraindo-se do total, os artigos duplicados que foram identificados) ao realizar as buscas nos motores Web. No **Quadro 3** é apresentada a fonte de busca e o número total de artigos identificados em cada uma delas, assim como na **Figura 4**, onde está exposto um gráfico em pizza destacando a proporção da quantidade de trabalhos encontrados em cada base, incluindo trabalhos duplicados.

Quadro 3 – Estudos primários identificados para análise

Base	**Retorno**
IEEE Xplore Digital Library	22
ACM Digital Library	293
Science Direct	111
Scopus	27
Web Of Science	72
Total	**525**

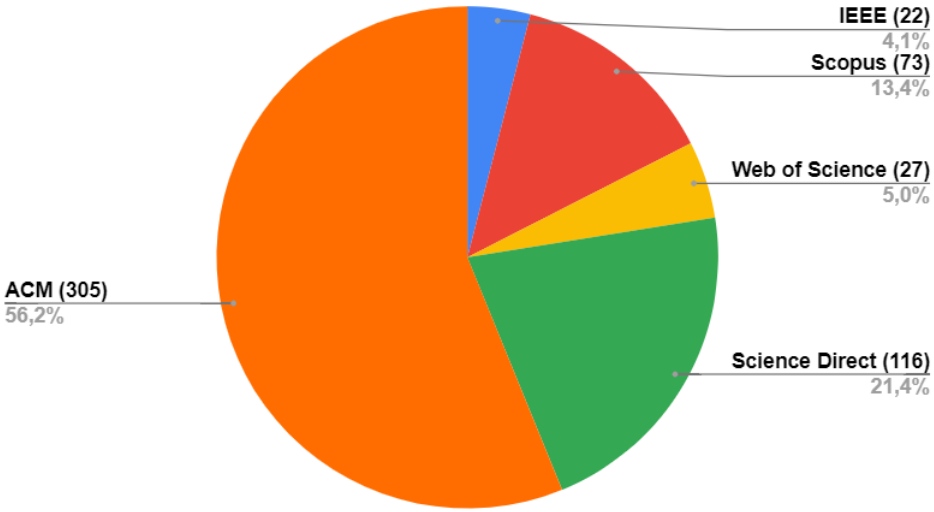
Fonte: Autoria própria (2019).

3.1.5 Seleção

Durante essa etapa todos os artigos foram classificados como **"Não Classificado"** e após a leitura dos resumos, palavras-chaves e títulos, foram atribuídos rótulos aos estudos assumindo os seguintes *status*: **Aceito**, **Rejeitado** e **Duplicado**. Para facilitar este processo, foi utilizada a ferramenta *Start*[1], que permite gerenciar todas as etapas de uma RSL. Quando se trata de trabalhos duplicados, a ferramenta identifica grande parte de forma automática, por meio da análise comparativa das referências dos estudos. Outros são

[1] Disponível em: http://lapes.dc.ufscar.br/tools/start_tool

Figura 4 – Distribuição dos estudos identificados na etapa de Seleção Primária por fonte de pesquisa.



Fonte: Autoria própria (2019).

identificados de forma manual, evitando a análise de um mesmo trabalho repetidamente.

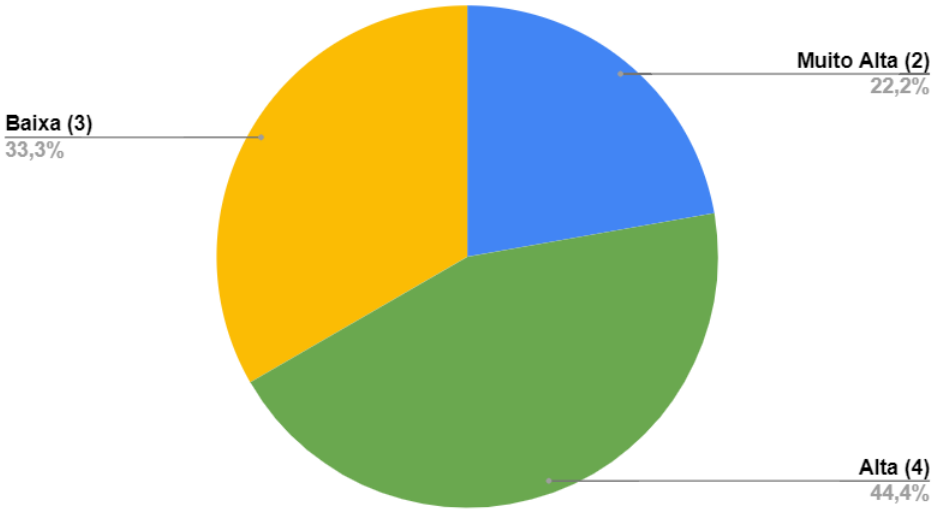
Considerando os critérios de exclusão e inclusão definidos no protocolo da RSL, na etapa de seleção foram rejeitados 493 artigos e apenas 9 foram aceitos. Da quantidade total, também foram rejeitados 50 trabalhos duplicados. Devido aos seus respectivos títulos, resumos e palavras-chave atenderem aos critérios de inclusão definidos, os nove estudos aceitos deram prosseguimento para a etapa de extração, que consistiu na leitura completa dos trabalhos para a extração das informações dos mesmos, de acordo com o definido na **Seção 3.1.3**. Para fins de organização da leitura, os trabalhos foram agrupados por ordem de relevância, selecionados de acordo com o fator de impacto de cada um, sendo definidos com auxílio da ferramenta *Start*, como mostrado na **Figura 5**. Com isso, três trabalhos foram selecionados com prioridade **Baixa** de leitura, quatro com prioridade **Alta** e dois com prioridade **Muito Alta**.

3.1.6 Extração

Através da seleção na etapa anterior, nove trabalhos foram considerados para a leitura completa e prosseguimento para a extração das informações dos estudos que cumpriram os critérios de inclusão da pesquisa e assim aplicar os critérios de qualidade, contribuindo para o objetivo da RSL.

Os artigos selecionados foram submetidos aos critérios de qualidade após a leitura completa dos mesmos e extração de suas informações. Então, os trabalhos selecionados na

Figura 5 – Prioridades de leitura dos trabalhos selecionado



Fonte: Autoria própria (2019).

etapa anterior foram analisados e submetidos a um *ranking* com base nas notas obtidas ao aplicar os critérios definidos na **Seção 3.1.3.1** com os resultados sendo apresentados na **Tabela 1**, onde os resultados da avaliação dos estudos quanto aos critérios de qualidade definidos no Protocolo RSL são resumidos. A pontuação apresentada no ranque, foi definida previamente nos critérios de qualidade, sendo possível obter uma informação derivada a partir da satisfação dos critérios de qualidade, tendo como propósito medir a adequação de cada estudo ao objetivo desta pesquisa.

Tabela 1 – Aplicação dos Critérios de Qualidade

Paper	**Qualis**	**CQ01**	**CQ02**	**CQ03**	***Ranking***
Passarini *et al.* (2015)	B1	Muito	Muito	< 3 anos	**6**
Passarini *et al.* (2013)	B2	Muito	Pouco	< 3 anos	**5**
Kern *et al.* (2014)	B5	Muito	Pouco	< 3 anos	**4**
Hu *et al.* (2014)	A2	Pouco	Muito	< 3 anos	**3**
Costa *et al.* (2015)	A1	Pouco	Muito	< 3 anos	**3**
Lasalle *et al.* (2011)	B1	Pouco	Muito	< 3 anos	**3**
Zhang e Feng (2014)	A1	Pouco	Pouco	< 3 anos	**2**
Zhou *et al.* (2010)	B4	Pouco	Muito	< 3 anos	**2**
Son *et al.* (2012)	B3	Pouco	Pouco	< 3 anos	**1**

Fonte: Autoria própria (2019).

Com o ranque estabelecido, foi possível caracterizar os trabalhos que mais atendem aos interesses da pesquisa, obtendo respostas das questões abordadas nessa RSL e assim executar a etapa de Sumarização da RSL.

Tabela 2 – Sumarização da Extração de Dados dos Estudos

Paper	**01**	**02**	**03**	**04**
Passarini *et al.* (2015)	AR e AB	Ocarina	S	S
Hu *et al.* (2014)	AR e AB	E-DEVSML	S	S
Costa *et al.* (2015)	AB	GROOVE	S	S
Zhang e Feng (2014)	AR e AB	Nenhuma	N	S
Son *et al.* (2012)	AB	M3B	S	S
Kern *et al.* (2014)	AB	Nenhuma	A	S
Zhou *et al.* (2010)	AB	Nenhuma	S	N
Lasalle *et al.* (2011)	AB	STD	S	S
Passarini *et al.* (2013)	AB e FE	Nenhuma	S	S

Fonte: Autoria própria (2019).

3.1.7 Sumarização

Mediante a etapa de extração, foi possível perceber uma grande divergência entre os trabalhos selecionados, tanto por parte da contribuição que os próprios trabalhos apresentam quanto em relação às ferramentas utilizadas em suas respectivas pesquisas. Esses trabalhos obtiveram resultados abrangentes e distintos, mas que serviram como uma base teórica elucidativa mediante a problemática de transformação de modelos, envolvendo o mapeamento de modelos entre diferentes ferramentas de modelagem, no contexto da MDE, aplicados em SMFCs.

A sumarização apresentada na **Tabela 2**, detalha a dificuldade de encontrar alguma abordagem predefinida que pudesse ser aplicada diretamente no mapeamento de modelos SMFC elaborados na ferramenta Ptolemy II® para a ferramenta Simulink®.

Também foi possível perceber na Tabela 2 uma grande heterogeneidade em questão das ferramentas de apoio utilizadas para o desenvolvimento das soluções, sendo elas:

Ocarina: processador modelo *Architecture Analysis & Design Language* (AADL) independente, escrito em Ada. Utilizada para o mapeamento de modelos AADL em redes de Petri, temporizadas (TINA) ou coloridas (CPN-AMI) (PASSARINI *et al.*, 2015). Para que a implementação seja conduzida adequadamente, o modelo do projeto deve ter detalhes suficientes para que a geração de código se torne simples, para que os programadores ou o software de geração de código possam interpretá-lo e gerar o respectivo código de programa em uma determinada linguagem de destino. Por exemplo, a ferramenta Ocarina pode executar a geração automática de código de um modelo AADL para linguagens C / C ++ ou ADA (PASSARINI *et al.*, 2015).

E-DEVSML: modelos desenvolvidos em Java e em linguagem de modelagem DEVS *(Discrete Event System Specification)* independente de plataforma que usa XML como meio de transformação. Quando o DEVS está vinculado a uma implementação específica da plataforma, os objetos de mensagem são trocados de acordo com os pares de valor de porta especificados na estrutura do modelo. Dessa maneira, as entidades são definidas como uma classe de dados para representar os tipos de mensagens e podem ser declaradas em componentes atômicos ou acoplados para reutilização. Embora a sintaxe abstrata do modelo acoplado permaneça o mais próximo possível do DEVS paralelo, qualquer transição de estado com base no conteúdo da mensagem não é realizável devido às limitações do DEVS Determinístico Finito (HU *et al.*, 2014). Na prática, quando modela-se a arquitetura do sistema com a versão anterior do DEVSML, o FD-DEVS *(Finite and Deterministic Discrete Event System Specification)* geralmente não consegue resolver um problema complexo (HU *et al.*, 2014);

GROOVE: utiliza da gramática gráfica para a transformação de modelos. Essa ferramenta adota uma representação compacta para as produções, cada produção é descrita por um único gráfico com notações diferentes onde os elementos em linhas finas tracejadas são excluídos, aqueles em linhas finas sólidas são preservados, elementos em linhas grossas tracejadas são proibidos e aqueles em linhas grossas sólidas são criados. As limitações de espaço proíbem a descrição de definições formais e da semântica de uma gramática gráfica (COSTA *et al.*, 2015);

M3-Level-based Bridges (M3B): transforma meta-modelos em um ambiente de meta-modelo intermediário. Kern *et al.* (2014) descreveram que o M3B é utilizado para a importação de meta-modelos de ferramentas de meta-modelagem. Um M3B transforma metamodelos em um ambiente intermediário de metamodelo, em seguida, um componente de ligação cria uma estrutura em árvore genérica para representar metamodelos. Embora os metamodelos importados implementem a mesma linguagem, ainda há alguma heterogeneidade entre seus elementos. Para superar essa heterogeneidade, o usuário define um mapeamento contendo correspondências entre diferentes elementos. Na próxima etapa, um gerador itera sobre os mapeamentos especificados e produz uma transformação executável de modelo para modelo. O gerador é específico para o par de ferramentas de metamodelo de origem e destino, pois a transformação deve ser capaz de ler modelos de origem e produzir modelos de destino válidos. As transformações de modelo e modelo

geradas são definidas nos metamodelos de origem e destino importados. A etapa final é a execução da transformação, incluindo a troca de modelos usando um M3B;

Simulink: gera automaticamente códigos em C e HDL. Para transformar o modelo do Simulink primeiro, usa-se um comando "save_system"no Matlab que converte o arquivo MDL do Simulink em arquivo XML. O arquivo traduzido do XML inclui todas as informações do Simulink. Além disso, o XMI (*Metadata Interchange*) baseado em XML, que é o arquivo de entrada usado para transformar o modelo, é convertido pelo conversor XMI porque o XML do Simulink não corresponde ao XMI, em seguida usa-se a transformação de modelo para transformar o arquivo traduzido do XMI do Simulink em arquivo do XMI do ECML. Mas, para modelar a técnica de transformação, o metamodelo é projetado pelo *designer*. O metamodelo do Simulink não existe para a transformação do modelo. Assim, projeta-se o metamodelo com base no arquivo XML do Simulink. A Transformação de Modelo escreve regras de transformação com a análise de semelhanças e diferenças de Simulink e ECML (SON *et al.*, 2012);

Smartesting Test Designer: utilizada para testes baseados em modelos. Essa tecnologia permite a geração automatizada de testes a partir de um modelo gravado com um subconjunto da linguagem de modelagem UML e das restrições da OCL *(Object Constraint Language)*. Esse fragmento UML / OCL é chamado UML4MBT. De acordo com Lasalle *et al.* (2011), essa ferramenta é implantada em domínios como TI corporativa e aplicativos de transações eletrônicas.

Algumas abordagens importantes foram identificadas nessa RSL, auxiliando na distinção e reconhecimento de ferramentas e conceitos para a transformação de modelos relacionados a SFCs. Nenhum trabalho abordou a transformação de modelos em específico para SMFCs, o que demonstra ser um tópico a ser estudado. Dentre os trabalhos encontrados, foram identificadas abordagens que podem ser aplicadas no mapeamento dos modelos gerados no Ptolemy II para os modelos do Simulink, especificando modelos de computação bem definidos que gerenciam as interações entre componentes. Foi observado o conceito para a identificação de elementos pertencentes a um modelo (ou meta-modelo), associando-os a outro conjunto de elementos, permitindo o acesso ou alteração das informações internas desses elementos de maneira uniforme.

3.2 Trabalhos Complementares

Nesta seção são descritos e analisados alguns trabalhos que serviram de complemento para embasar a solução proposta neste trabalho, mostrando as contribuições fornecidas em comparação a outros estudos.

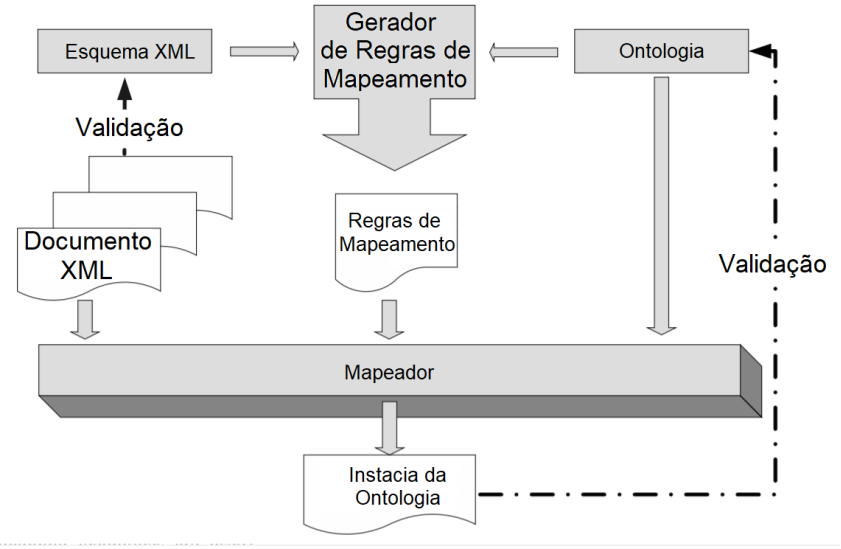
3.2.1 Mapping XML to OWL for seamless information retrieval in contextaware environments

Kobeissy *et al.* (2007) propuseram em seu trabalho o mapeamento de um modelo XML para a *Ontology Web Language* (OWL), que torna possível ao gerenciador de contexto sondar as diferentes entidades que constroem o contexto do modelo, por exemplo, dispositivos, sensores, perfis, entre outros. Seu foco não está em mapear os nós ou atributos XML para classes ou propriedades na ontologia em sua totalidade, apenas os nós apropriados e compatíveis são selecionados e mapeados.

O foco foi utilizar a estrutura e sintaxe válidas, contidas dentro do documento XML para que fossem definidas regras de mapeamento de seus elementos e atributos para conceitos definidos pela ontologia. Então essas regras de mapeamento foram aplicadas em instância XML para validação e assim prover instâncias da ontologia, como demonstrado na **Figura 6** onde o arquivo de "Regras de Mapeamento" é criado na fase de configuração do módulo de mapeamento XML para OWL, definido como "Gerador de Regras de Mapeamento". A "Ontologia" é armazenada em um banco de dados local. No tempo de execução, o "Documento XML" e a "Ontologia" são carregados no resolvedor de mapeamento chamado "Mapeador". As regras de mapeamento são carregadas no portador das regras de mapeamento. Por fim, é construída uma representação interna do gráfico de dependência e estruturas de dados relacionadas ao mapeamento na "Instancia da Ontologia". Esse processo é repetido até que todo o conteúdo do XML seja processado.

Para abordar partes de um documento XML, a utilização de expressões *XML Path Language* (XPath) para retornar um conjunto de nós correspondentes, após selecionar os nós XML, se faz presente no processo, por se tratar de uma linguagem que endereça partes de um documento XML e traz facilidades básicas para manipulação de cadeias de caracteres, números e valores lógicos. Com isso, cada singularidade da classe OWL mapeada foi criada retornando o mapeamento de propriedade do tipo de dados de acordo

Figura 6 – Visão global do procedimento de mapeamento.



Fonte: Traduzido de Kobeissy *et al.* (2007).

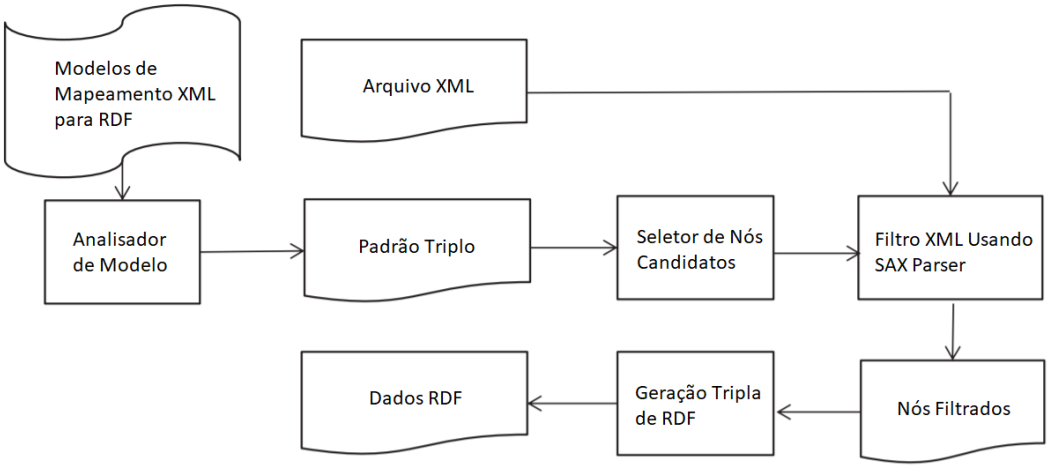
com o conteúdo de elemento ou atributo.

3.2.2 Streaming Transformation of XML to RDF using XPath-based Mappings

Huang *et al.* (2015) propuseram uma abordagem de mapeamento do XML para *Resource Description Framework* (RDF), com capacidade de processar fluxos de dados de maneira a projetar uma linguagem de modelo XML para RDF com base no XPath, permitindo uma fácil criação de regras de mapeamento.

Na visão da sua abordagem, apresentada na **Figura 7**, há a necessidade de dois arquivos de entrada, um "Arquivo XML"e um arquivo de "Modelos de Mapeamento XML para RDF". Quando o modelo é fornecido pelo usuário, um "Analisador de Modelos"é criado com o intuito de recuperar "Padrões Triplos"do arquivo de modelo. Os padrões analisados são alimentados em dois módulos principais, sendo eles o "Seletor de Nós Candidatos"e o "Filtro XML". O seletor de nós candidatos seleciona nós candidatos do arquivo XML, que satisfaz as regras de construção fornecidas no modelo. Com isso o filtro XML pode processar o documento XML com um fluxo. Se o filtro encontrar nós XML correspondentes para as regras de construção, esses "Nós Filtrados"serão usados para criar assuntos ou objetos RDF de acordo com as regras de construção. O seletor candidato conseguirá produzir um conjunto de componentes usando a "Geração Tripla de RDF"para agregar triplos semelhantes com o mesmo padrão de *template*, assim que tiver obtido nós suficientes para gerar esses componentes em "Dados RDF".

Figura 7 – Arquitetura de Mapeamento XML para RDF



Fonte: Traduzido de Huang *et al.* (2015).

3.2.3 *A Prototype of Model-Based Design Tool and its Application in the Development Process of Electronic Control Unit*

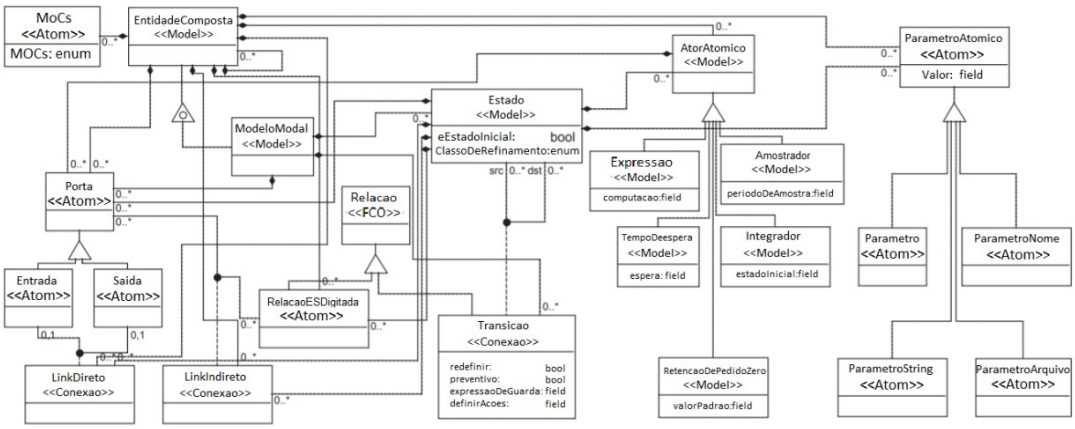
Li *et al.* (2011) entregaram aos projetistas a possibilidade de modelar sistemas embutidos com um rico conjunto de *Model of Computations* (MoCs) suportados pelo Ptolemy II®, enquanto têm a arquitetura de meta-modelagem implementada em *Generic Modeling Environment* (GME) para apoiar a transformação e interpretação de modelos. Um GME é um *Model-Integrated Computing* (MIC), ferramenta configurável baseada na modelagem específica de domínio, realização de análise e transformação de modelos.

Foi desenvolvida uma ferramenta baseada em GME chamada MoDAL em que utiliza algumas classes-chave do Ptolemy II®. O diagrama de classes do metamodelo MoDAL implementado no GME é mostrado na **Figura 8** onde dentro do ModalModel, uma Maquina de Estado Finito (MEF) é representado como o segundo nível usando Estados e Transições. Uma Transição possui dois atributos, são eles: “expressaoDeGuarda” e “definirAcoes”. Um estado contém vários componentes para representar a dinâmica contínua:“Amostrador”, “TempoDeespera” “Expressao” e “Integrador”. Em particular, um atributo denominado “ClasseDeRefinamento” descreve o Modelo de Computação (MoC) usado no refinamento desse estado.

O conceito de Objeto de Primeira Classe (OPC) é introduzido para permitir que objetos que são diferentes sejam capazes de herdar de uma classe base comum. Conceitos de modelagem, incluindo “Model”, “Atom”, “Conjunto”, “Conexao” e” Referencia”, são

definidos como OPC. Uma “EntidadeComposta” é semelhante ao “CompositeActor” em Ptolemy II® e seu objetivo principal é fornecer uma gerência hierárquica de modelos. Ela contém MoCs, “AtorAtomico”, “Porta”, “LinkDireto”, “LinkIndireto”, “RelacaoESDigitada”. Ele também pode conter outra “EntidadeComposta”. Cada “EntidadeComposta” no MoDAL contém um MoC, como o diretor em Ptolemy II®, gerenciando a execução e as interações dos componentes.

Figura 8 – Metamodelo do MoDAL implementado em GME



Fonte: Traduzido de Li *et al.* (2011).

Após a interpretação do metamodelo em GME, um ambiente de modelagem hierárquica MoDAL é construído com a possibilidade da construção ser executada de forma heterogênea. Li *et al.* (2011) apresentaram a possibilidade de modelos do Simulink® serem importados para o MoDAL através de um adaptador. As interações entre modelos podem ser organizadas hierarquicamente através da mistura adequada de vários MoCs.

3.2.4 Considerações Finais do Capítulo

Com os resultados encontrados na revisão sistemática e na pesquisa por trabalhos complementares, foi possível observar o estado da arte do tópico de transformação de modelos para SMFCs.

A principal semelhança entre as abordagens propostas pelos trabalhos apresentados neste capítulo, é o acesso continuo ao mapeamento, tornando-o o ponto central para a execução dos outros processos de cada estudo. Também foi possível observar que os autores não focaram no ambiente ao qual o mapeamento está inserido. Em relação às diferenças, os modelos utilizados pelas abordagens são distintos, assim como suas estruturas, não

havendo um melhor padrão a ser seguido.

Dessa forma, não há uma abordagem que melhor se adéqua aos paradigmas e às ferramentas de modelagem, pois é possível utilizar diversos meios para realizar o mapeamento e transformação de modelos. Com isso, notou-se a possibilidade do desenvolvimento de um *framework* para a concepção do trabalho.

4 PROPOSTA DE *FRAMEWORK* PARA TRANSFORMAÇÃO DE MODELOS

Com o intuito de contemplar o objetivo de diminuir o tempo e esforço de concepção de SMFCs e reduzir divergências entre esses modelos, este trabalho propõe um *framework* denominado TADEM para Transformação de Modelos. Uma solução que busca acelerar os processos de migração e adaptação de modelos entre diferentes plataformas de modelagem computacional, com a possibilidade de identificar os componentes atrelados diretamente ao sistema que está sendo modelado e que através de um mapeamento de componentes, tem-se a capacidade de gerar um novo modelo com características semelhantes para ser analisado e testado em uma segunda ferramenta de modelagem.

Na **Figura 9** é apresentado o cenário de aplicação do *framework* TADEM. Como pode ser observado, o processo como um todo é dividido em duas fases, a fase de análise e a fase de síntese, partindo como requisito inicial os dados obtidos através de um arquivo, cujo o conteúdo é composto pelo modelo de entrada que está configurado de acordo com as particularidades referentes ao software de modelagem primário. Após todas as etapas, um arquivo de saída contendo o modelo configurado referente à ferramenta de modelagem destino é concebido.

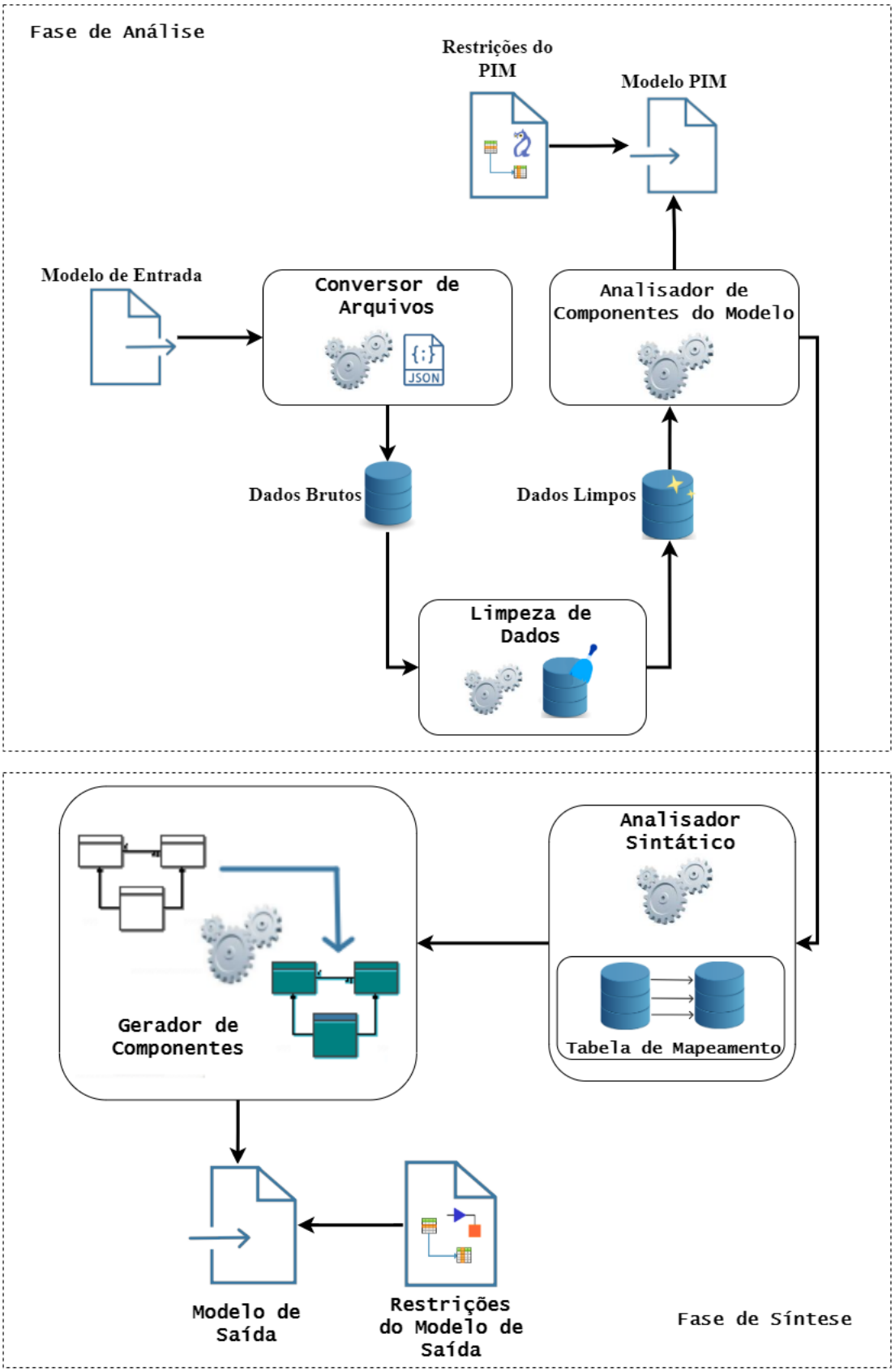
4.1 Fase de Análise

O ponto central desta fase está no modelo de entrada, pois a partir dele que todas as etapas tem como base de funcionamento. O conteúdo desse modelo é analisado sintaticamente, obtendo as strings, caracteres e valores, fazendo com que o conteúdo dentro dela seja lido e respectivamente analisado. Os componentes do *framework* envolvidos nesta fase são descritos detalhadamente nas subseções a seguir, relacionando suas entradas e saídas, partindo do modelo de entrada, padronizando seus dados convertendo seu formato, excluindo sucessivos dados irrelevantes para o modelo. Assim os componentes podem ser analisados, selecionando o conteúdo necessário para a construção de um modelo PIM.

4.1.1 Modelo de Entrada

As ferramentas utilizadas para modelagem de software, por padrão, tem seus modelos salvos na máquina do utilizador. Esses arquivos que estão armazenados precisam ter o acesso aos dados em formato aberto, ou seja, disponibilizar o conteúdo para visualização

Figura 9 – Cenário de aplicação do *framework*.



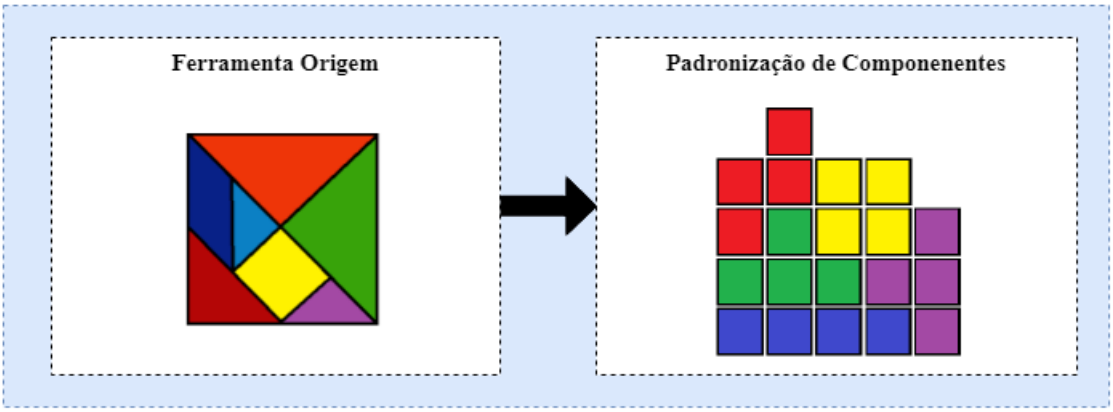
Fonte: Autoria própria (2021).

e edição por outros softwares, como editores de texto por exemplo. O arquivo necessita disponibilizar os dados (componentes, nomes de variáveis, valores, entre outros) acerca de um determinado modelo para que seja possível a extração e leitura do seu conteúdo pelo *framework*. As regras que regem a composição do arquivo precisam estar em formato de textos com significado em uma linguagem formal, isto é, que seja possível abstrair sua semântica.

4.1.2 Padronização e Limpeza dos Dados

A partir do conteúdo referente ao modelo encontrados no arquivo, há uma conversão dos dados, visando a padronização de tipagem com intuito de facilitar a interação durante todos os processos em ambas as fases. Os arquivos gerados por uma ferramenta específica contém dados de configuração do próprio software de modelagem em que o modelo foi primariamente criado, por este motivo é necessário realizar uma filtragem para selecionar apenas os dados referentes aos elementos que compõem o modelo em si, dada a sua dinâmica comportamental. A **Figura 10** ilustra o processo de padronização dos dados, onde o as regras de sintaxe e semântica particulares do arquivo de entrada são convertidos para um formato que será utilizado durante todo o processo de mapeamento. Após a padronização é realizado o processo de limpeza, que consiste no descarte de dados desnecessários ou obsoletos que fazem referencia somente à ferramenta de origem do modelo. Com isso tem-se a possibilidade de maior legibilidade para mapear e configurar os componentes da ferramenta de origem para a ferramenta de destino.

Figura 10 – Padronização dos Componentes.



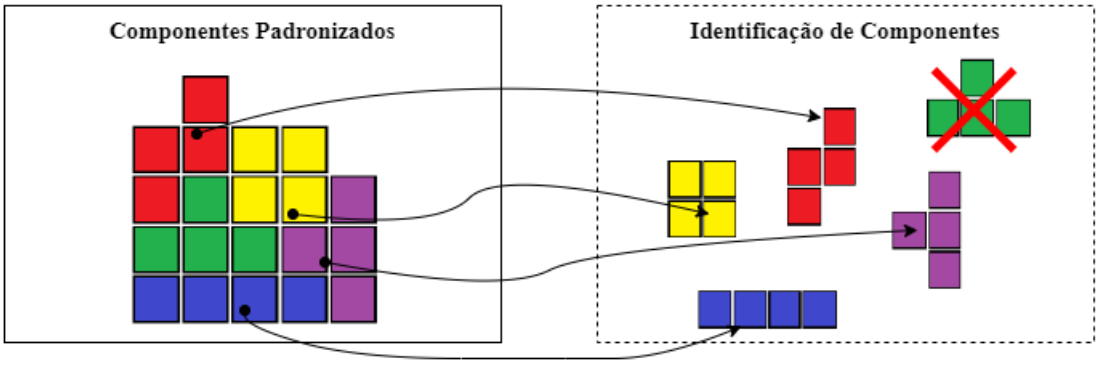
Fonte: Autoria própria (2021).

4.1.3 Análise de Componentes e Atributos

Como ilustrado na **Figura 11**, o processo de análise de componentes e seus respectivos atributos por meio do Analisador de Componentes do Modelo, traz ao *framework* a possibilidade de remover os componentes redundantes e indesejados. A verificação dos caracteres, tais como caracteres especiais, símbolos e letras também é importante, pois durante a escrita do modelo ou documento, utilizar tais caracteres, pode causar erros de codificação para arquivos, banco de dados e linguagem de programação, o dificultaria a implementação do *framework*.

O analisador de componentes analisa as palavras-chave lexicais e estruturais no código para fazer a correspondência do componente que o representa dentro de um *token* com o significado correto. Esse *token* tem sua representação alinhada ao PIM, para que a sua geração tenha todo o conceito do modelo do software a ser transformado mesmo possuindo algumas características do modelo de origem.

Figura 11 – Identificação de Componentes Essenciais do Modelo.



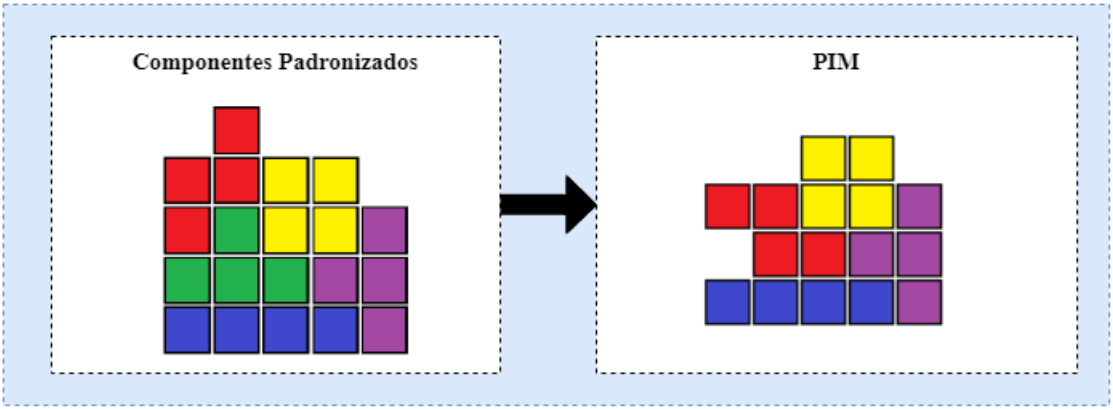
Fonte: Autoria própria (2021).

4.1.4 Construção do Modelo Independente de Plataforma

A abordagem para transformar o PSM em um PIM é definido com regras de transformação. Essas regras de transformação são usadas para transformar um modelo em outro modelo. São um conjunto de diretrizes projetadas para tornar a tradução de um formato para outro o mais fácil possível. É importante levar em consideração todas as restrições de configuração que esse modelo independente possui, podendo tais restrições serem elementos de nível hierárquico, informações do meta-dados como títulos e *charsets*.

Utiliza-se uma regra comum de transformar um modelo de origem em modelos de destino que possuem restrições dependentes. Essas cláusulas condicionais permitem expressar um nível mais alto de dependência nos modelos de saída, sem reduzir a complexidade geral do modelo. O PIM ainda teria características e nomenclaturas referentes ao modelo de origem como pode ser visto na **Figura 12**.

Figura 12 – Processo de Construção do Modelo Independente de Plataforma.



Fonte: Autoria própria (2021).

A utilidade do PIM se dá em futuras transformações, com objetivo de atualizar ou construir um novo modelo conceitual base, para que possa ser gerado novos modelos a partir deste, diminuindo ainda mais o tempo e esforço de desenvolvimento, não só em plataformas que o *framework* está sendo implementado, como também em terceiras ferramentas de modelagem, necessitando do desenvolvimento da tabela de mapeamento para isto.

4.2 Fase de Síntese

Na fase de síntese do *framework*, um grupo de mapas ou representações é criado. Essas representações do mapa podem conter qualquer tipo de informação necessária para mapear as saídas finais para os componentes do modelo. A questão chave a ser observada é que essas representações do mapa normalmente são baseadas em *tokens*, que terão que aderir ao formato especificado pela plataforma destino.

4.2.1 Tabela de Mapeamento

A tabela de mapeamento de modelo compara o comportamento estrutural dos modelos usando apenas um único conjunto de dados estruturais. Pode-se visualizar facilmente as relações entre as estruturas dos modelos em termos das relações existentes. Quando o *framework* identifica um componente semelhante em qualquer um dos registros na tabela de mapeamento, ele será capaz de obter as informações contidas naquela célula diretamente. Dessa forma, é possível selecionar qualquer componente em particular e compará-lo com o componente correspondente na ferramenta destino.

Uma biblioteca de componentes armazena as várias representações de componentes de um objeto em um formato que é conveniente para o *framework*. Como o nome sugere, uma biblioteca de componentes permite que o mapeamento seja conveniente à representação de cada componente no arquivo de cada modelo, que a priori é estabelecida manualmente de acordo com escopo do modelo.

Outro uso fornecido pela tabela de mapeamento é que ela ajuda a identificar a dependência dos componentes do modelo. Além disso, a dependência de um componente individual também pode ser especificada. Nesse caso, se os dois componentes não forem semelhantes em função, eles terão que se submeter a regras de mapeamento. Portanto, esse recurso ajuda a identificar o modelo correto para o tipo de componente e vice-versa.

4.2.2 Gerador de Componentes

Quando o *framework* identifica o componente que foi alocado na tabela de mapeamento, dá-se sequência à criação desses componentes de acordo suas características e atributos, sempre analisando sua dependência e interligação com outros componentes. Os componentes que possuem uma dependência de outros componentes só podem ser adicionados após esses componentes de dependência já tiverem sido alocados pelo *framework*. Isso incluí também a relação que existe entre esses componentes, o qual podem não ser limitados apenas a pares.

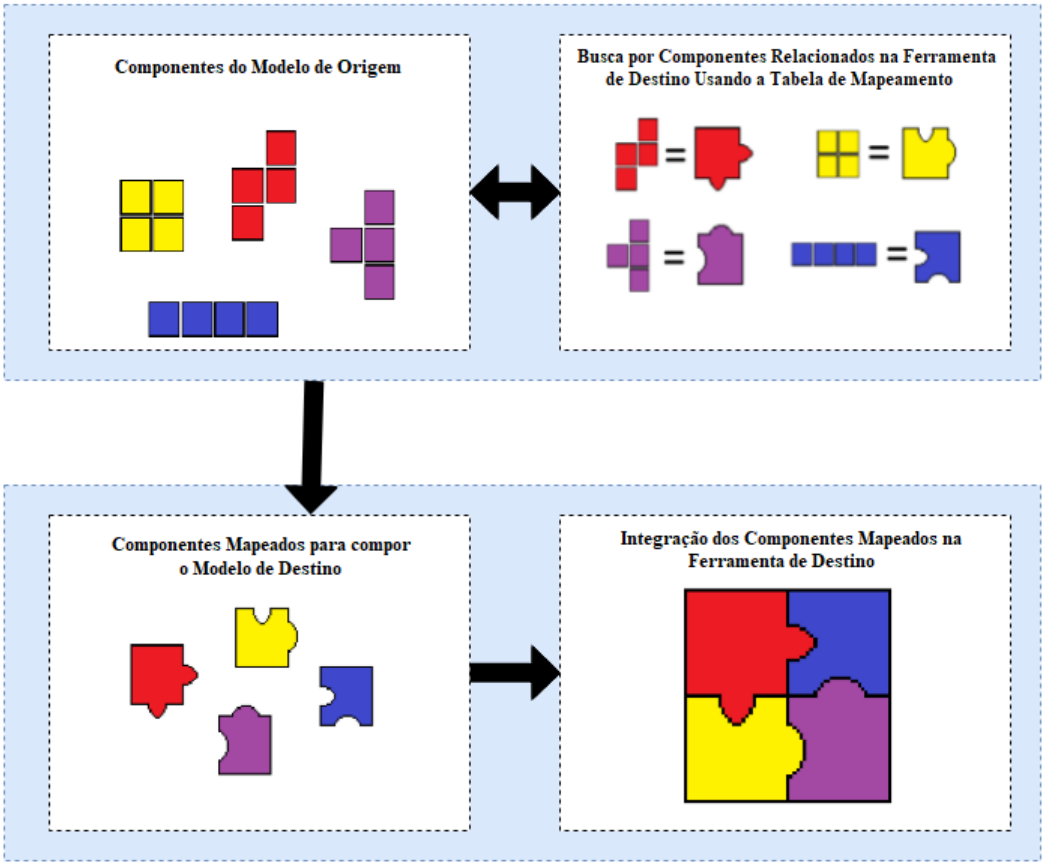
4.2.3 Construir um Modelo Dependente de Plataforma

Com todos os componentes possíveis alocados pelo *framework*, segue a estruturação do arquivo no qual a ferramenta destino necessita para executar o modelo, onde cada

componente deve ser posicionado, como também identificar o comportamento que esses componentes exercem dentro da ferramenta destino, como mostrado na **Figura 13**.

Modelos criados a partir de uma ferramenta, possuem dados em comum para todo e qualquer modelo criado a partir dela. Esses dados não fazem referência aos componentes essenciais do modelo, são formalidades para que a própria ferramenta consiga obter informações sobre a máquina que o seu *software* está instalado, versão da ferramenta, dados sobre usuário, dentre outros. Com isso é possível identificar essas informações em comum para o modelo de saída, atribuindo-as previamente ao conteúdo composto do arquivo que está sendo gerado.

Figura 13 – Processo de Construção do Modelo Dependente de Plataforma.

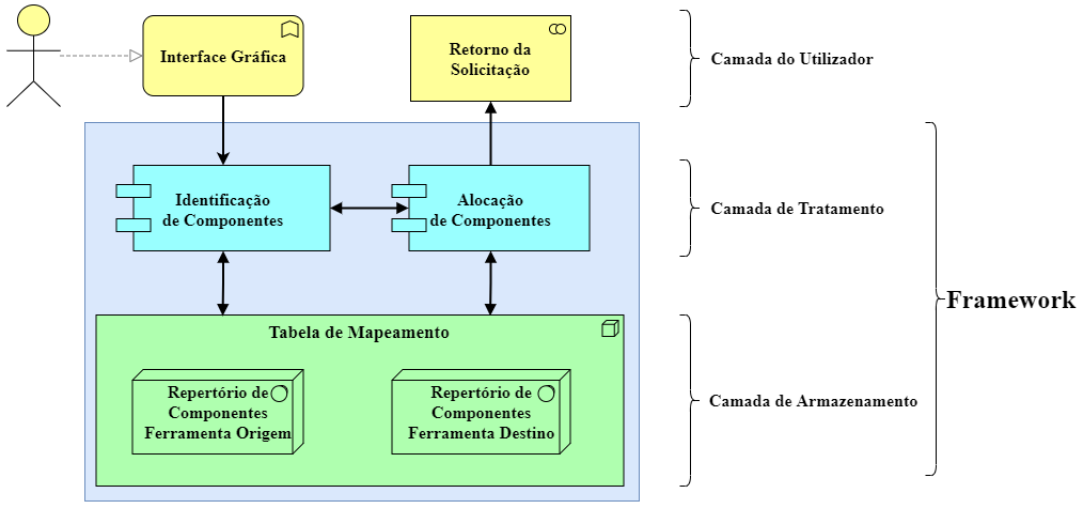


Fonte: Autoria própria (2021).

4.3 Arquitetura do *Framework* TADEM

Para que o usuário possa, de maneira facilitada, priorizando tornar rápida a utilização do *framework*, sem a necessidade de um conhecimento ou estudo prévio, se faz uso da interface de um sistema construído para esse propósito. O intuito é de que o usuário forneça apenas o arquivo contendo o modelo de entrada a ser transformado.

Figura 14 – Arquitetura do *framework* TADEM.



Fonte: Autoria própria (2021).

Com a idealização do cenário de utilização do *framework*, foi possível descrever sua arquitetura. Com isso, o *framework* pode entrar em funcionamento, lendo e coletando os dados acerca do modelo. Esses dados podem então ser processadas para dar início a listagem de componentes presentes no modelo.

Podemos descrever o *framework* apresentado na **Figura 14** da seguinte maneira. A camada do utilizador dispõe de dois componentes, a interface gráfica contém uma classe que carrega o arquivo do modelo e invoca um método que checa a compatibilidade desse arquivo, verificando se há possibilidade de continuar com o processo de transformação. Nessa camada também é atualizada a informação sobre o retorno da solicitação da transformação com informações sobre o caminho do arquivo no sistema.

Na sequencia em que os componentes forem sendo encontrados e identificados como parte do modelo do *software* em si e não de componentes referentes à ferramenta de origem, um novo conjunto de componentes é listado de acordo com a relação identificada na tabela de mapeamento, onde estão armazenadas as relações entre os componentes da

ferramenta de origem e componentes, ou conjunto de componentes, da ferramenta destino.

O framework possui duas camadas para tratar os dados que foram submetidos. A camada de tratamento possui o componente de identificação de componentes do modelo, onde a sintaxe do documento é analisada, verificando se algum desses componentes estão presentes no repertório de componentes da ferramenta de origem, que por sua vez, está situado na camada de armazenamento. A alocação dos componentes em um novo arquivo presente no repertório de componentes da ferramenta destino, se dá quando um componente é encontrado e comparado na tabela de mapeamento. Os componentes que estão nos repertórios de componentes, de ambas as ferramentas, são alocados de acordo com a função associada entre as duas ferramentas, formando o a tabela de mapeamento.

O *framework* faz da tabela de mapeamento como "armazenamento" para listar o conjunto de componentes que estão sendo identificados, alocando-os todos em uma nova lista de acordo com que novos componentes são encontrados e identificados na coluna referente a ferramenta destino.

4.4 Considerações Finais do Capítulo

Com as informações adquiridas a partir dos trabalhos apresentados no referencial teórico (**Capítulo 2**) e nos trabalhos relacionados (**Capítulo 3**), com os resultados da RSL e dos trabalhos complementares, foi possível idealizar e estruturar como se daria uma transformação de modelos entre diferentes ferramentas de modelagem. As informações destes trabalhos serviram como base para a construção do *framework* apresentado neste capítulo. O suporte para a transformação de modelos se dá em relação à tabela de mapeamento, onde são listados todos os componentes possíveis que mantém relação entre as ferramentas, onde o modelo é originado e a qual se destina. Já as regras de transformação, se baseiam justamente no mapeamento, e variam de acordo com as ferramentas a serem trabalhadas. Quando esses conhecimentos são unidos, é possível aplicar o *framework* a diversas ferramentas de modelagem, não se limitando às ferramentas adotadas como base para o desenvolvimento e validação do *framework* TADEM.

5 TADEM: DESENVOLVIMENTO E VALIDAÇÃO DO *FRAMEWORK*

Neste capítulo, foram apresentados o desenvolvimento e a aplicação do *framework* TADEM, tal como sua utilização na prática.

5.1 Introdução

Não basta apenas a idealização do *framework*, há a necessidade de implementar e testar sua utilização em cenários reais com intuito de demonstrar sua utilidade. Assim, partiu-se para entender como se deu o desenvolvimento do TADEM, explicando as tecnologias escolhidas para sua composição e o motivo da escolha de tais tecnologias. Neste capítulo, é demonstrada a sequência de passos procedimentais do *framework* seguindo os meios no qual se fizeram necessários para sua composição.

Para demonstrar o uso do *framework* TADEM, uma aplicação foi desenvolvida e executada dando como entrada um dos modelos computacionais de cenário clínico criado no trabalho realizado por Silva *et al.* (2015), neste caso o controlador do modelo da bomba de insulina. Seu desenvolvimento se deu na ferramenta de modelagem Ptolemy II®, em que o arquivo do modelo encontra-se no formato XML. Com isso, por meio da aplicação oriundo do *framework* TADEM gerou-se um modelo com funcionalidades equivalentes para serem executadas na ferramenta Simulink®.

5.2 Implementação de Referência

Para todo processo de desenvolvimento de software necessita-se de uma metodologia coerente. Visto que, mesmo com os requisitos claros, existe a possibilidade de mudanças.

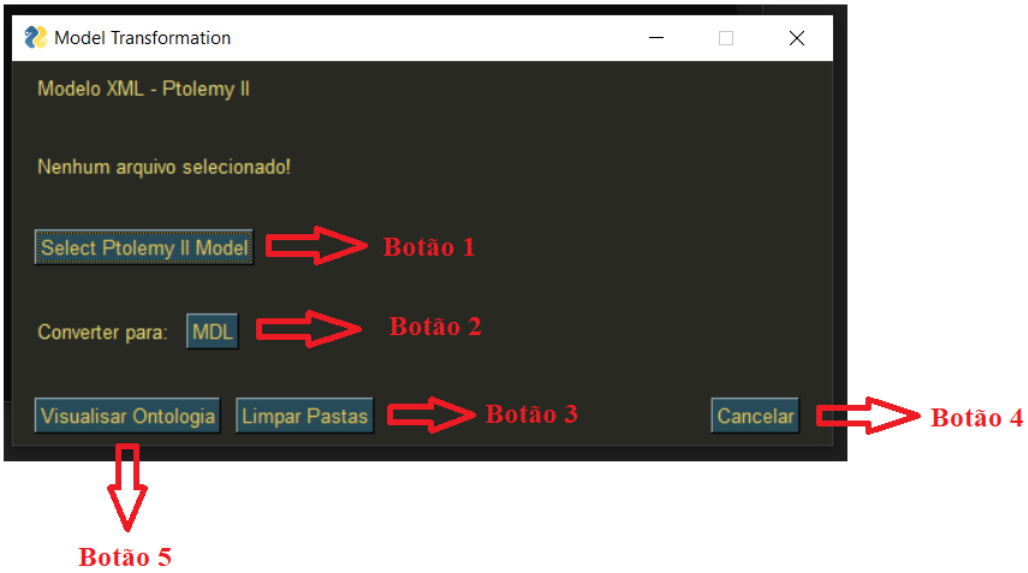
A fim de controlar os processos e aperfeiçoar a previsibilidade, foram seguidos os pilares descritos por Santos *et al.* (2013) sendo eles: **Transparência**, onde é garantido que todos os processos que envolvam o resultado sejam claros; **Inspeção**, realizada durante desenvolvimento, com o objetivo de detectar as variações, sendo possível ajustar cada processo; e **Adaptação**, com as inspeções realizadas foi possível adaptar o processo para as variações encontradas.

5.2.1 Utilização do Framework pelo Usuário

A primeira etapa para a concepção do *framework* TADEM, partiu da idealização do esboço com intuito de apresentar uma melhor maneira de organizar os componentes para a sua utilização. Com isso, para o projeto inicial foi obtido o resultado mostrado na **Figura 15**.

A interface para a utilização do *framework* TADEM pelo usuário busca priorizar a rápida interação com o único objetivo de realizar a transformação entre os modelos.

Figura 15 – Tela Principal da Aplicação do *Framework* TADEM



Fonte: Autoria própria (2021).

Diversas linguagens de programação, *frameworks* e várias ferramentas possibilitam a implementação do TADEM. Com a finalidade de trazer um rápido resultado para o desenvolvimento, por questões de afinidade e baixa curva de aprendizagem, a linguagem de programação Python foi escolhida como um meio para alcançar o objetivo da pesquisa.

O usuário interage com interface de maneira simples e intuitiva, de maneira a entender rapidamente onde deve-se clicar e o que fazer. O foco não é aprender uma nova ferramenta de desenvolvimento de modelos, mas sim, evitar o consumo de tempo e esforço no desenvolvimento de modelos.

Ao clicar no “Botão 1” é aberta uma janela padrão do sistema operacional para seleção de arquivos. Com o “Botão 2” inicia-se a transformação para o formato/ferramenta de destino. Alguns arquivos auxiliares são gerados para fins de verificação ou análise,

auxiliando o controle e transporte dos dados do modelo. A função executada ao clicar no "Botão 3" elimina esses arquivos auxiliares, pois após a transformação, não são necessários para a ferramenta. Para encerrar a aplicação, o "Botão 4" pode ser selecionado. Por fim, o formato ontológico foi escolhido para servir como o PIM, uma vez que se é possível desenvolver um modelo a partir de seu código, com isso, o "Botão 5" permite abrir o local em que está armazenado o arquivo OWL referente ao modelo gerado.

O componente visual da aplicação do *framework* TADEM foi modelada de maneira a facilitar a compreensão acerca do processo de transformação. Desenvolvida utilizando o pacote tkinter[1] ("interface Tk") que se trata de uma interface Python padrão para o kit de ferramentas GUI *Tool Command Language* (Tcl).

O objetivo da interface única é que o usuário entre com o modelo a ser mapeado e dê início ao processo de transformação em apenas um clique. Aguardando o resultado da transformação que é reportado através de uma janela flutuante (modal), indicando o caminho ao qual o *framework* TADEM armazenou o documento.

5.2.2 Entendimento do Cenário de Uso

Com a mudança de cenário, deixando de lado a interação do usuário e partindo para o funcionamento do *framework*, foi iniciada a coleta de informações acerca da ferramenta primária em que o modelo de entrada a ser transformado foi criado.

Como descrito no capítulo anterior, inicia-se uma padronização do conteúdo do modelo de entrada, ou seja, alterar a estrutura que o arquivo está formatado para um estrutura mais simples e compreensível, ajudando no processo de desenvolvimento.

Após análise do código contido no arquivo gerado pelo Ptolemy II® e identificação de como se deu a organização dos componentes em um arquivo XML, estabeleceu-se que o conteúdo extraído seria convertido para um formato mais legível no intuito de facilitar a identificação dos componentes úteis. Para isso, o formato *JavaScript Object Notation* (JSON) foi escolhido por ser um formato leve de intercâmbio de dados e com fácil leitura e escrita para os humanos.

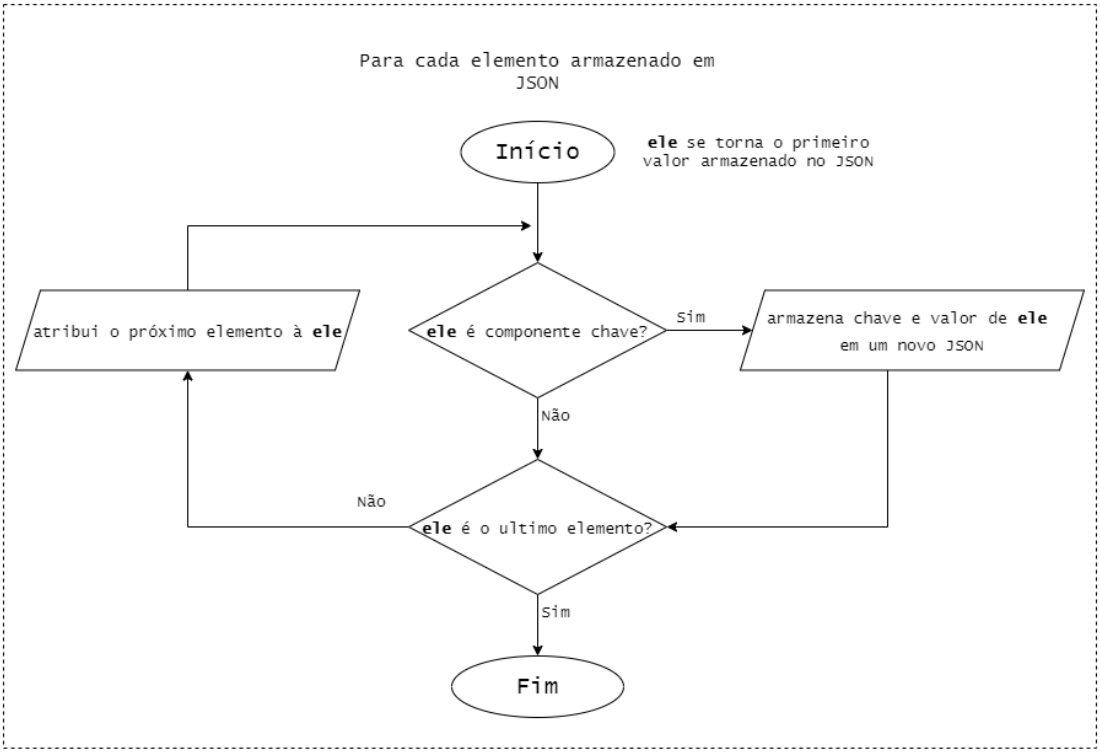
Também é possível analisar e gerar conteúdo por parte das máquinas. O módulo de processamento de XML, uma biblioteca contendo a interfaces Python para o processamento dessa linguagem que são agrupadas no pacote *"xml"*, facilitou a conversão sem prejudicar

[1] Disponivel em: docs.python.org/3/library/tk.html

o conteúdo do arquivo. Uma análise do modelo dentro da ferramenta Ptolemy II® em paralelo à análise do código JSON fez-se necessária para a identificação dos termos chave, usados para identificar os componentes do modelo. Essas chaves são nomes de tipos de atores contidos no Ptolemy II® para identificação pela própria ferramenta, assim tem-se um *"array"* contendo esses termos.

Assim, é possível fazer uma listagem de nomes de componentes comparando cada elemento do arquivo JSON a ser transformado com os termos chave encontrados no Ptolemy II®. Como demonstrado no fluxograma da **Figura 16**, um elemento chave (variável **ele**) é buscado no arquivo por vez, onde todos os nomes e valores arquivo JSON são anualizados e comparados com os termos chave. Caso o componente seja identificado como um componente chave do modelo, seu nome e valor é armazenado em uma nova estrutura JSON. Essa função é finalizada após a análise do ultimo elemento.

Figura 16 – Busca por Componentes Chaves no Arquivo JSON.

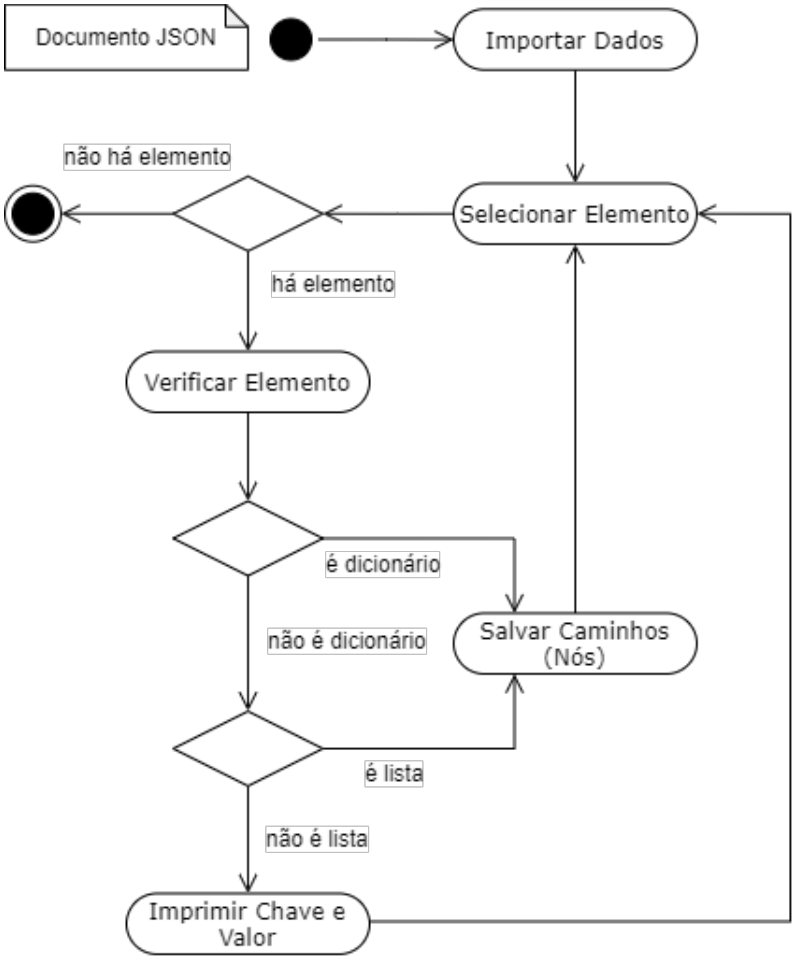


Fonte: Autoria própria (2021).

Na **Figura 17** é demonstrado um fluxograma da varredura do documento JSON, onde cada nó é percorrido. O objetivo é identificar se o elemento que está sendo percorrido no momento é: uma *"string"*, contendo o valor do componente referente a chave na linha atual; um *"array"* contendo um conjunto de componentes, não sendo referenciado como

um componente em si, mas uma lista de outros componentes; ou um “objeto”, contendo um conjunto de componentes que são referenciados por um componente, chamado de dicionário.

Figura 17 – Varredura no Documento JSON.



Fonte: Autoria própria (2021).

5.2.3 Listagem e Identificação de Componentes

Para que pudesse ser identificado quais componentes seriam descartados e quais poderiam prosseguir para análise, cada linha do arquivo teve que ser analisada, onde cada componente que fosse identificado como relevante, imediatamente, iria para alocação em memória e posteriormente ser adicionado em um novo arquivo com formato JSON contendo apenas o conteúdo selecionado como cerne do modelo. O conteúdo selecionado conforme a **Tabela 3**, onde cada valor referente a um componente, passa pelo nome da variável, que por sua vez está atrelada ao seu tipo, e por fim, na sequência, os nós listados durante a varredura que foi realizada percorrendo cada nó.

Tabela 3 – Organização dos Componentes Selecionados

nó	**tipo**
entity.@class	ptolemy.actor.TypedCompositeActor
...	...
entity.property[5].@class	ptolemy.data.expr.Parameter
entity.property[5].@class	ptolemy.data.expr.Parameter
...	...

nome	**valor**
entity.@name	Insulin_Pump_Actuator2
...	...
entity.property[5].@name	BloodGlucose_high
entity.property[5].@value	188
...	...

... : Componentes Ocultos
Fonte: Autoria própria (2021).

5.2.4 Criação do PIM

Para alocar os componentes selecionados em uma estrutura independente de plataforma, primeiro foi preciso identificar qual seria essa estrutura. A maioria dos trabalhos disponíveis na literatura sobre PIM utiliza a estrutura *Unified Modeling Language* (UML) para sua concepção, mas como demonstrado pelo trabalho de Kobeissy *et al.* (2007), é possível utilizar o formato ontológico para este fim. A principal vantagem que levou à escolha em utilizar uma ontologia formal, ao invés de uma estrutura UML, foi a verificação automática da consistência do modelo que ela suporta. Com isso, o PIM foi especificado em OWL, que é a linguagem recomendada pelo *World Wide Web Consortium* (W3C) para a construção de ontologias formais[2]. Para a estruturação do documento OWL verificou-se que a sintaxe *Terse RDF Triple Language* (Turtle) era a mais adequada, pois ela permite uma escrita mais simples, em forma de texto compacto e natural. Na **Figura 18** é apresentado um exemplo que utiliza a sintaxe Turtle, retirado da documentação da linguagem[3]. Os componentes listados seguem o padrão de nomenclatura do Ptolemy II® e são alocados por uma função recursiva dividindo os componentes em "classes" e "indivíduos". Em uma ontologia, as classes representam conjuntos de indivíduos do domínio, enquanto os indivíduos são objetos ou instâncias dessas classes (EUZENAT *et al.*, 2007). Após todos os componentes serem alocados em uma variável, uma segunda função armazena o conteúdo

[2] Disponivel em: w3.org/TR/owl2-overview/
[3] Disponivel em: w3.org/TR/turtle/

em um arquivo OWL.

Figura 18 – Exemplo da sintaxe Turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:titulo "Especificação da Sintaxe RDF/XML (Revisada)" ;
  ex:editor [
    ex:nomeCompleto "Dave Beckett";
    ex:paginaInicial <http://purl.org/net/dajobe/>
  ] .
```

Fonte: Beckett *et al.* (2014).

5.2.5 Mapeamento de elementos do modelo inicial para elementos dos modelo final

A tabela de mapeamento limita-se aos componentes identificados nos modelos de cenários clínicos de Silva *et al.* (2015), que fora, estudados e utilizados para concepção e validação do *framework* TADEM. Há a necessidade de se identificar quais os componentes tem sua equivalência entre as duas ferramentas.

Para desenvolver esse mecanismo, primeiro foi estabelecido um conjunto de regras de mapeamento primitivas. Esse conjunto de regras são a lógica principal usada no processo de mapeamento de componentes entre as ferramentas de origem e destino. Durante o processo, foram instanciados todos os componentes identificados na tabela de mapeamento, os que não estiverem mapeados, não são instanciados, impossibilitando seu mapeamento. básicos necessários.

No processo de desenvolvimento, foi identificado que algumas combinações de mapeamentos necessitavam de um agrupamento de componentes, de maneira que para conseguir obter a dinâmica comportamental de um componente da ferramenta de origem, é necessário integrar um subconjunto de componentes na ferramenta de destino e, vice-versa. Assim, essas combinações foram utilizadas para formar modelos de mapeamentos. Na **Tabela 4** é listada uma parte dos mapeamentos.

A tabela de mapeamento contém elementos do modelo de entrada da ferramenta de origem e seus parâmetros são os elementos a serem instanciados na ferramenta de destino para compor o modelo de saída. Por exemplo, o elemento contido no modelo do Ptolemy II® é *"Scale"*, o parâmetro relacionado no Simulink® é a junção de dois componentes a

"Constant" e *"Product"*. Os demais componentes e agrupamentos de componentes podem ser visualizados no **Apêndice A**.

Tabela 4 – Trecho da Tabela de Mapeamento

Ptolemy II	**Simulink**
Const	Constant
Scale	Constant Product
TypedCompositActor	SubSystem
TypedIOPort.output	outport
TypedIOPort.input	inport
...	...

Fonte: Autoria própria (2021).

5.2.6 Criação do PSM

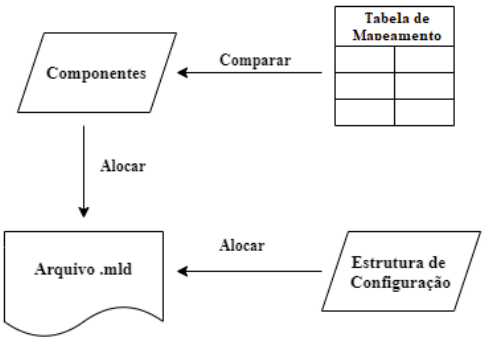
Já com todos os componentes identificados, é iniciado o processo de alocação dos componentes como ilustrado na **Figura 19**. Ao consultar a tabela de mapeamento pode-se verificar qual o componente ou subconjunto de componentes com estrutura e comportamento equivalente no Simulink® para ser adicionado no modelo destino. O nó contido na *string* do componente, informa qual a ordem hierárquica onde esse componente deve ser alocado no arquivo MDL do Simulink®. O alocação do posicionamento desses componentes mapeados para o modelo de saída se deu de maneira incremental, independendo do posicionamento dos componentes do modelo de entrada. A estrutura base do arquivo MDL contendo as informações de configuração para execução do modelo no Simulink®, tem seu conteúdo fixo, sem alteração, utilizando configurações padrão para o funcionamento do modelo pela ferramenta. Esse conteúdo base é incrementado de acordo com que os componentes são identificados e comparados na tabela de mapeamento.

5.3 Validação

Para verificar a validade e funcionalidade do *framework* TADEM por meio da aplicação desenvolvida, considerando as ferramentas de origem e destino para o seu desenvolvimento inicial, foram aplicados estímulos de validação procurando respostas em termos de mecanismos internos ao modelo.

Dessa maneira, as etapas seguidas foram: construção do protótipo de um programa;

Figura 19 – Sequência do *Framework* TADEM para a Alocação dos Componentes.



Fonte: Autoria própria (2021).

utilização de um modelo no contexto médico físico-cibernético como artefato de estudo; e fornecimento do artefato ao cenário de teste para observar as respostas.

Essas respostas, positivas ou negativas, são aplicadas observando o modelo de saída do teste, analisando como a saída poderia ter sido produzida pelos mecanismos internos do *framework* TADEM. O objetivo é expor o modelo a estímulos controlados e analisar de maneira detalhada como se deu cada resposta, seja em relação ao processo de mapeamento, à necessidade de complementação do modelo transformado e, por fim, ao comportamento do modelo de saída.

5.3.1 Análise do Problema e Projeto de Validação

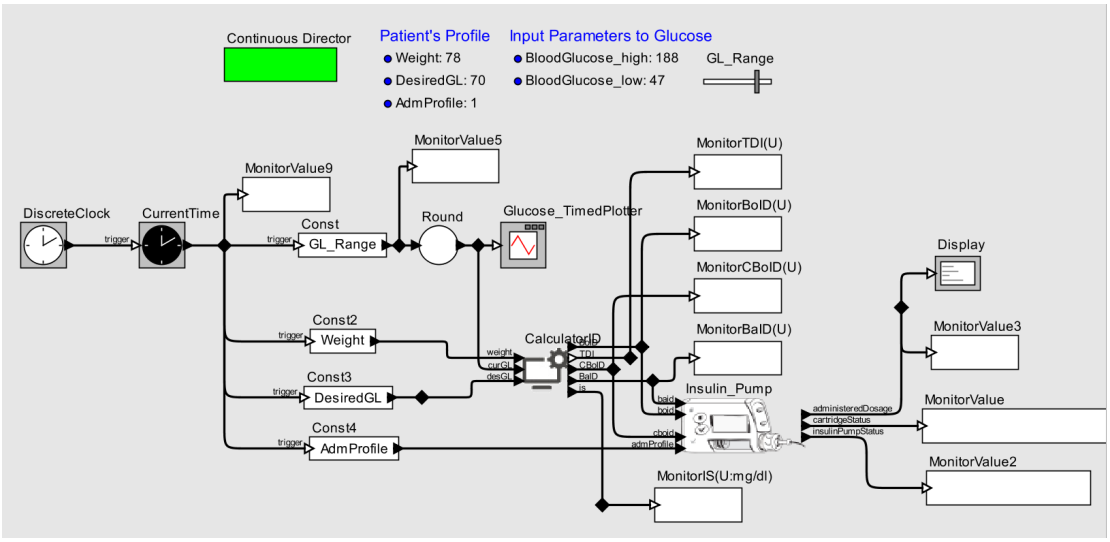
De maneira restrita, o objeto de estudo se dá com um modelo de bomba de insulina projetado no Ptolemy II®, mostrado na **Figura 20**, interagindo com o *framework* TADEM.

Para este trabalho, definiu-se a população de pesquisa como o conjunto de todos os componentes do modelo de bomba de insulina, se tratando de um cenário criado para analisar o comportamento do modelo quando usado por pessoas com Diabetes. Este cenário foi escolhido por já estar totalmente desenvolvido no trabalho de SILVA *et al.* (2015) e ser de grande importância, pois o diabetes mellitus é uma doença crônica que afeta aproximadamente 463 milhões de pessoas em todo o mundo, segundo a Organização Mundial da Saúde (WHO, 2020).

No contexto de uso desse modelo de cenário clínico, o paciente deve ajustar a bomba de insulina de acordo com a prescrição médica e a dieta diária. A programação de sua bomba envolve a escolha de um perfil de administração de: insulina basal, secretada de forma contínua, para manter minimamente constante durante o dia; e a especificação

de doses de insulina em bolus, liberada em quantidades modificadas e padrão, em horários específicos do dia, para impedir que o açúcar dos alimentos se acumule no sangue (SCHEINER; BOYER, 2005).

Figura 20 – Modelo da Bomba de Insulina no Ptolemy II®.



Fonte: Silva *et al.* (2014).

Para a validação foi realizado um experimento de mecanismo de caso único para avaliar a implementação e investigar o problema com o intuito de receber informações sobre o comportamento do *framework* TADEM. Para realizar o experimento, primeiro foi necessário analisar a estrutura organizacional dos modelos gerados pelas ferramentas Ptolemy II® e Simulink®.

Os modelos gerados a partir do Ptolemy II® são organizados da seguinte maneira: ***@class*** identifica a classe de domínio de cada componente; ***@name*** é um nome único atribuído ao componente na camada. Vale ressaltar que componentes dentro de outros componentes, em hierarquias diferentes, podem ter o mesmo nome; ***@value*** é o valor atribuído ao componente, alguns não tem a necessidade de receber valores; ***entity*** representa componentes com alguma complexidade lógica. Os modelos no PtolemyII® são construídos dentro de uma entidade superior. Outras entidades são criadas dentro de outras entidades, possuindo sua própria classe, nome, links, portas, propriedades e outras entidades; ***port*** atribui um nome e o tipo de porta para recebimento ou envio de dados de um componente; ***relation*** é uma associação entre componentes, sinalizando o "caminho" percorrido por algum dado; ***link*** associa a porta de um componente à porta de outro componente através da *relation* a partir do nome do componente; ***property*** são propriedades das entidades,

podem ser dados referentes ao componente com relação ao Ptolemy II®, determinando seu comportamento e aparência dentro do programa, mas também podem ser componentes importantes para o modelo, como parâmetros.

As principais classes do Ptolemy II®, identificadas no modelo apresentado na **Figura 20**, são descritas a seguir segundo a documentação da ferramenta [4]:

- ***ModalModel:*** é uma representação de comportamentos de conjuntos finitos e regras que gerenciam a transição entre esses conjuntos;
- ***TypedCompositeActor:*** é uma agregação de atores;
- ***Parameter:*** monitora as entradas definindo o parâmetro de valor igual a cada *token* de chegada;
- ***MonitorValue:*** exibe as entradas definindo o parâmetro de valor igual a cada *token* que chega;
- ***IntRangParameter:*** é um parâmetro do tipo inteiro com um intervalo limitado. Seu valor é um *token* inteiro que é restringido para ficar dentro dos limites especificados por seus dois parâmetros;
- ***DiscreteClock:*** produz um sinal periódico, uma sequência de eventos em intervalos regularmente espaçados;
- ***TypedIOPort:*** é uma porta de entrada e saída com um tipo de dado;
- ***CurrentTime:*** produz um *token* de saída em cada disparo com um valor que é a hora atual;
- ***Const:*** produz um valor de saída constante;
- ***ContinuousDirector:*** é um domínio cronometrado que suporta sinais de tempo contínuo, sinais de eventos discretos e combinações dos dois. Existe uma noção global de tempo da qual todos os atores estão cientes;
- ***Round:*** gera um *token* de saída em cada disparo com um valor que é igual ao valor arredondado especificado da entrada;
- ***TimePlotter:*** é um plotter de sinal, onde dados na entrada, que podem consistir em qualquer número de canais, são apresentados;
- ***Display:*** exibe os valores dos *tokens* que chegam nos canais de entrada em uma área de texto na tela;
- ***Scale:*** produz um *token* de saída em cada disparo com um valor que é igual a

[4] Disponivel em: ptolemy.berkeley.edu/ptolemyII/ptII10.0/ptII10.0.1$_2$0141217/doc/

uma versão em escala da entrada;

- ***Expression:*** avalia uma expressão que pode incluir referências às entradas, hora atual e uma contagem do disparo;
- ***MultplyDivide:*** é um multiplicador e/ou divisor polimórfico.
- ***State:*** tem duas portas, uma para vincular as transições de entrada e a outra para as transições de saída.
- ***Transition:*** é uma ***relation***, com propriedades expressões esperam ser verdadeiras para prosseguir, parâmetros de saída e um conjunto de ações programadas para serem executadas durante a transição;
- ***Refinement:*** é ator de composição tipado que suporta o espelhamento de suas portas em seu contêiner (que deve ser um ModalModel), o que, por sua vez, garante o espelhamento de portas em cada um dos refinamentos e no controlador;
- ***AddSubtract:*** é um somador/subtrator polimórfico, onde os dados que chegam na porta de entrada denominada *plus* serão adicionados e os dados que chegam na porta de entrada denominada *minus* serão subtraídos.

As máquinas de estado finito no Ptolemy® seguem o mesmo padrão dos demais componentes, o que difere das máquinas de estado finito do Simulink®, que passa a utilizar agrupamentos de estados (*states*) dentro de seu escopo para determinar seu estado atual. Funções com linguagem de ação escritas em C (ou na própria linguagem do MATLAB®) definem a sintaxe para ações de estado e transição responsáveis por determinar o comportamento da máquina através das *transitions*. Os diagramas Stateflow® nos modelos Simulink® têm uma propriedade de linguagem de ação que define a sintaxe para ações de estado e transição.

Para a estrutura de uma máquina de estados, o Simulink® utiliza ***State*** para alterar entre dinâmica de tempo periódica e contínua. Em seu gráfico (*chart*), são usados estados para modelar um sistema dinâmico periódico ou contínuo combinado com lógica de comutação que usa transições, havendo a possibilidade de acessar entradas e saídas de um gráfico em cada estado.

Uma ***transition*** é a transição de uma linha com uma ponta de seta que liga um objeto gráfico a outro. Uma transição normalmente conecta uma origem e um objeto de destino. O objeto de origem é onde a transição começa e o objeto de destino é onde a

transição termina.

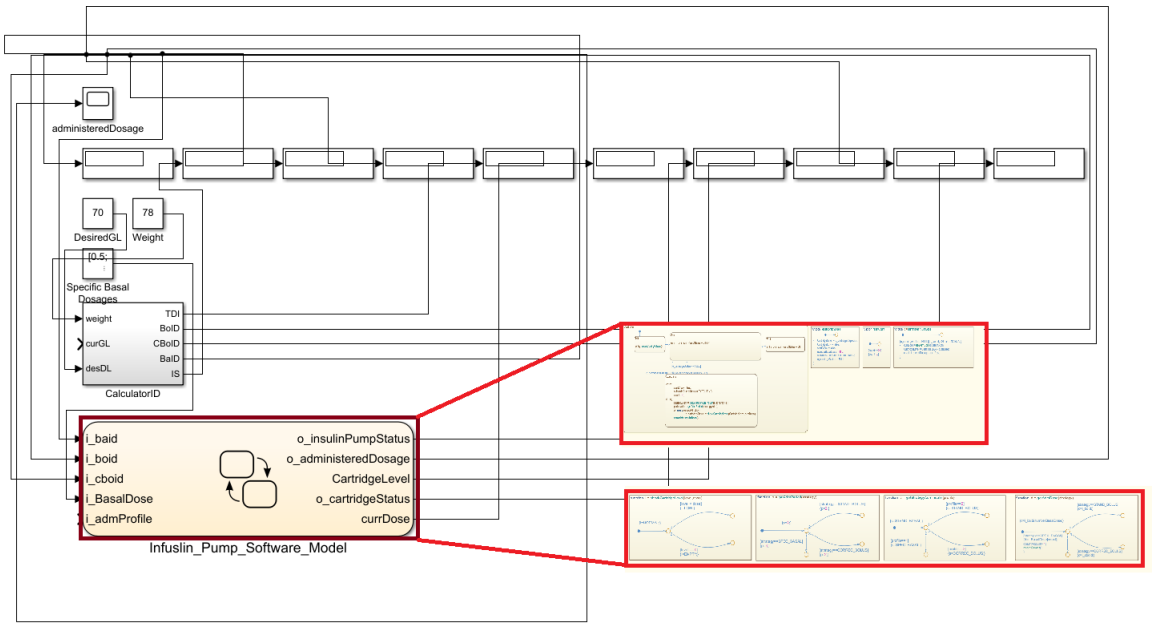
As ***junctions*** representam uma junção conectiva entre um ponto de decisão em um caminho de transição. É possível combinar transições e junções conectivas para criar caminhos de uma fonte comum para vários destinos ou de várias fontes para um destino comum.

A estrutura do Simulink® é definida por blocos (*Block*). As funções desses blocos são denominadas através de seu parâmetro *BlockType* (semelhantes às classes do Ptolemy II®) e um identificador denominado *Simulink Identifier*, usado para identificar um objeto enquanto ele estiver inserido no modelo, mesmo se seu nome for alterado. Por esse motivo, o identificador é um valor único e não modificável.

5.3.2 Execução da Pesquisa

Após a realização dos testes do *framework*, selecionando como modelo de entrada o arquivo XML do modelo de cenário clínico apresentado na subseção anterior, foi possível gerar o modelo MDL mostrado na **Figura 21**, contendo algumas restrições.

Figura 21 – Modelo da Bomba de Insulina no Simulink® a partir do Modelo no Ptolemy II®.



Fonte: Autoria própria (2021).

Alguns componentes não puderam ser mapeados, como a faixa de glicose que é disparada por um gatilho de acordo com o *"DiscretClock"* e o *"CurrentTime"*, já que esses

componentes seriam mapeados para o *"SignalBuilder"* do Simulink®, uma vez que esse componente é composto por uma série de caracteres de comprimento fixo, não podendo ser traduzidos por uma ferramenta fora do Simulink®. Esse empecilho pode ser facilmente configurado manualmente pelo projetista no Simulink®, tornando possível a execução do modelo pela ferramenta.

A organização do modelo destino quando gerado, também não está estruturada de forma esteticamente agradável, mas pode ser configurada apenas arrastando os componentes para posições mais adequadas, apesar de não interferir na execução pelo Simulink®.

5.3.3 Análise dos Dados

Levando em consideração que poucas alterações no modelo transformado foram feitas para a execução do mesmo pelo Simulink®, pode-se afirmar que, após os testes realizados, considerou-se o processo de transformação de modelo pela aplicação que faz uso do *framework* TADEM bem-sucedido, uma vez que 82,2% dos componentes e grupos de componentes puderam ser transformados, enquanto apenas 17,8% não puderam.

Esses componentes não mapeados são apresentados no **Apêndice B**, uma vez que não houve a possibilidade de ler um arquivo de valores separados por virgula (csv) no Simulink®. A notação da posição dos componentes são distintas entre as ferramentas, o que por si só não impediram o mapeamento, mas por conta da máquina de estados ser dependente da posição, e os componentes associados ao Ptolemy II® independerem de organização para seu funcionamento, não foi possível haver um mapeamento referente a isso. Também foram encontrados alguns componentes sem associação entre o Ptolemy II® e o Simulink®. Estes encontram-se listados na tabela de componentes (**Apêndice B**) sem um componente associado à segunda ferramenta. Um exemplo já citado anteriormente é o do *"SignalBuilder"* .

Para descrever o comportamento da aplicação do *framework* TADEM, uma vez criada sua arquitetura, pôde-se analisar e explicar o comportamento do objeto de estudo em termos dessa arquitetura. Com os experimentos, foi possível realizar a análise a seguir.

Com relação à complementação do modelo transformado, foi necessário realizar alguns aprimoramentos referentes à organização e estética do modelo, além do acréscimo de alguns componentes, como o *"SignalBuilder"*, para possibilitar a execução pela ferramenta de destino.

Para contemplar o processo de mapeamento com relação aos efeitos da simulação, tendo em vista que o objetivo desta pesquisa foi propor um mecanismo automático ou semiautomático para a transformação de modelos entre duas ferramentas de modelagem, e como o resultado obtido teve uma margem positiva, pode-se afirmar que o teste realizado obteve um bom resultado para os requisitos da pesquisa.

Outro ponto está ligado ao comportamento do *framework* TADEM caso haja alteração na arquitetura do modelo. Outros componentes, além dos utilizados no modelo de estudo dessa pesquisa, podem ser usados para modelagem de softwares na ferramenta Ptolemy II® para a criação de modelos de SMFC. Cabe ao desenvolvedor identificar quais componentes são adequados. Caso seja utilizada outra arquitetura que seja composta desses mesmos componentes mapeados para este modelo, ocorre o mesmo cenário mostrado neste teste. Por se tratar de uma primeira versão do *framework* TADEM, se outros componentes que não se fazem presentes na tabela de mapeamento forem utilizados, acontecerá uma transformação mais limitada. Nesse caso será necessário realizar uma adição manual dos componentes e seus relacionamentos não mapeados. Para que haja a possibilidade de uma conversão mais próxima da total do modelo de origem para o modelo de destino, será necessária a adição dos componentes faltosos na tabela de mapeamento, caso esses componentes equiparem-se em relação aos seus comportamentos em suas respectivas ferramentas.

Quando tratamos do comportamento do *framework* TADEM caso haja alteração no contexto do modelo, tem-se a mesma ideia do tópico anterior. Uma vez que um cenário desse sistema físico-cibernético se difere do cenário clínico, mas que utilize os mesmos componentes presentes na tabela de mapeamento, seria possível a realização da transformação.

É possível analisar a proposta do *framework* TADEM, caso haja inversão entre o modelo de entrada com o modelo de saída, de tal maneira que o modelo gerado pelo Simulink® seja dado como entrada. Para isso acontecer, há a necessidade de uma varredura de seus componentes, identificando o que faz parte do modelo e o que faz parte da ferramenta. Assim, é possível submeter cada componente à tabela de mapeamento (uma vez que a tabela tenha sido revisada) havendo a possibilidade de transformar esses componentes contidos no arquivo MDL do Simulink® em componentes XML do Ptolemy II®.

5.4 Considerações Finais do Capítulo

Neste capítulo foi observado como se deu a aplicação do *framework* TADEM, apresentando a base para sua concepção, mostrando as tecnologias utilizadas e entendendo como se deu o cenário de uso e abstração dos componentes. O *framework* TADEM também teve seu comportamento verificado, inserindo um modelo construído a partir da ferramenta Ptolemy II® para obtenção de um modelo para execução na ferramenta Simulink®.

6 CONCLUSÃO

Com a demanda de softwares computacionais de sistemas físicos cibernéticos que necessitam de validação e testes antes de serem aplicados na prática, em especial para sistemas que envolvem cenários clínicos, há a responsabilidade, por parte dos desenvolvedores de sistemas, de entregar a melhor experiência para os usuários.

Diminuir o esforço e o custo do desenvolvimento de sistemas é uma tarefa não trivial e bastante útil nos tempos atuais, onde a constante evolução das tecnologias traz uma gama de oportunidades e ferramentas para esse fim.

Neste trabalho, foi discutida uma solução para trazer um novo mecanismo de transformação automática de componentes presentes no modelo de simulação do cenário clínico gerado a partir da ferramenta Ptolemy II®, para os componentes do Simulink®. Pôde-se garantir que o protótipo do *framework* proposto foi automático e fornece uma base estrutural suficiente para servir de solução como transformação de modelos para diferentes cenários clínicos.

A solução se deu na concepção de uma proposta inicial do *framework* TADEM. Para melhorar sua usabilidade, pretende-se incorporar novos componentes, incluir outras ferramentas e a geração de um documento formal com a especificação dos elementos no arquivo gerado, como discutido na seção de trabalhos futuros deste capítulo.

6.1 Limitações

A solução apresentada neste trabalho está restrita a duas ferramentas de modelagem que estão há muito tempo no mercado e em constante evolução, o que pode acarretar em problemas de compatibilidade com versões diferentes das adotadas neste trabalho de tais ferramentas. Dessa forma, é necessária uma constante atualização e evolução do próprio *framework*. Também há a questão de componentes que são impossibilitados de se mapear por conta de alguma propriedade da própria ferramenta, dificultando a execução dos modelos por outras ferramentas, sejam elas para transformação ou qualquer outro propósito. Com isso, o uso do *framework* não garante a transformação total do modelo, ou seja, o mapeamento completo dos componentes do modelo de origem para componentes do modelo de destino. Portanto, em geral, torna-se necessário que o utilizador do *framework* complemente o modelo destino para garantir a total compatibilidade entre os modelos e,

principalmente, suas dinâmicas comportamentais.

6.2 Trabalhos Futuros

Há algumas ideias a serem aplicadas no TADEM para entregar a melhor funcionalidade e experiência para quem for desenvolver um software de modelagem de sistemas físicos cibernéticos.

Uma delas é a de utilizar a tabela de mapeamento alocada diretamente no modelo independente de plataforma em formato ontológico, pois a ontologia é utilizada para representar um conjunto de conceitos de um domínio e os relacionamentos entre eles de maneira a realizar inferências sobre esses domínios, sendo assim possível utilizá-la como uma ferramenta de modelagem que estaria intrínseco o mapeamento entre diversas ferramentas, não só as estudadas durante a realização deste trabalho.

Para os próximos passos, busca-se primeiro retirar o mapeamento dos componentes que estão contidos em funções no código desenvolvido, para um local semelhante a um banco de dados, ou um banco de dados propriamente dito, onde seria possível adicionar novos mapeamentos para contemplar o maior número de componentes possíveis entre as diferentes ferramentas de modelagem. Esse passo facilita a concepção de um documento formal com a especificação dos elementos gerados e os que não foram contempladas pela ferramenta.

REFERÊNCIAS

ATKINSON, C.; KUHNE, T. Model-driven development: a metamodeling foundation. **IEEE software**, IEEE, v. 20, n. 5, p. 36–41, 2003.

BAHETI, R.; GILL, H. Cyber-physical systems. **The impact of control technology**, v. 12, n. 1, p. 161–166, 2011.

BECKETT, D.; BERNERS-LEE, T.; PRUD'HOMMEAUX, E.; CAROTHERS, G. RDF 1.1 Turtle: Terse RDF Triple Language. W3C Recommendation 25 February 2014. **World Wide Web Consortium**, 2014. Disponível em: <http://www.w3.org/TR/turtle>.

BéVAN, R.; BERRUET, P.; LAMOTTE, F. de; ADAM, M.; CARDIN, O.; CASTAGNA, P. Generation of multiplatform control for transitic systems using a component-based approach. In: **Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)**. Krakow, Poland: IEEE, 2012. p. 1–8.

COSTA, A.; CAVALHEIRO, S.; FOSS, L.; RIBEIRO, L. From uml diagrams to simulink models: a precise and verified translation. In: **Proceedings of the 30th Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2015. p. 1547–1552. ISBN 9781450331968. Disponível em: <https://doi.org/10.1145/2695664.2695869>.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Sistemas Distribuídos: Conceitos e Projeto**. Porto Alegre, RS, Brasil: Bookman Editora, 2013.

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

DABNEY, J. B.; HARMAN, T. L. **Mastering simulink**. Upper Saddle River, New Jersey: Pearson, 2004.

DERLER, P.; NADERLINGER, A.; PREE, W.; RESMERITA, S.; TEMPL, J. Simulation of let models in simulink and ptolemy. In: SPRINGER. **Monterey Workshop 2008: Foundations of Computer Software. Future Trends and Techniques for Development**. Budapest, Hungary, 2008. p. 83–92.

EKER, J.; JANNECK, J. W.; LEE, E. A.; LIU, J.; LIU, X.; LUDVIG, J.; NEUENDORFFER, S.; SACHS, S.; XIONG, Y. Taming heterogeneity-the ptolemy approach. **Proceedings of the IEEE**, IEEE, v. 91, n. 1, p. 127–144, 2003.

EUZENAT, J.; SHVAIKO, P. *et al.* **Ontology matching**. [S.l.]: Springer, 2007. v. 18.

GILL, H. Nsf perspective and status on cyber-physical systems. **Austin. Internet: http://varma. ece. cmu. edu/CPS/Presentations/gill. pdf [zuletzt aufgesucht am 1.04. 2015]**, 2006.

GÜRDÜR, D.; ASPLUND, F.; EL-KHOURY, J. Measuring tool chain interoperability in cyber-physical systems. In: IEEE. **2016 11th System of Systems Engineering Conference (SoSE)**. Kongsberg, Norway, 2016. p. 1–4.

HU, J.; HUANG, L.; CAO, B.; CHANG, X. Executable modeling approach to service oriented architecture using soaml in conjunction with extended devsml. In: IEEE. **2014 IEEE International Conference on Services Computing**. NW, Washington, 2014. p. 243–250.

HUANG, J.-Y.; LANGE, C.; AUER, S. Streaming transformation of xml to rdf using xpath-based mappings. In: ASSOCIATION FOR COMPUTING MACHINERY. **Proceedings of the 11th International Conference on Semantic Systems**. San Diego, CA, 2015. p. 129–136.

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.

JOUAULT, F.; BÉZIVIN, J. Km3: a dsl for metamodel specification. In: SPRINGER. **International Conference on Formal Methods for Open Object-Based Distributed Systems**. Bologna, Italy: Springer, 2006. p. 171–185.

KERN, H.; STEFAN, F.; DIMITRIESKI, V.; ČELIKOVIĆ, M. Mapping-based exchange of models between meta-modeling tools. In: **Proceedings of the 14th Workshop on Domain-Specific Modeling**. New York, USA: Association for Computing Machinery, 2014. p. 29–34.

KHERRAF, S.; LEFEBVRE, É.; SURYN, W. Transformation from cim to pim using patterns and archetypes. In: IEEE. **19th Australian Conference on Software Engineering**. Adelaide, Australia, 2008. p. 338–346.

KLEE, H.; ALLEN, R. **Simulation of dynamic systems with MATLAB® and Simulink®**. Florida: Crc Press, 2018.

KOBEISSY, N.; GENET, M. G.; ZEGHLACHE, D. Mapping xml to owl for seamless information retrieval in context-aware environments. In: IEEE. **IEEE International Conference on Pervasive Services**. Istanbul, 2007. p. 361–366.

LASALLE, J.; BOUQUET, F.; LEGEARD, B.; PEUREUX, F. Sysml to uml model transformation for test generation purpose. **ACM SIGSOFT Software Engineering Notes**, ACM New York, NY, USA, v. 36, n. 1, p. 1–8, 2011.

LEE, E. A. The past, present and future of cyber-physical systems: A focus on models. **Sensors**, v. 15, n. 3, p. 4837–4869, 2015. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/15/3/4837>.

LEE, I.; SOKOLSKY, O. Medical cyber physical systems. In: IEEE. **Design automation conference**. California, 2010. p. 743–748.

LEE, I.; SOKOLSKY, O.; CHEN, S.; HATCLIFF, J.; JEE, E.; KIM, B.; KING, A.; MULLEN-FORTINO, M.; PARK, S.; ROEDERER, A. *et al.* Challenges and research directions in medical cyber–physical systems. **Proceedings of the IEEE**, IEEE, v. 100, n. 1, p. 75–90, 2011.

LI, R.; ZHOU, R.; LI, G.; HE, W.; ZHANG, X.; KOO, T. J. A prototype of model-based design tool and its application in the development process of electronic control unit. In: IEEE. **2011 IEEE 35th Annual Computer Software and Applications Conference Workshops**. Munich, Germany, 2011. p. 236–242.

MELO, T.; SILVA, L.; PERKUSICH, A.; SILVA, J.; NETO, J. da R. Software environments as learning tools for modeling engineering systems. In: **Proceedings of the 7th International Conference on Computer Supported Education-Volume 2**. Lisbon,Portugal: Association for Computing Machinery, 2015. p. 224–231.

PASSARINI, R. F.; BECKER, L. B.; FARINES, J.-M. The assisted transformation of models: Supporting cyber-physical systems design by extracting architectural aspects and operating modes from simulink functional models. In: IEEE. **2013 III Brazilian Symposium on Computing Systems Engineering**. Brazil, 2013. p. 47–52.

PASSARINI, R. F.; FARINES, J.-M.; FERNANDES, J. M.; BECKER, L. B. Cyber-physical systems design: transition from functional to architectural models. **Design Automation for Embedded Systems**, Springer, v. 19, n. 4, p. 345–366, 2015.

PASTOR, O.; ESPAÑA, S.; PANACH, J. I.; AQUINO, N. Model-driven development. **Informatik-Spektrum**, Springer, Deutschland, v. 31, n. 5, p. 394–407, 2008.

POOVENDRAN, R.; SAMPIGETHAYA, K.; GUPTA, S. K. S.; LEE, I.; PRASAD, K. V.; CORMAN, D.; PAUNICKA, J. L. Special issue on cyber-physical systems [scanning the issue]. **Proceedings of the IEEE**, IEEE, v. 100, n. 1, p. 6–12, 2011.

SANTOS, D. E. dos; SOUZA, I. T. de; CAMARGO, T. Metodologias ágeis para o desenvolvimento de software: Aplicação e o uso da metodologia scrum em contraste ao modelo tradicional de gerenciamento de projetos. **Revista Computação Aplicada-UNG-Ser**, v. 2, n. 1, p. 39–46, 2013.

SCHEINER, G.; BOYER, B. A. Characteristics of basal insulin requirements by age and gender in type-1 diabetes patients using insulin pump therapy. **Diabetes research and clinical practice**, Elsevier, USA, v. 69, n. 1, p. 14–21, 2005.

SILVA, L. C.; ALMEIDA, H. O.; PERKUSICH, A.; PERKUSICH, M. A model-based approach to support validation of medical cyber-physical systems. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 15, n. 11, p. 27625–27670, 2015.

SILVA, L. C. *et al.* Uma abordagem baseada em modelos para suporte à validação de sistemas médicos físico-cibernéticos. Universidade Federal de Campina Grande, 2015.

SILVA, L. C.; PERKUSICH, M.; BUBLITZ, F. M.; ALMEIDA, H. O.; PERKUSICH, A. A model-based architecture for testing medical cyber-physical systems. In: ASSOCIATION FOR COMPUTING MACHINERY. **Proceedings of the 29th Annual ACM Symposium on Applied Computing**. Gyeongju Republic of Korea, 2014. p. 25–30.

SON, H. S.; KIM, W. Y.; KIM, R. Y.; MIN, H.-G. Metamodel design for model transformation from simulink to ecml in cyber physical systems. In: **Computer Applications for Graphics, Grid Computing, and Industrial Environment**. Angneug, Korea: Springer, 2012. p. 56–60.

TENÓRIO, L. E. F. **Transformormlets: um framework para construção de transformadores de modelos mda**. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2004.

TUAN, L. A.; ZHENG, M. C.; THO, Q. T. Modeling and verification of safety critical systems: A case study on pacemaker. In: IEEE. **2010 Fourth International Conference on Secure Software Integration and Reliability Improvement**. Singapore: IEEE, 2010. p. 23–32.

WHO. **Vision impairment and blindness, Fact Sheet N^{o}282**. 2020. <http://www.who.int>, Last accessed on 2021-01-20.

WIERINGA, R. Design science as nested problem solving. In: ASSOCIATION FOR COMPUTING MACHINERY. **Proceedings of the 4th international conference on design science research in information systems and technology**. Philadelphia, Pennsylvania, 2009. p. 8.

WIERINGA, R. J. **Design Science Methodology for Information Systems and Software Engineering**. Philadelphia, Pennsylvania: Springer, 2014.

WOLF, W. Cyber-physical systems. **Computer**, IEEE Computer Society, v. 42, n. 03, p. 88–89, 2009.

XIAO, T.; FAN, W. Modeling and simulation framework for cyber physical systems. In: **Advanced Methods, Techniques, and Applications in Modeling and Simulation**. Korea: Springer, 2012. p. 105–115.

XIONG, Y.; LEET, E.; LIU, X.; ZHAO, Y.; ZHONG, L. C. The design and application of structured types in ptolemy ii. In: IEEE. **2005 IEEE International Conference on Granular Computing**. China, 2005. v. 2, p. 683–688.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. **IEEE Transactions on industrial informatics**, IEEE, China, v. 10, n. 4, p. 2233–2243, 2014.

ZHANG, L.; FENG, S. Integration design and model transformation for cyber physical systems. In: IEEE. **2014 IEEE 5th International Conference on Software Engineering and Service Science**. China, 2014. p. 754–757.

ZHAO, C.; YAN, H.; LIU, D.; ZHU, H.; WANG, Y.; CHEN, Y. Co-simulation research and application for active distribution network based on ptolemy ii and simulink. In: IEEE. **2014 China International Conference on Electricity Distribution (CICED)**. China, 2014. p. 1230–1235.

ZHOU, F.; TOP, S.; SIERSZECKI, K.; ANGELOV, C. Simulink analysis of component-based embedded applications. In: **Proceedings of the 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software**. Belgium: Association for Computing Machinery, 2010. p. 61–68.

APÊNDICE A - GRUPO DE COMPONENTES MAPEADOS

Simulink	Ptolemy II
Modelo Geral	
Display	MonitorValue
Constant	Const
Constant	Parameter
SubSystem	TypedCompositeActor
SubSystem.InsulinPump_Software	TypedCompositActor.InsulinPump
Scope	Display
Import	TypedIOPort
Outport	TypedIOPort
CalculatorID	
SubSystem	TypedCompositeActor
CalculatorID.CalculatorTDI	
Constant ProductDivide Rounding Output	Scale Round TypedIOPort
CalculatorID.CalculadorBoID	
Import Constant BusCreat Fcn Rounding	Input Const Expression Round
CalculatorID.CalculationIS	
Constant BusCreator Fcn Rounding	Expression Round
CalculatorID.CalculationCBoID	
BusCreate Fcn Switch Constant Rounding Outport	Expression Round.Round TypedIOPort
CalculatorID.CalculationBoID	
Gain	Scale
Constant	Const
ProductDivide	MultiplyDivide
Rounding	Round

Simulink	Ptolemy II
Insulin_Pump_Model	
Stateflow{machine}	ModalModel
State{OR_STATE}	ModalController
State{OR_STATE}	State
State{FUNC_STATE} Transition	Transition.CommitActionsAttribute
State{OR_STATE} Transition CONNECTIVE	Transition.StringAttribute
State{FUNC_STATE}	Refinement
State{FUNC_STATE} Transition CONNECTIVE	TypedCompositActor AddSubtract Expression Round Const TypedCompositActor.Const TypedCompositActor.Expression
State{FUNC_STATE} Transition CONNECTIVE	DiscretClock CurrentTime
State Transition CONNECTIVE	TimeDelay Const BooleanSwitch Counter Expression Parameter
State Transition CONNECTIVE	TimeDelay Const BooleanSwitch Counter Expression Parameter
State Transition CONNECTIVE	TimeDelay Const BooleanSwitch Counter Expression LogicGate Parameter
State Transition CONNECTIVE	TimeDelay Const BooleanSwitch Counter Const Parameter

Simulink	Ptolemy II
State (For Each) CONNECTIVE_JUNCTION Transition	BooleanSwitch Const Const(boolean)
	BooleanSwitch Const Limiter
	BooleanSwitch Const Expression Limiter
	BooleanSwitch Const Acumulator Delay Comparator Expression RecordUpdater
State{FUNC_STATE} Transition CONNECTIVE	Constante Expression AddSubtract Limiter
State{FUNC_STATE} Transition CONNECTIVE	TypedCompositActor.Const Const
State{FUNC_STATE} Transition CONNECTIVE	Const Parameter

APÊNDICE B - GRUPO DE COMPONENTES NÃO MAPEADOS

Simulink	Ptolemy II
Modelo Geral	
SignalBuilder	CurrentTime DiscretClock Const IntRangeParameter Round
Constant	csv File
SignalBuilder.Administrator	Const
Positions	Positions
	Safety Property
CalculatorID	
CalculatorID.CalculatorTDI	
CalculatorID.CalculadorBoID	
CalculatorID.CalculationIS	
CalculatorID.CalculationCBoID	
CalculatorID.CalculationBoID	
Insulin_Pump_Model	
SignalBuilder	Parameter
	Discard
	ElementsToArray