Universidade Estadual de Londrina

MÁRCIO DE ABREU MOREIRA

DESENVOLVIMENTO DE AMBIENTES VIRTUAIS ORIENTADO POR MODELOS

Londrina
2015

MÁRCIO DE ABREU MOREIRA

DESENVOLVIMENTO DE AMBIENTES VIRTUAIS ORIENTADO POR MODELOS

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação do Departamento de Computação da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Profª. Drª. Jandira Guenka Palma.

Londrina
2015

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Moreira, Márcio de Abreu .

Desenvolvimento de Ambientes Virtuais Orientado por Modelos / Márcio de Abreu Moreira. - Londrina, 2015.

92 f. : il.

Orientador: Jandira Guenka Palma.

Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2015.

Inclui bibliografia.

1. Engenharia de Software - Tese. 2. Desenvolvimento Orientado a Modelo - Tese. 3. Arquitetura Orientada a Modelo - MDA - Tese. 4. Ambientes Virtuais - Tese. I. Palma, Jandira Guenka . II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

MÁRCIO DE ABREU MOREIRA

DESENVOLVIMENTO DE AMBIENTES VIRTUAIS ORIENTADO POR MODELOS

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação do Departamento de Computação da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

_____

Orientadora: Profª. Drª. Jandira Guenka Palma
Universidade Estadual de Londrina – UEL

_____

Prof. Dr. Arthur Jose Vieira Porto
Universidade de São Paulo - USP

_____

Prof. Dr. Claiton de Oliveira
Universidade Tecnológica Federal do Paraná - UTFPR

_____

Prof. Dr. Evandro Baccarin
Universidade Estadual de Londrina - UEL

Londrina, 09 de Junho de 2015.

Este trabalho é dedicado à minha esposa, meu filho, meus pai, aos amigos e professores que me apoiaram em mais esta etapa da minha vida.

AGRADECIMENTOS

Agradeço a Deus que me permitiu chegar até aqui, a minha família pelo apoio incondicional em todos os momentos, minha Professora Orientadora Jandira Guenka Palma pela oportunidade dada e por acreditar em mim para o desenvolvimento deste trabalho. Obrigado a todos os meus professores do programa de mestrado em Ciência da Computação que contribuíram na evolução do meu conhecimento. Obrigado ao meu colega acadêmico e amigo Wilson Hissamu Shirado, por todo apoio dado no decorrer desta caminhada, esta conquista não é somente minha, é nossa. Obrigado aos alunos do curso de graduação em Ciência da Computação que colaboraram com este trabalho. Obrigado a todos aqueles que contribuíram de alguma maneira para que este trabalho tornasse realidade.

Agradecimento especial à coordenação e direção do Departamento de Computação da Universidade Estadual de Londrina que me acolheram como aluno, a CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pela concessão da bolsa durante todo o período de realização deste mestrado.

“O Senhor é o meu pastor, nada me faltará.”

(Bíblia Sagrada, Salmos 23:1).

MORIERA, M. de A. **Desenvolvimento de ambientes virtuais orientado por modelos**. 2015. 92p. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2015.

RESUMO

O desenvolvimento de Ambiente Virtual não é uma tarefa trivial, pois tais sistemas são complexos e gerados dentro de um processo clássico de desenvolvimento de software, composto pelas fases de análise, planejamento, codificação, implantação, teste e manutenção. As fases bem definidas estão presentes em empresas que adotam critérios de qualidade e/ou tenha processos de desenvolvimento definido. No entanto, muitas empresas não possuem processo de desenvolvimento definido, e as que têm normalmente encontram dificuldades em produzir documentos com alto valor agregado para a manutenção, atualização e reutilização, em razão de uma comunicação restrita por falta de conhecimento, de maturidade e principalmente de rastreabilidade desde o requisito ao código e vice versa, assim geram uma coleção de documentos que não correspondem de forma fidedigna ao código implementado e nem são atualizados regularmente. Estes problemas podem ser minimizados com uma abordagem do paradigma de MDD (*Model-Driven Development*), este paradigma tem o desenvolvimento dirigido por modelos, ou seja, o objetivo principal não é desenvolvimento orientado para geração de código fonte para a criação do produto software, mas para o desenvolvimento dos modelos que o representam. Portanto, o paradigma requer a criação de modelos completos e consistentes, pois estes modelos são os principais artefatos do desenvolvimento, transformados automaticamente (ou semiautomático) em código por meio de ferramentas de modelagem e transformação de modelos. Neste trabalho é abordado o desenvolvimento de Ambientes Virtuais orientado por modelo. Para isto é proposto um metamodelo que utiliza a abordagem *Model-Driven Architecture*-MDA com seus diferentes níveis de abstração para gerar os ambientes e seus elementos por meio dos modelos construídos com a Linguagem de Modelagem UML, (*Unified Modeling Language*-UML) e Redes de Petri. Para validar o metamodelo a proposta é aplicada no desenvolvimento de um ambiente virtual, especificamente um ambiente de Realidade Aumenta.

Palavras-chave: Engenharia de software. Desenvolvimento drientado a modelo. MDD. MDA. Ambientes virtuais. Simulação de ambientes. Realidade virtual. Realidade aumentada.

MORIERA, M. de A. **Development of virtual environments model-oriented**. 2015. 92p. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2015.

ABSTRACT

The development of virtual environment is not a trivial task, since such systems are complex and generated in a classic process of software development, which comprises the stages of analysis, design, coding, implementation, testing and maintenance. The well-defined phases are present in a company that adopt quality criteria and / or has a defined process. However, many companies do not have a defined development process, and those that have usually find it difficult to produce documents with high added value for maintenance, update and reuse, because of a restricted communication for lack of knowledge, maturity and especially traceability from requirements to code and vice versa, thus generate a collection of documents that do not correspond reliably implemented the code, nor are updated regularly. These problems can be minimized with an MDD (Model-Driven Development) paradigm approach, this paradigm has models driving the development, ie, the main objective is not development-oriented source code generation for product creation software, but for the development of models that represent it. Therefore, the paradigm requires the creation of complete and consistent models because these models are the main developmental artifacts, processed automatically (or semi) coded by modeling tools and transformation of models. This essay discusses the development of Virtual Environments driven model. For this is proposed a metamodel that uses the Model-Driven Architecture, MDA approach with different levels of abstraction to generate the environment and its elements through models built with UML Modeling Language (Unified Modeling Language-UML) and Networks Petri. To validate the proposal metamodel is applied in developing a virtual environment, specifically a reality environment increases.

Keywords: Software engineering. Model-driven development. MDD. MDA. Virtual environments. Environment simulation. Virtual reality. Augmented reality.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

MDA Arquitetura Orientada a Modelo (*Model Driven Architect*)

MDD Desenvolvimento Orientado a Modelo (*Model Driven Development*)

VR *Virtual Reality*

RV Realidade Virtual

AR *Augmented Reality*

RA Realidade Aumentada

PN *Petri Net*

RdP Rede de Petri

W3C *World Wide Web Consortium*

XML Linguagem de Marcação Extensivel (*Extensible Markup Language*)

EAV Editor de Ambientes Virtuais

AI Inteligência Artificial (*Artificial Intelligence*)

API *Interface* de Programação de Aplicativos (*Application Programming Interface*)

IDE *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado

SUMÁRIO

INTRODUÇÃO

Ambientes que simulam a realidade como Realidade Virtual - RV e Realidade Amentada - RA aceleram a concretização de projetos em diversas áreas do conhecimento, bem como o estudo da viabilidade de novos projetos, pois analisa alternativas sem utilizar equipamentos reais em todo o estudo. Desta forma, os sistemas de simulação tornam-se cada vez mais importantes, uma vez que permitem que novas ideias sejam testadas, com a intensão de verificar a existência de problemas e corrigi-los com maior rapidez.

Sob o ponto de vista do desenvolvimento sistemas de simulação da realidade passam pelos mesmos processos de construção que qualquer outro tipo de software. Existem ferramentas de apoio específicas para geração de ambientes virtuais e objetos tridimensionais, como as APIs (*Application Programming Interface*) ARToolKit [1], NyarToolkit [2] e FLARToolkit [3], VirTraM [4], GIS2R [5] e *"Look!"* [6]. Porém, a maioria destas ferramentas não é fácil de ser utilizada, focam na resolução de problemas específicos de uma determinada área, para um determinado propósito ou ainda exige conhecimento especializado de uma linguagem de programação. Isto, na maioria das vezes, dificulta a reutilização de modelos de um projeto em novos projetos.

Aplicações de ambientes virtuais fornecem ao usuário a possibilidade de interação com o ambiente e seus elementos em tempo real, através da utilização de equipamentos especiais, como sensores, motores, marcações, óculos para visão tridimensional e outros. Para que a interação acontece também é necessário conhecimento a respeito do padrão de funcionamento e da programação destes equipamentos para que possam ser integrados ao sistema. Outras dificuldades estão relacionadas ao uso de técnicas de visão computacional, tratamento gráfico, identificação de deformação de objetos, colisões, monitoramento das interações dos usuários, tudo isto em tempo real. Estas características são deste tipo de *software* que tornam o processo de desenvolvimento complexo.

Com relação às etapas de desenvolvimento, uma atividade comum nos projetos de *software* é a modelagem do *software*. Através dos modelos o desenvolvedor procura representar o sistema que será construído através de abstrações, de acordo o levantamento e análise de requisitos. Os modelos criados também são utilizados para documentar o *software*, validar os requisitos, auxiliar na criação de casos de teste, facilitar a comunicação entre os membros da equipe desenvolvedora, servindo ainda como base principal para o desenvolvimento do *software* projetado. Porém, esta não é uma realidade muito presente nos projetos de *software*.

O fato é que muitas vezes os modelos criados são utilizados nas fases iniciais do desenvolvimento, mas à medida que o este avança, tais modelos caem no desuso e tornam-se obsoletos. Programadores que desenvolvem códigos manualmente, também realizam alterações diretamente no código e não atualizam os modelos, resultando assim em modelos que não refletem no código implementado. Estas são ações não aproveitam algumas vantagens da utilização de modelos direcionando o desenvolvimento. Vantagem como a reutilização de modelos em novos projetos, que pode proporcionar aumento de produtividade, apoiar a manutenções, além de gerar uma documentação de *software* condizente entre modelo e código. Outra vantagem na utilização de modelos é a portabilidade, uma vez que os modelos são representações independentes de uma linguagem específica.

Na busca por soluções que minimizem dificuldades no desenvolvimento de software e melhore a utilização e aplicação da modelagem a fim de se obter códigos que sejam reflexos de sua modelagem, surge o paradigma de Desenvolvimento Orientado a Modelos – MDD (*Model-Driven Development*) [7], também conhecido como MDE (*Model-Driven Enineering*) [8] ou MDSD (*Model-Driven Software Development*) [9]. O MDD valoriza a utilização do modelo para o desenvolvimento de *software*, tratando-o como parte integrante do *software*, não apenas uma referência. A ideia principal é a de que os engenheiros de software não interajam manualmente com o código, mas que se concentrem na criação de modelos de mais alto nível, independentes de plataforma. Estes modelos passam por ferramentas de transformação responsáveis por gerar automaticamente o código-fonte.

O efetivo emprego do paradigma de MDD se dá através da aplicação da abordagem de Arquitetura Orientada a Modelos – MDA (*Model-Driven Architecture*). MDA é uma abordagem padronizada e mantida pela OMG (*Object Management Group*) que é um consórcio internacional de empresas que definem e mantem padrões na área de Orientação a Objetos [10], [11].

A abordagem MDA descreve seus modelos através da Linguagem Unificada de Modelagem – UML (*Unified Modeling Language*), que também é a linguagem de modelagem padrão adotada pela OMG. Porém, existem outras formas de representar sistemas, como por exemplo, através modelos em Rede de Petri – RdP, uma linguagem formal de modelagem gráfica e matemática utilizada principalmente para representar e simular sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos ou estocásticos [12]. A RdP pode modelar e simular sistemas especialistas e aplicações de Realidade Virtual e Aumentada [13]. Existem trabalhos que combinam modelos UML e RdP para projetar *software* [14] [15] [16] [17].

Os modelos na abordagem MDA passam por transformações entre diferentes níveis de abstração, até que por fim há a geração automática de código. MDA pode ser aplicado para o desenvolvimento de qualquer tipo de sistema, de pequeno ao de grande porte, com muita ou pouca complexidade, inclusive para criação de ambientes de virtuais e de simulação.

Para que não ocorram problemas de interpretação de termos, entenda-se ao longo deste trabalho, ambientes virtuais ou de simulação, como toda representação do mundo real criada computacionalmente de forma artificial e tridimensional, com interatividade em tempo real, gerado por aplicações, *software* ou sistemas concebidos para este fim.

1.1 OBJETIVO

Com intuito de apoiar o desenvolvimento de ambientes virtuais e minimizar a complexidade na criação deste tipo de software, este trabalho tem como objetivo propor um metamodelo fundamentado nos conceitos e técnicas de desenvolvimento orientado a modelos, que utiliza a abordagem MDA, modelos UML e modelos de RdP.

Para concretização do objetivo principal se faz necessário alcançar alguns objetivos específicos como:

- Compreender os conceitos de MDD;
- Conhecer os diferentes níveis de abstração da MDA, suas transformações, modelos e códigos resultantes;
- Pesquisar ferramentas que suportam a abordagem MDA;
- Levantar trabalhos que utilizaram e/ou implementaram algum tipo de abordagem similar a desta proposta;
- Conhecer os conceitos de Redes de Petri;
- Compreender e Analisar os conceitos de criação e controle de Ambientes Virtuais;
- Criar modelos em UML e Rede de Petri para o desenvolvimento e controle de software;
- Apresentar o metamodelo proposto neste trabalho;
- Aplicar o metamodelo proposto na geração de um ambiente virtual.

1.2 JUSTIFICATIVA

A abordagem MDA automatiza a geração de códigos a partir da modelagem estrutural e comportamental do sistema, isto reduz a codificação manual, minimiza os riscos de falha humana na programação, acelera o desenvolvimento e a manutenção do software, além de proporcionar a reutilização dos modelos existentes em novos projetos, visto que, uma vez que a modelagem do software esteja construída, o código é automaticamente gerado [18], [19].

Assim, como existem ferramentas de apoio para criação de ambientes virtuais existem também ferramentas de apoio ao desenvolvimento de software, as chamadas ferramentas CASE (*Computer-Aided Software Engineering*). Estas ferramentas cobrem parcialmente ou totalmente as atividades de engenharia de software, como análise de requisitos, modelagem até a programação. Um bom suporte a estas atividades proporcionam uma redução de problemas no projeto e no desenvolvimento do software, além de viabilizar produtos de software com maior qualidade, dentro do prazo estipulado e dos custos estimados.

Outra vantagem da utilização deste tipo de ferramenta, principalmente as que possuem modelagem visual, é que o desenvolvedor projeta o software arrastando e soltando modelos gráficos na área reservada à criação de modelos, processo este mais intuitivo e amigável ao desenvolvedor.

Existem também as ferramentas que além da modelagem visual transformam automática seus modelos em código, como é o caso do Enterprise Architect (EA) da Sparx Systems [20] e o Rational Rose da IBM [21]. Estas são ferramentas que possuem características adequadas aos princípios de MDD e fornecem suporte ao desenvolvimento de software de acordo com a abordagem MDA.

1.3 ORGANIZAÇÃO DO TRABALHO

A organização dos conteúdos necessários para execução deste trabalho está dividida por capítulos e seções. O primeiro capítulo introduz o assunto tratado, contextualiza e apresenta as causas que motivaram a elaboração deste trabalho assim como a proposta de solução desenvolvida. Este capítulo introdutório também apresenta os objetivos em seguida a justificativa deste trabalho.

O segundo capítulo apresenta todo referencial teórico necessário para realização deste trabalho, como conceitos de modelagem, linguagens de modelagem, MDD e MDA. Além dos conceitos relacionados à engenharia de software, os conceitos sobre ambientes virtuais, especificamente RA e RV e como os softwares voltados a esses temas são desenvolvidos atualmente. Após o embasamento teórico, o terceiro capítulo apresenta a proposta deste trabalho. Nele são detalhados etapa a etapa os procedimentos necessários para o desenvolvimento de software conforme os princípios de MDD apoiado pela abordagem de MDA, utilizando uma combinação de modelos UML e modelos Rede de Petri. Em seguida, no quarto capítulo, a abordagem proposta é aplicação na geração de um ambiente virtual, experimentando na prática os conceitos apresentados. Por fim, no quinto capítulo são apresentados os resultados, conclusões e proposta de trabalhos futuros.

2 LEVANTAMENTO BIBLIOGRÁFICO

O presente trabalho trata do desenvolvimento de ambientes virtuais a partir da transformação de modelos. Tendo isso em vista, é importante entender os conceitos de modelos e linguagens de modelagem para criação de modelos que são utilizados neste trabalho, como a UML e a RdP. Bem como a linguagem de marcação XML utilizada para o intercâmbio de dados entre diferentes ferramentas e sistemas de informação, e que neste trabalho é utilizada para representar textualmente os modelos UML e RdP.

Como o tema central do trabalho trata do desenvolvimento orientado a modelos, é crucial conhecer e compreender os conceitos do paradigma MDD, assim como a abordagem MDA responsável pelas transformações dos modelos, e que é parte do paradigma MDD. Tendo em vista que a abordagem de MDA pode ser aplicada para o desenvolvimento de ambientes virtual e aumentado, também são apresentados neste levantamento bibliográfico os conceitos sobre Ambientes Virtuais, especificamente sobre Realidade Virtual-RV e Realidade Aumentada-RA, com o intuito de melhor compreender o funcionamento e a estrutura necessária para criar e executar este tipo de aplicação, além de entender como este tipo de *software* é desenvolvido atualmente.

2.1 Modelagem

A atividade de modelagem é a ação de criar modelos, dar forma e contorno à imaginação ou então imitar algo que seja real. Na modelagem de *software* modelos são construídos para explicar as características ou o comportamento do mesmo. Através de modelos é possível identificar características e funcionalidades do *software*, além de servir de documentação para o planejamento e desenvolvimento [22].

Para Sampaio (1998), um modelo pode ser visto como um novo mundo construído para representar fatos/eventos/objetos/processos que acontecem no mundo real ou em um mundo imaginário [23]. Mellar et.al. (1994), enfatiza que o modelo cristaliza, momentaneamente, as relações causais através de uma topologia específica, passando a existir como um mundo artificial [24]. Segundo Santos at.al. (2003), o mundo artificial criado, transforma-se num objeto de análise que estará disponível para exploração, e mesmo modificação [25].

Na abordagem MDA os modelos são os principais artefatos para o processo de desenvolvimento de *software*. Em MDA, um *software* pode ser representado por vários modelos em níveis de abstrações diferentes.

2.2 Linguagens de Modelagem

A formalização de modelos é feita através das linguagens de modelagem, que por sua vez são definidas por um metamodelo [26], como a UML. Metamodelo é também um modelo que formaliza os conceitos fornecidos pela linguagem de modelagem, e define como ela é estruturada. A OMG adotou como linguagem de modelagem padrão a UML, que é baseada no *Meta Object Facility* (MOF). Mas existem outras linguagens para representação de sistemas, como a RdP [12].

A RdP é uma linguagem gráfica e matemática para modelagem de sistemas, muito indicada para modelar sistemas com alto índice de paralelismo, concorrência, entre outros [18]. Neste trabalho as linguagens de modelagem UML e RdP serão aplicadas na abordagem MDA de forma conjunta para geração de ambientes virtuais, a UML representa os modelos estruturais e a RdP os modelos comportamentais.

Para entender melhor cada uma das linguagens utilizadas neste trabalho a seguir são apresentados os conceitos teóricos de ambas.

2.2.1 UML

UML é uma linguagem de modelagem usada para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos [22]. A versão inicial da UML surgiu em 1996, originada da união dos principais métodos de modelagem orientada a objeto da época, método Booch de Grady Booch, o método OMT (*Object Modeling Technique*) de James Rumbaugh e o método OOSE (*Object-Oriented Software Engineering*) de Ivar Jacobson [27] [22]. O método Booch é voltado às fases de projeto de construção de sistemas, o método OOSE aborda o levantamento de requisitos, análise e projeto de alto nível e o OMT abrange desde a etapa de análise, passando pelo projeto, até a implementação do software. O esforço para criação de uma linguagem de modelagem padrão teve inicio com a unificação do método Booch com o método OMT, o que resultou no lançamento do Método Unificado no fim de 1995. Em 1996 Booch, Rumbaugh e

Jacobson, lançam a primeira versão da UML 0.9 [28]. Em 1996 a OMG lança uma RFP (*Request for Proposals*), na qual solicitava participação de empresas para contribuírem com a evolução da UML, isto resultou no inicio de 1997 na versão 1.1 da UML. A Figura 1 apresenta evolução cronológica da UML.

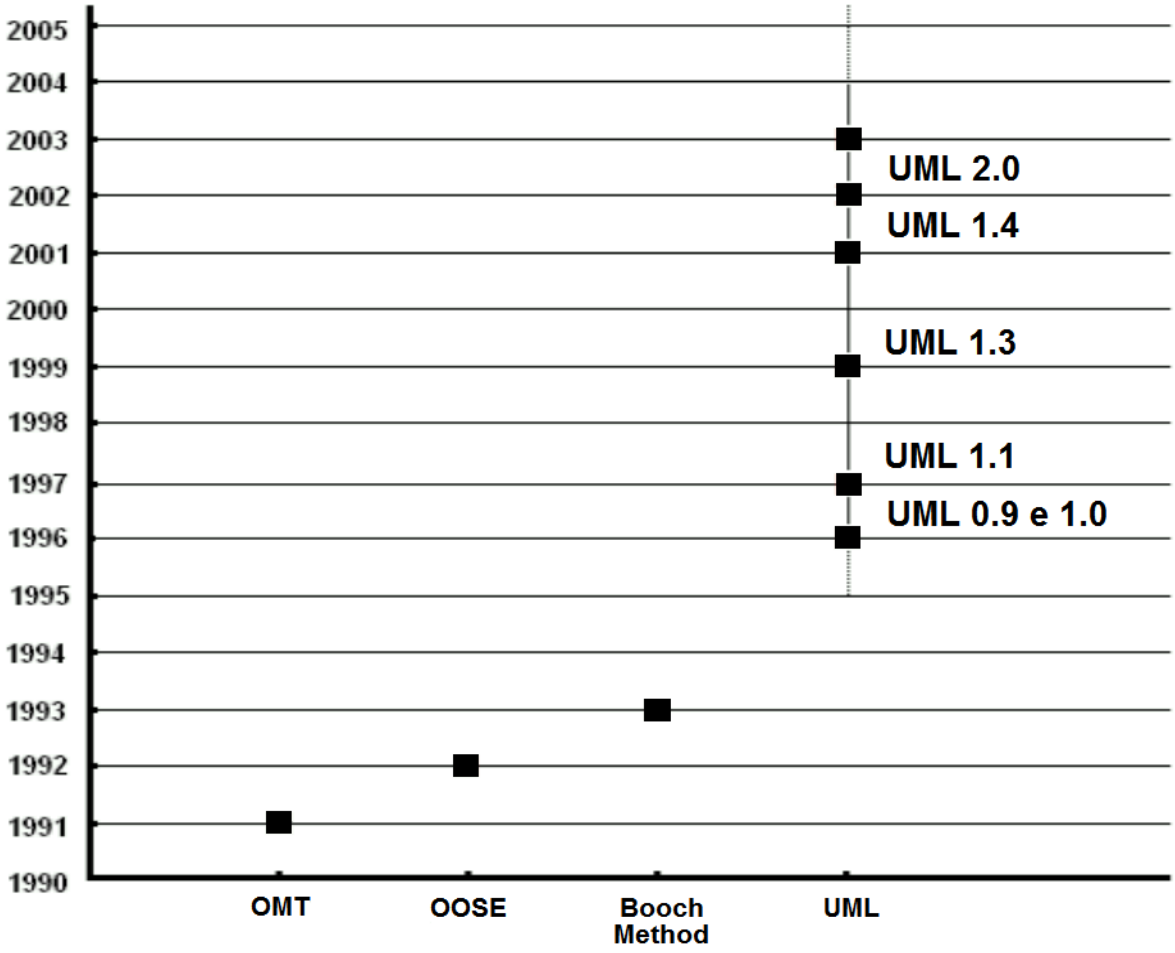


Figura 1- Cronologia da UML [22].

Atualmente toda documentação da UML é mantida e controlada pela OMG que a adotou como linguagem de modelagem padrão.

2.2.2 Redes de Petri - Rdp

Esta proposta também utiliza Redes de Petri (RdP) para criação de modelos, o que torna necessário se apropriar dos conceitos relacionados à RdP para que seja possível a sua utilização de maneira correta e eficaz, pois os modelos de RdP são aplicados neste trabalho para controlar o comportamento dos elementos do ambiente. Outro fator importante é

entender de que forma pode-se utilizar a modelagem de RdP em conjunto com a modelagem de UML.

Segundo Murata, Rede de Petri é um método de modelagem gráfica e matemática que pode ser aplicada a diversos sistemas, sendo uma ferramenta indicada para descrever e estudar informações processadas em sistemas caracterizados como concorrentes, assíncronos, distribuídos, paralelos e/ou estocásticos [12].

A RdP oferece um ambiente uniforme para modelagem, análise formal e simulação de sistemas a eventos discretos, permitindo uma visualização simultânea da sua estrutura e de seu comportamento [29]. Através da RdP é possível analisar aspectos da rede por meio de seus eventos e condições, bem como, as relações entre eles. Segundo esta caracterização, em cada estado da rede verificam-se determinadas condições, estas condições podem possibilitar a ocorrência de eventos que por sua vez podem ocasionar a mudança de estado da rede [30].

A RdP é composta por quatro elementos básicos, sendo eles:

- Lugar ou *Place*;
- Transição ou *Transition*;
- Arco ou *Arc*;
- Marca ou *Token*.

Consiste num grafo direcionado, com peso e bipartido, composto por dois elementos estruturais: lugares (*places*) e transições (*trasintions*). O *place* é representado graficamente por um círculo (Figura 2a) e a *transiton* por uma barra (Figura 2b). Os elementos estruturais são utilizados para criar o modelo, no qual arcos (*arcs*) orientados (Figura 2c) conectam *places* a *transitions* e *transitions* a *places*. Estes *arcs* podem ser rotulados com um valor inteiro positivo, indicando o peso do *arc*. Um *arc* de peso *k* pode ser interpretado como *k arc* paralelos de peso unitário.

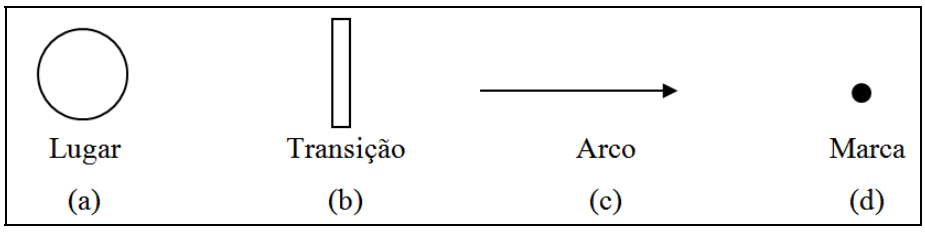


Figura 2 - Elementos da Rede de Petri [31].

Na modelagem de Redes de Petri, utiliza-se o conceito de condições e eventos, *places* representam condições, e *transitions* representam eventos. A presença de um

token (Figura 2d) em um determinado *place* é interpretado como condição verdadeira associada ao *place*. Outra interpretação, *k tokens* são colocadas em um *place* indicando que *k* dados ou recursos estão disponíveis. Uma *transition* (evento) tem certo número de *places* de entrada e saída que representam as pré-condições e pós-condições de eventos, respectivamente. Os *arcs* fazem a ligação dos *places* de entrada/pré-condição com as *transitions* e das *transitions* com os *places* de saída/pós-condição.

2.2.2.1 Definição formal das RdP

As RdPs podem ser analisadas segundo critérios matemáticos bem definidos, sendo formalmente definida como uma quíntupla $PN = (P, T, F, W, M_0)$, segundo [12]:

a. $P = \{p_1, p_2, \ldots, p_m\}$ é um conjunto finito de lugares,

b. $T = \{t_1, t_2, \ldots, t_n\}$ é um conjunto finito de transições,

c. $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos (relação de fluxo),

d. $W : F \rightarrow \{1,2,3,\ldots\}$ é a função peso,

e. $M_0 : P \rightarrow \{0,1,2,3,\ldots\}$ é a marcação inicial (quantidade de marcas em cada lugar),

f. $P \cap T = \phi$, $P \cup T \neq \phi$,

A estrutura da rede de Petri $N = (P, T, F, W)$ sem qualquer marcação inicial específica é denotada por N. Uma rede de Petri com uma dada marcação inicial é denotada por (N, M_0).

Além disto, denomina-se:

a. $M(p_i)$ é a marcação $\in \mathbf{N}$ em p_i,

b. $\bullet p = \{t \mid (t,p) \in F\}$ é o conjunto de transições de entrada de p,

c. $p\bullet = \{t \mid (p,t) \in F\}$ é o conjunto de transições de saída de p,

d. $\bullet t = \{p \mid (p,t) \in F\}$ é o conjunto de lugares de entrada de t,

e. $t\bullet = \{p \mid (t,p) \in F\}$ é o conjunto de lugares de saída de t,

f. $R(N, M_0)$ ou simplesmente $R(M_0)$ é o conjunto de todas possíveis marcações alcançáveis a partir de M_0 e

g. $L(N,M_0)$ ou simplesmente $L(M_0)$ é o conjunto de todas possíveis sequências de disparos de transições a partir de M_0.

Cada lugar pode possuir marcas, indicando um estado. A marcação do sistema (estado) é denotada por um vetor u de dimensão igual ao número de lugares do modelo. O *p*-ésimo componente de u, indicado por $M(p)$consiste no número de marcas do lugar p. O vetor M_0 consiste no vetor de marcação inicial, isto é, estado inicial do sistema.

O comportamento de diversos sistemas pode ser descrito em temos de seus estados e de sua respectiva mudança ou transição para outros possíveis estados. Na RdP, a mudança de estado ocorre de acordo com a regra de habilitação e disparo das transições. A ocorrência de um evento é denominada disparo de transição.

Uma transição está habilitada para disparar se para todos os lugares de entrada da transição, possuir um número de marcas maior ou igual ao peso do arco que conecta lugar a transições. Uma transição habilitada pode disparar ou não, dependendo se o evento realmente ocorrer; no disparo de uma transição habilitada, todo lugar que possui um arco para a transição tem seu número de marcas reduzidas pelo valor do peso deste arco, e todo lugar que possui um arco procedente da transição tem seu número de marcas acrescido do valor do peso deste arco.

Uma transição sem lugares de entrada é denominada transição fonte e a sem lugares de saída é chamada de transição sumidouro. Uma transição fonte está sempre habilitada, e o disparo de uma transição sumidouro consome marcas, mas não produz nenhuma.

Com relação à representação textual do modelo de RdP, nesta proposta será adotado formato padrão *Petri Net Markup Language-PNML* [32], que é baseado no formato *Extensive Markup Language- XML* [33], voltado especificamente para RdP. A escolha deste formato foi feita após análise de literatura relacionada e identificação de que além de ser um formato amplamente aceito e utilizado em outros trabalhos atende às necessidades desta proposta, uma vez que facilita o intercâmbio de modelos entre o editor de Rede de Petri e o *Engine* de execução de modelos.

2.2.3 Linguagem para representação de modelos

No desenvolvimento de software é comum a utilização de modelos gráficos para representar os elementos que se deseja construir. Porém o símbolo que representa a

atividade, classe, associação ou qualquer outro elemento, não é entendido por uma ferramenta diferente daquela na qual o símbolo foi construído. Para possibilitar o intercâmbio de dados entre diferentes ferramentas é comum o uso das linguagens de marcação.

Uma linguagem de marcação é um conjunto de códigos aplicados a um texto ou dados, com intuito de adicionar informações particulares sobre esse texto ou dado, ou sobre trechos específicos. Este conceito recente envolve a codificação simples de sequências de dados em um arquivo de computador no formato texto-puro, ou seja, capaz de ser lido tanto por pessoas quanto por máquinas [34]. Um exemplo de linguagem de marcação difundida mundialmente é *eXtensible Markup Language* –XML.

2.2.3.1 Linguagem de Marcação Extensivel

A XML é uma linguagem de marcação de dados extensível padronizada pela W3C (*World Wide Web Consortium*) e utilizada para estruturar informações da *Web*. através da tecnologia dos marcadores (*markups*) que é a mesma tecnologia utilizada na linguagem HTML (*HiperText Markup Language*), ambas derivadas da SGML (*Standard Generalized Markup Language*). Entretanto, o XML possui um diferencial que é extensibilidade, uma vez que permite a criação de novos marcadores conforme necessidade. Além disso, o XML é amplamente utilizado como meio para facilitar o compartilhamento de informações através da rede mundial de computadores [35].

A flexibilidade da XML a torna um meio de realizar intercâmbio de dados entre ferramentas, como no caso dos modelos UML e RdP.

Modelos são os artefatos mais importantes na abordagem proposta neste trabalho, eles sofrem transformações e são interpretados durante o decorrer do processo de desenvolvimento, até que por fim, são gerados os ambientes desejados. Porém os modelos não podem ser tratados como uma figura ou imagem simplesmente. Como solução a OMG adotou a XML linguagem padrão para facilitar o intercambio dos dados dos modelos entre ferramentas.

2.3 Desenvolvimento Orientado a Modelo – MDD

O *Model-Driven Development-MDD*, Modelo de Desenvolvimento Orientado a Modelo, é uma abordagem de desenvolvimento que tem como foco a criação de

modelos [36], que são mais que artefatos primários na evolução do *software* [37], são como parte integrante do sistema [38]. O modelo é uma interface para adaptar, reparar, estender e atualizar o aplicativo [37].

O modelo de desenvolvimento orientado a modelo fornece diretrizes, linguagens, modelo e regras de transformação, métodos e ferramentas para apoiar a representação de requisitos de negócios e permite a geração de uma solução de tecnologia específica para cada empresa [39].

A proposta do MDD é fazer com que o engenheiro de *software* não precise interagir manualmente com todo o código fonte (Figura 3), concentrando-se em modelos de alto-nível. Ficando protegido das complexidades geradas na implementação com diferentes plataformas [38].

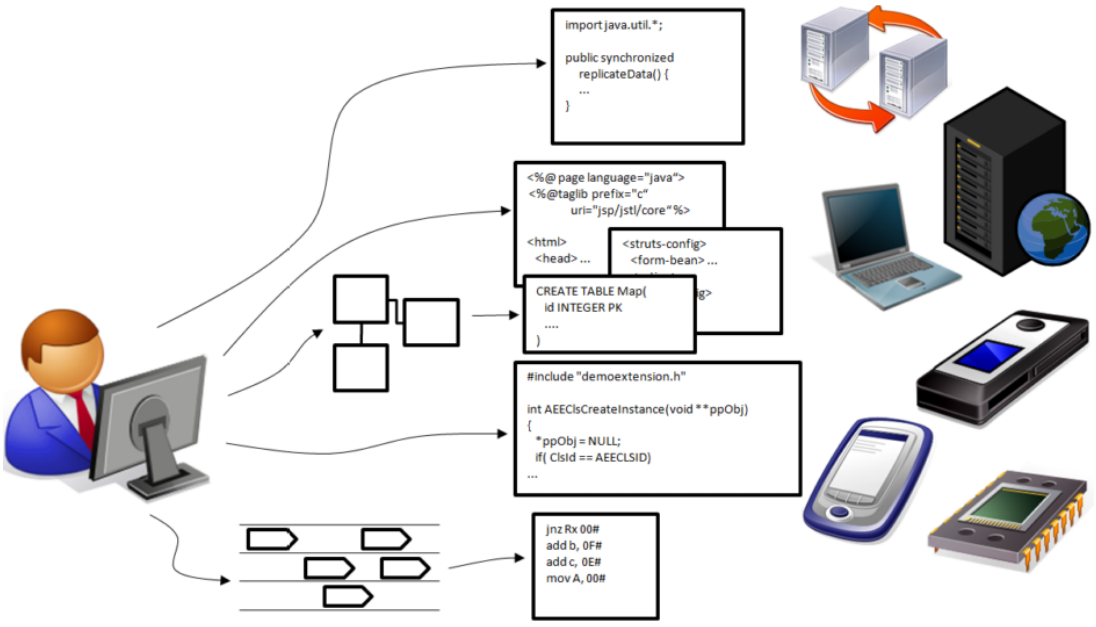


Figura 3 - Processo de desenvolvimento de software orientado a modelo [38].

Essa abordagem de desenvolvimento tem como principais vantagens:

- produtividade: redução de tempo de desenvolvimento com tarefas repetitivas através da possibilidade de geração de código automática [38] [40] [41] e através do conhecimento de especialista já agregado ao modelo [42].
- portabilidade: modelos de alto-nível podem ser transformados em código para diversas plataformas [38] [40].

- interoperabilidade: possibilidade de criação de adaptadores e conectores para a comunicação entre diferentes plataformas [38] [40].
- facilidade de manutenção: a manutenção é realizada diretamente no modelo o que facilita a compreensão do problema e a concepção de uma solução [38] [40].
- comunicação: os modelos de mais alto-nível tornam a comunicação acessível entre todas as pessoas envolvidas no processo de desenvolvimento [38] [40].
- reutilização: é possível fazer a reutilização de modelos adaptando-os a um novo contexto [38] [42] [40] [43].
- otimização: modelos possuem mais ferramentas para a verificação semântica e otimizações automáticas, assegurando implementações mais eficientes [38].
- código e conceitos confiáveis e primorosos: o alto-nível de abstração dos modelos permite uma maior correspondência com os conceitos de negócios e a possível automatização de geração de código elimina erros acidentais no código fonte [38] [42] [41] [43].

Em resumo, o desenvolvimento orientado a modelo permite que em longo prazo se tenha maior produtividade, qualidade e maior agregação de conhecimento nos produtos [38] [42] [40].

No entanto, a abordagem possui alguns desafios a enfrentar como:

- rigidez: devido à dependência dos modelos elaborados e a pouca influência do desenvolvedor [38];
- complexidade: as ferramentas de modelagem, transformações e geradores de código geram maior complexidade ao processo de desenvolvimento [38], e quanto mais modelos relacionados, maior a complexidade de ligação entre os artefatos [43].
- redundância: possibilidade de utilização de modelos que possuem representações de um mesmo conceito em diferentes níveis de abstração [43].

- desempenho: na utilização de geradores automatizados, o desempenho pode ser afetado devido a inclusão de código desnecessário no sistema [38].
- curva de Aprendizado: o MDD exige profissionais com habilidades na construção de modelos, manipulação de ferramentas de modelagem, transformações e geradores de código [38]. Os desenvolvedores devem ter conhecimento do impacto de cada etapa do processo no produto final [43].
- Alto Investimento Inicial: a preparação e a implantação do MDD necessitam tempo e esforço, mas os ganhos posteriores são significativos [38].

Apesar destes desafios, o desenvolvimento orientado a modelo é uma abordagem promissora para a criação de modelos complexos [44]. A modelagem é uma abstração de algum aspecto do sistema permitindo maior compreensão do problema. A linguagem usualmente utilizada para essa tarefa é a *Unified Modeling Language (UML)* [37].

A UML é uma linguagem padrão da indústria para modelagem de software visual, também chamada de *Domain Specific Language (DSL)* [40]. Ela permite a representação do sistema em vários níveis de abstração: requisitos, análise e projeto [42]. Contudo, a falta de fundamento matemático é visto como um obstáculo em níveis abaixo do de requisitos. Assim, as Redes de Petri, por ter uma forte base matemática, são adequadas para a análise formal [45].

Contudo, não há uma única forma de implementar sistema orientado a modelo [39], assim como a definição de quais atividades são necessárias no processo de desenvolvimento [38].

O documento de apoio para a execução do MDD é o MDD *Maturity Model*, o Modelo de Maturidade de MDD. Ele fornece um conjunto de práticas de engenharia, gestão e suporte para esse processo de desenvolvimento [46].

2.3.1 Arquitetura - Orientada a Modelo

A Arquitetura Orientada por Modelos, MDA (*Model-Driven Architecture*) [47], é um abordagem de desenvolvimento de *software* definida pela OMG [7], na qual os modelos são ponto chave no processo de desenvolvimento de software.

Modelos são abstrações criadas a partir das especificações do software, são entidades de primeira classe e durante o ciclo de desenvolvimento MDA estes modelos são transformados entre diferentes níveis de abstração, até que por fim é gerado automaticamente o código a partir das transformações dos modelos.

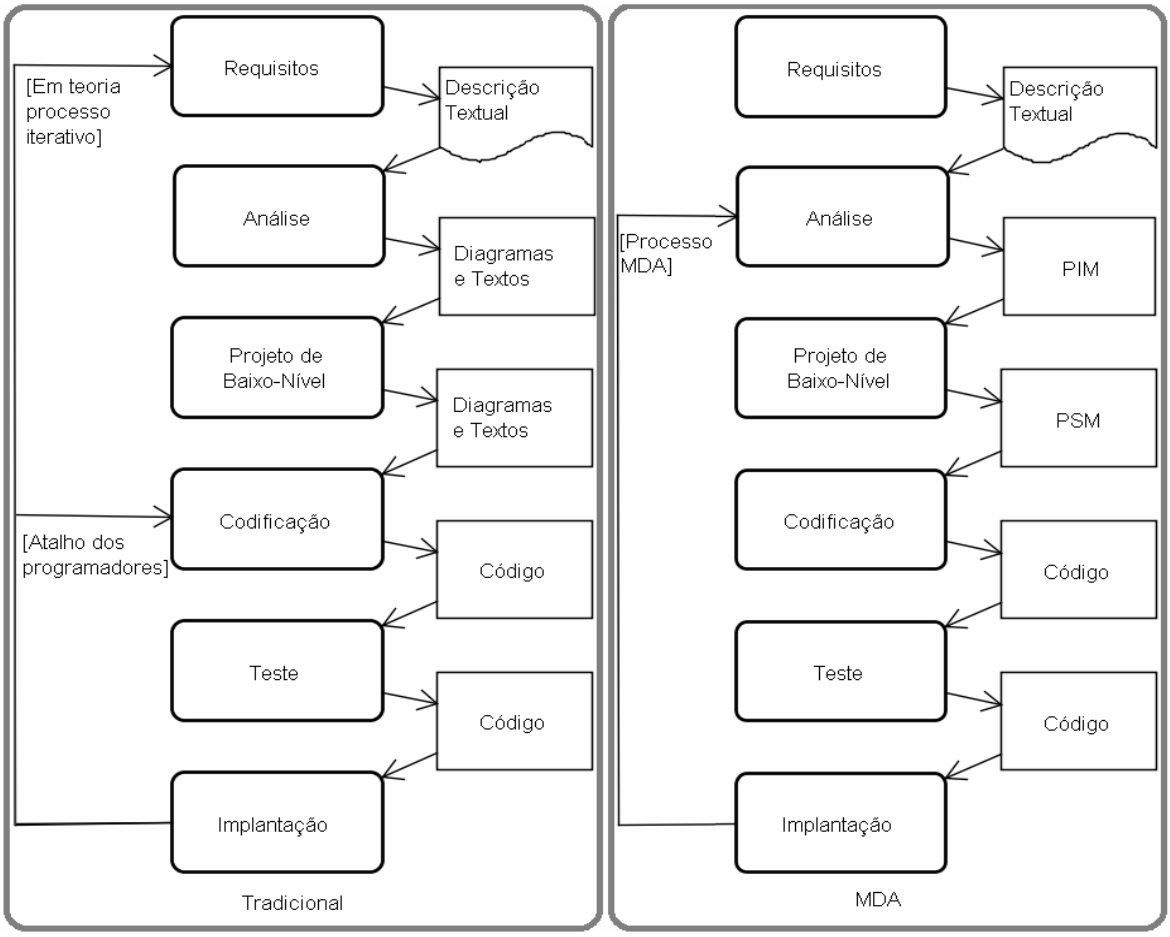


Figura 4 - Ciclos de vida de desenvolvimento. Adaptado de [7].

O ciclo de vida do desenvolvimento da abordagem MDA não é muito diferente do ciclo de vida tradicional de desenvolvimento, tanto que algumas fases são as mesmas, como pode ser visto na **Erro! Fonte de referência não encontrada.**. Ao comparar os dois ciclos de desenvolvimento algumas vantagens se evidenciam com relação à utilização da abordagem MDA. No ciclo Tradicional, “Diagramas e Textos” são especificados de acordo com uma arquitetura, logo inovações tecnológicas ou mesmo alterações nas tecnologias podem introduzir incompatibilidades no processo. Já no MDA, como os artefatos iniciais são independentes de plataforma, a portabilidade é facilitada proporcionando um aumento de produtividade dado à utilização da transformação entre os modelos [48].

O desenvolvimento tradicional é muito voltado à implementação dos códigos, dificilmente retorna as fases da modelagem, quando um problema é identificado altera-se diretamente no código sem passar pelos modelos de projeto, isso faz com que a codificação não seja o reflexo da modelagem do software o acarreta no desuso destes modelos.

Segundo Kleppe (2003), a MDA possui três modelos fundamentais na sua abordagem, são eles: PIM (*Platform Independent Model*), PSM (*Platform Specific Model*) e Código (*Code*) [7]. Existe também um modelo predecessor aos apresentados que trata do domínio da aplicação, suas regras de negócio, pessoas, coisas e lugares reais relacionadas ao software, este modelo é chamado na MDA de CIM (*Computation Independent Model*). No processo de desenvolvimento tradicional é conhecido como modelo de negócio ou de domínio [49].

Para realizar as transformações entre modelos e utilizá-los são adotados princípios que definem a arquitetura de modelos, que fornecem um conjunto de diretrizes para estruturação das especificações [50]. A seguir serão apresentados conceitos dos modelos MDA.

2.3.1.1 CIM

O Modelo Independente de Computação (*Computation Independent Model*-CIM) é uma representação simplificada de um sistema sem informações específicas sobre como deve ser implementado (sem definir detalhes como estruturas e processamento do sistema). Geralmente é um modelo UML conceitual ou de análise, que deve ser rastreável pelos modelos PIM e PSM que o irão implementar.

De acordo com Kleppe (2003), o CIM é um modelo independente de *software* relacionado ao modelo de negócios, especificam os processos de negócio, os *stakeholders*, departamentos, dependências entre processos, entre outras coisas, isto é, apresenta uma visão organizacional ampla do negócio. Quando uma parte do negócio necessita de um sistema de apoio, um modelo de *software* específico para esta parte do sistema é criado. Este modelo é uma descrição desta parte *software*.

Cada sistema de *software* apoia uma parte diferente de um modelo de negócio, há uma relação entre modelo de negócio e os modelos de *software* que descrevem o *software* de suporte ao negócio, como exemplificado na Figura 5 [7].

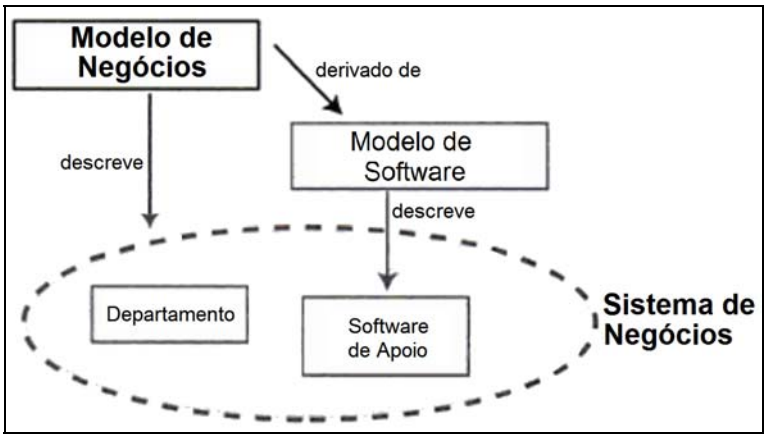


Figura 5 - Modelos de *Software* e de Negócio [7].

Partes do CIM podem ser apoiadas por um sistema de *software*, mas o próprio CIM permanece independente de *software* ou computação. Cada sistema de apoio parte inicialmente de um CIM. A transformação automática de um CIM para um PIM não é possível, pois a escolha das partes do modelo de negócio que necessitam de software de apoio é realizada por pessoas.

2.3.1.2 PIM

O Modelo Independente de Plataforma (*Platform Independent Model*-PIM) consiste em uma visão detalhada das funcionalidades do sistema, porém a partir de um ponto de vista independente da plataforma e sem detalhes técnicos. Considera-se este modelo como os requisitos e o projeto do sistema [51]. O foco deste modelo está em expressar as funcionalidades de negócio e comportamento, desconsiderando detalhes como em qual plataforma será implementado. Esta independência possibilita o uso do mesmo PIM em múltiplas plataformas. Segundo Barbosa (2011), a MDA enfatiza que o projeto correto de um PIM deverá sobreviver às mudanças tecnológicas nas quais esse modelo será submetido.

A OMG utiliza UML como linguagem padrão para construção de modelos para suas abordagens de desenvolvimento de software, como por exemplo, na criação do PIM. Porém, outras linguagens de modelagem também podem ser utilizadas para representar um PIM. Por exemplo, a dinâmica de um sistema concorrente pode ser representada através de um modelo de RdP que apresenta características deste tipo de sistema, como a sincronização. Portanto, um sistema pode ser representado por modelos diferentes escritos, em linguagens de modelagem diferentes.

A Figura 6 apresenta esta situação, onde um sistema é descrito através de duas modelagens diferentes, um modelo Entidade Relacional que pode definir a estrutura e um modelo RdP que pode definir o comportamento do sistema. Ambos os modelos são considerados notações adequadas para representar o PIM [51].

Kleppe (2003), afirma que algumas linguagens são mais expressivas e mais adequadas para representar aspectos específicos de um determinado sistema [7], que é o caso da RdP que é uma linguagem adequada para representar sistemas concorrentes.

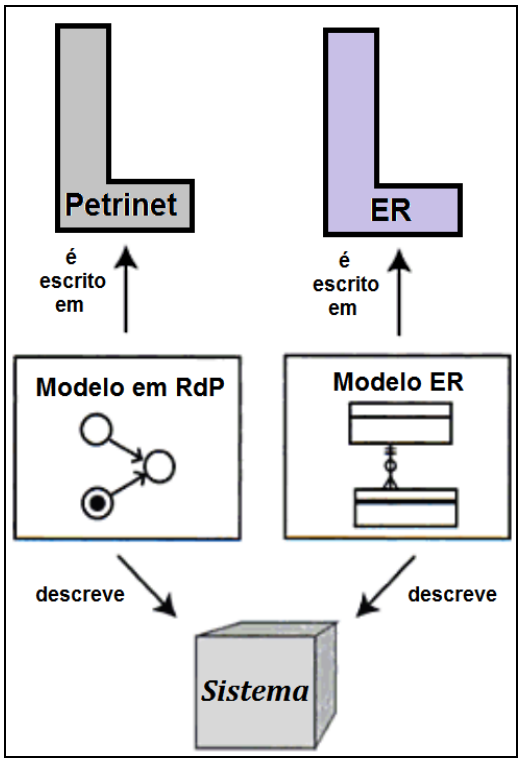


Figura 6 - Modelos diferentes de um sistema escrito em diferentes linguagens de modelagem [7].

2.3.1.3 PSM

Modelo Específico de Plataforma (*Platform Specific Model*-PSM) contém as mesmas especificações de um PIM, porém, com detalhes sobre a utilização de uma plataforma específica. Para que aconteça a transformação de PIM em PSM é necessário definir os detalhes da plataforma para qual se deseja converter o PIM. Este modelo geralmente contém informação suficiente para permitir a geração de códigos. Esta transformação de PIM para PSM pode ser feita manualmente ou automaticamente através de ferramenta [52]. As ferramentas de suporte a MDA geralmente possuem definições prévias para transformação do PIM para algumas plataformas específicas. Outras, além de oferecer

opções prévias de plataformas, também permitem a criação de novas definições para transformação de outras plataformas específicas.

Segundo Caliari (2007), o PSM pode ser aplicado na geração de código fonte ou na geração de novo PSM, isto é, um PSM pode ser utilizado como PIM em uma nova transformação para outra plataforma específica [53].

2.3.1.4 Código

O código é o resultado da transformação do PSM em sintaxe concreta da linguagem previamente definida, permitindo sua execução direta na plataforma especificada. Geralmente representado em alguma linguagem de programação, arquivos binários ou executáveis [51].

A Figura 7 apresenta a estrutura da MDA com seus modelos, artefatos de entrada e saída e suas transformações, uma visão geral do ciclo de vida de desenvolvimento a partir da MDA.

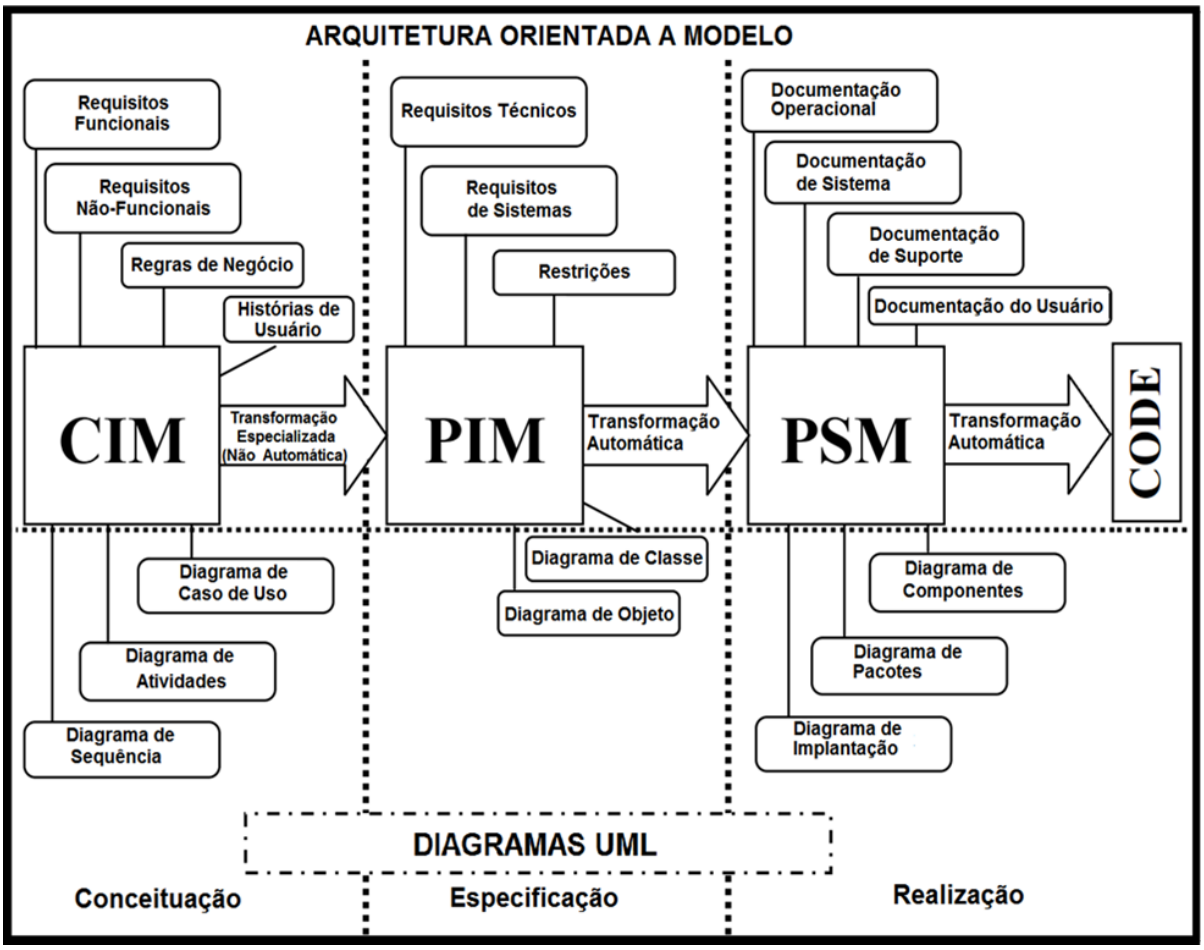


Figura 7 - Estrutura da Arquitetura Orientada a Modelo [54].

Kleppe (2003) destaca que funcionalidade dinâmica que não seja possível de ser representada através de modelos, necessita ser adicionada manualmente no código gerado [7]. Isto acontece devido ao fato de que ferramentas de transformação utilizadas nos projetos de desenvolvimento de software como Enterprise Architect (EA) [55] e Astah [56] dão suporte a modelagem de diagramas estruturais e comportamentais, porém não fornece suporte a transformação dos diagramas comportamentais para código fonte. Isto foi constatado durante a utilização dos mesmos para elaboração deste trabalho.

A mesma situação também é apresentada no trabalho de Caliari (2007). Nele o autor afirma que códigos de métodos mais complexos não são gerados automaticamente pelas ferramentas adotadas em seu estudo que são AndroMDA e ArcStyler [53]. Ele explica que as ferramentas dão suporte à geração da estrutura do sistema e os principais métodos de acesso.

Esta situação é prevista pela OMG no *MDA Guide Version 1.0.1*, que apresenta as técnicas possíveis de serem utilizadas para realização das transformações da abordagem MDA. As transformações entre modelos PIM e PSM até geração de código fonte podem ser feitas de forma automática utilizando software próprio para este fim, através de técnicas de transformação de modelos de forma manual respeitando todas as regras de transformação e mapeamentos entre os diferentes níveis de abstração, ou utilizando uma combinação de técnicas automáticas e manuais para transformação de modelos, geração e aprimoramento de código [52].

2.3.2 Modelos e Metamodelos

Uma das características do MDD é a geração automatizada de código fonte [43], sendo que para a realização dessa automação é necessário que os modelos tenham um significado bem definido [42].

Para garantir que os modelos construídos estejam semanticamente corretos e completos, as principais abordagens existentes utilizam o conceito de meta-modelagem, representada na Figura 8 [38].

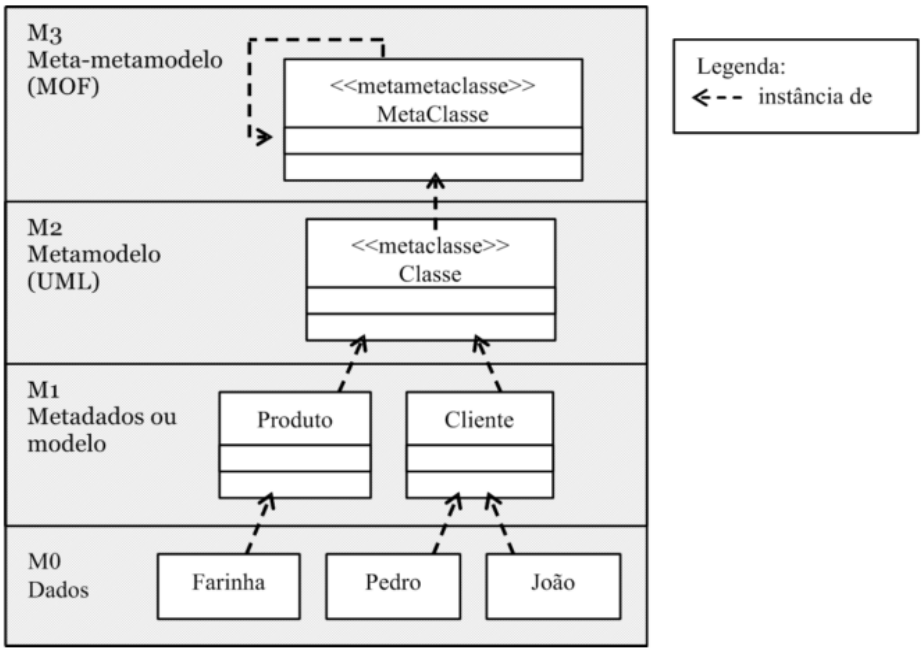


Figura 8 - Representação da meta-modelagem [38].

De acordo com a representação da meta-modelagem apresentada na Figura 8, o primeiro nível (M0) corresponde aos dados (instâncias) do sistema. O segundo nível (M1) é onde se pode encontrar o modelo que descreve os dados. O terceiro nível (M2) é o nível onde é feita a definição dos modelos que descrevem os dados, os metamodelos. E o quarto nível (M3) correspondente à definição dos metamodelos [38].

Um exemplo de meta-metamodelo é o *Meta-Object Facility* (MOF) definido pela *Object Management Group* (OMG), que é o meta-metamodelo que define a UML, além de servir como base para o *MDA* [42].

O principal objetivo do metamodelo é estabelecer regras arquiteturais específicas e fornecer um mecanismo para que os projetistas realizem a modelagem padronizada e livre de ambiguidades [41]. Volter 2004 menciona padrões que podem ser utilizados no desenvolvimento orientado a modelos.

O modelo é um conjunto de elementos que descrevem formalmente determinados aspectos como interface, segurança, banco de dados, cenário de utilização de um sistema. O desenvolvimento de um *software* pode envolver uma ou mais modelagens distintas [44].

Ele facilita a comunicação entre pessoas e máquinas, verificação da integralidade, análise da condição de corrida[1], geração de casos de teste, controle de indicadores como de estimativa de tempo, utilização de padrões e transformação em código fonte [42].

2.4 Realidade Virtual - RV

Algumas tecnologias vistas nos filmes e livros de ficção científica acabaram se tornando realidade com o avanço das tecnologias reais, principalmente com o aprimoramento dos recursos computacionais, alcançados ao longo dos anos através do trabalho de pesquisadores, cientistas, engenheiros, etc., que buscam criar soluções que auxiliem na resolução de problemas e/ou execução de tarefas realizadas pelo homem.

Robôs, máquinas capazes de tomar decisões por conta própria, detentoras de uma inteligência artificial (*Artificial Intelligence*-AI), mundos imaginários onde pessoas interagiam com personagens animados (*Virtual Reality*-VR) ou animações que saiam de um mundo fictício para mundo real e são passiveis de interação com pessoas e/ou outros elementos reais (*Augmented Reality*-AR) e muitos outros, são exemplos de tecnologias que saíram da ficção e são empregadas em diversas áreas do conhecimento.

A Realidade Virtual-RV é uma destas tecnologias que é utilizada em vários setores, como na medicina, aviação, eletrônica, química, indústrias automotivas, exploração de petróleo e várias outras [57]. Dentre os propósitos de sua utilização destaca-se a possibilidade de analisar processos, projetar produtos, avaliar designer e ergonomia, realizar treinamentos, auxiliar na tomada de decisões e várias outras possibilidades.

Para Aukstakalnis e Blatner (1992), Realidade Virtual é uma forma de visualizar, manipular e interagir com computadores e dados extremamente complexos [58]. Segundo Kirner (1997), é uma técnica avançada de interface, onde o usuário pode realizar imersão, navegação e interação em um ambiente sintético tridimensional gerado por computador, utilizando canais multissensoriais [59].

A experimentação de envolvimento no ambiente virtual pode ser percebida de forma imersiva e ou não imersiva. A RV imersiva é aquela em que o usuário não percebe

[1] Condições de Corrida (*Race Conditions*): situações em que dois ou mais processos leem e escrevem dados compartilhados e o resultado final depende da ordem de quem precisamente executa [79].

os estímulos externos ao ambiente de RV, pois os seus principais sentidos, como a visão e a audição, são incitados pelos dispositivos de saída. Já a RV não-imersiva utiliza apenas parte dos sentidos do usuário, possibilitando assim que o ambiente externo influencie em suas percepções [60].

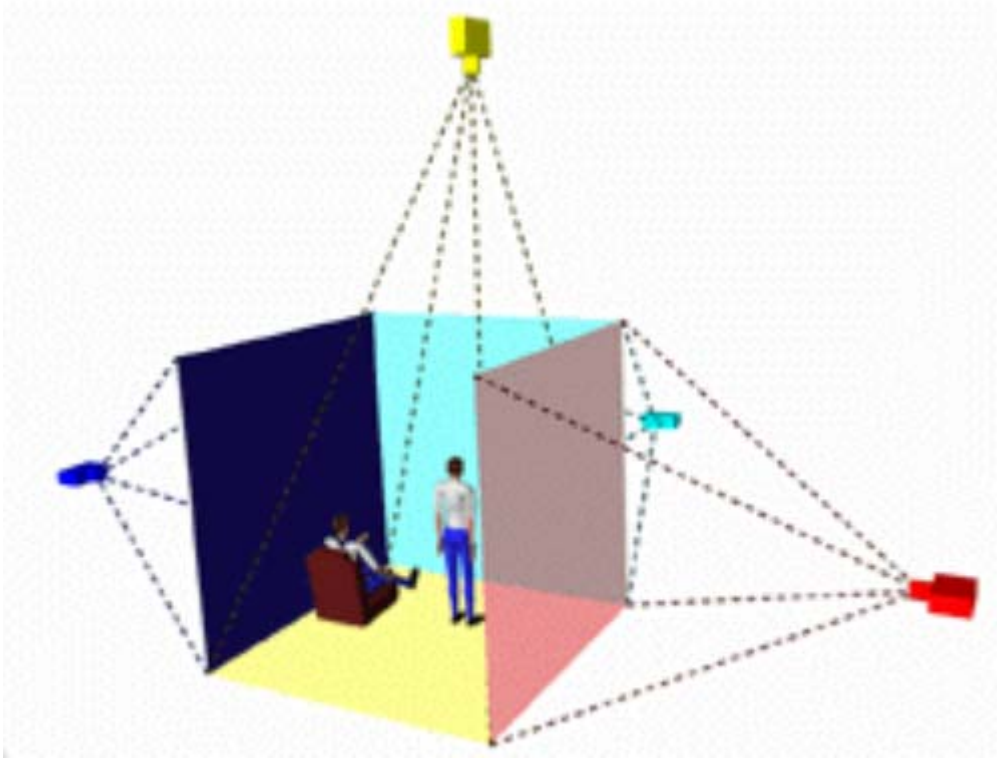


Figura 9 - Representação de uma CAVE [61].

Um exemplo de RV imersiva é alcançado através da utilização de um dispositivo chamado *Cave Automatic Virtual Environment* (CAVE) [62], uma espécie de cômodo em forma de cubo como apresentado na Figura 9, onde imagens estereoscópicas sincronizadas são projetadas nas paredes, teto e piso, conjuntamente com estímulos sonoros, proporciona ao usuário a sensação de estar dentro da realidade artificial gerada por meio de processamento computacional [63]. Outro exemplo de RV imersiva é o experimentado através da utilização do capacete ou óculos de RV. Tanto no CAVE como no capacete de RV, os usuários se valem de assessórios especiais e sensores para interagir com o ambiente virtualizado. A sensação de imersão pode ser melhorada com adição de dispositivos multissensoriais que estimulam outros sentidos como o olfato e o tato.

RV não-imersiva é uma forma mais simples se ver um ambiente virtual, ela pode ser experimentada basicamente através da utilização monitores. Porém a utilização de dispositivos que incitam outros sentidos como a audição, pode introduzir algum nível de

imersão nesta forma de percepção do ambiente virtual [64]. Apesar da impossibilidade de uma imersão completa no ambiente virtual, este tipo de percepção de RV tem seus pontos positivos, como o baixo custo e a facilidade de uso, evitando as limitações técnicas e problemas decorrentes do uso de capacetes de RV [65].

Cada vez mais pesquisadores utilizam a potencialidade da RV em diversas áreas do conhecimento para diversos fins, a versatilidade e multidisciplinariedade sempre garantem avanços ao universo da RV. Um exemplo disto são as soluções utilizadas pelas indústrias automotivas.

A Ford utiliza CAVE para auxiliar nos projetos de seus carros. No desenvolvimento da B-MAX (Figura 10), minivan compacta vendida na Europa, a Ford usou o CAVE para testar a eficiência do sistema de portas traseiras corrediças com pilar central integrado. Também ajudou a aprimorar a visibilidade das janelas traseiras e os limpadores tipo "borboleta", que se movem em direções opostas [66]. Outro modelo, o Focus da Ford também passou por alterações devido à utilização do CAVE, que permitiu aumentar a eficiência dos limpadores; ajudou a ampliar o espaço dos passageiros de trás, testando diferentes desenhos de bancos dianteiros e apoios de cabeça, também serviu; para avaliar o impacto do desenho da moldura das portas na visibilidade e para minimizar os reflexos nos vidros e mostradores.

Figura 10- CAVE da Ford [66].

Outra fabricante de carros que utiliza RV em seus projetos é Volkswagen. Inaugurado em 2008 em São Bernardo do Campo, o Centro de Realidade Virtual utiliza a tecnologia 3D para simular virtualmente os novos projetos e compartilha-los com todo o Grupo Volkswagen (Figura 11) [67].

Figura 11 - Centro de Realidade Virtual da Volkswagen [67].

O centro de Realidade Virtual da Embraer já foi considerado na época de sua inauguração o maior e mais desenvolvido da América Latina. Diante de uma concorrência globalizada e com intuito de ganhar mercado internacional, projetistas, técnicos, engenheiros e funcionários têm à sua disposição, um moderno centro de tecnologia que tornou possível a redução do prazo e dos custos de desenvolvimento do projeto de uma aeronave [67].

2.4.1 Realidade Aumentada - RA

Para entendimento do funcionamento da RA, é importante se ater a uma forma mais ampla de entendimento da realidade, a Realidade Misturada (RM) representada na Figura 12.

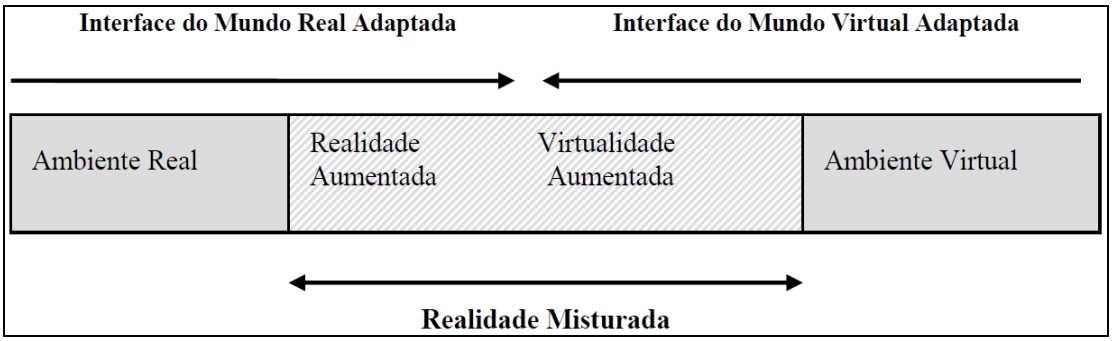


Figura 12 - Ambiente de Realidade Misturada [68].

A RM é dividia em Interface do Mundo Real Adaptado e *Interface* do Mundo Virtual Adaptado, sendo a RA um caso particular da RM assim como a Virtualidade Aumentada (VA), porém a RA é o termo mais utilizado de maneira mais ampla.

A realidade aumentada usa técnicas computacionais que geram, posicionam e mostram objetos virtuais inseridos no ambiente real, enquanto a virtualidade aumentada usa

técnicas computacionais para capturar elementos reais e reconstruí-los, como objetos virtuais realistas, colocando-os dentro de mundos virtuais e permitindo sua interação com o ambiente. Em qualquer dos casos, o funcionamento do sistema em tempo real é uma condição essencial [68].

A realidade aumentada envolve quatro aspectos importantes [68]:

- renderização de alta qualidade do mundo combinado;
- calibração precisa, envolvendo o alinhamento dos objetos virtuais em posição e orientação dentro do mundo real;
- interação em tempo real entre objetos reais e virtuais.

A Realidade Aumentada é conceituada por vários autores, que conforme a sua área tem uma visão sobre o a técnica.

Para Azuma (1997), RA consiste numa técnica avançada de *interface* computacional, que combina elementos virtuais tridimensionais com ambiente real e permite interação em tempo real [69]. Insley (2003) define como uma melhoria do mundo real com textos, imagens e objetos virtuais, gerados por computador [70]. Segundo Milgram (1994), é a mistura de mundos reais e virtuais em algum ponto da realidade/virtualidade contínua que conecta ambientes completamente reais a ambientes completamente virtuais [71].

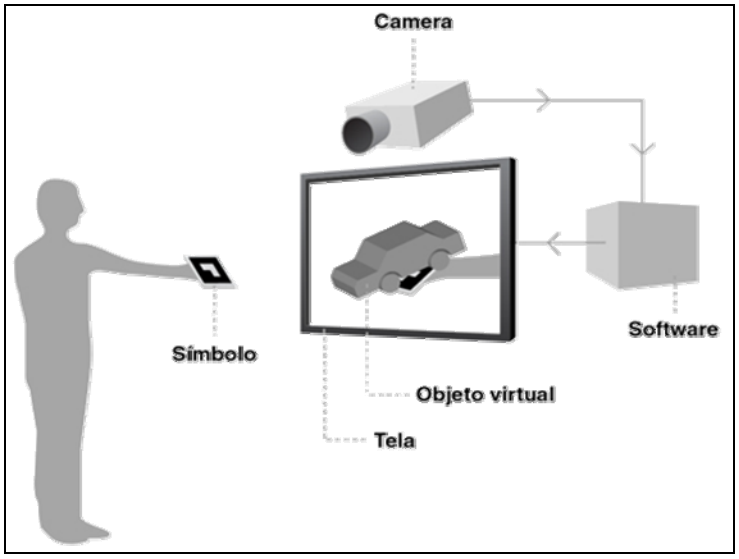


Figura 13 - Esquema básico de funcionamento da RA [72].

Uma aplicação típica de RA é obtida a partir de uma imagem captada por uma câmera, um marcador (Figura 14) que é substituído por um objeto tridimensional que pode ser visto incluído na imagem do ambiente real como apresentação na Figura 13.

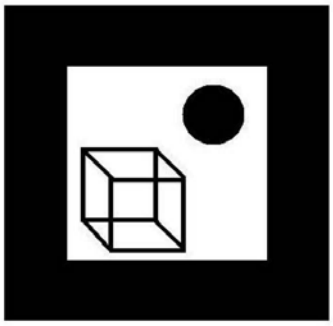


Figura 14 - Marcador [72].

Existem também aplicações de RA que utilizam QR-Code que são marcadores mais sofisticados que podem envolver codificação [72], ou ainda aplicações não baseada no uso de marcador, porém o uso de marcador é bastante comum.

Aplicações de RA são bastante exploradas para fins de marketing e propaganda, que utilizam deste recurso para apresentar suas marcas de forma criativa e diferente das até então convencionais.

Existem aplicações de RA para dispositivos móveis, como os *smartphones*, *tablets* e outros dispositivos do gênero, uma vantagem com relação à utilização do um computador comum é a que esses equipamentos dispõem de todos os periféricos básicos para RA em um único aparelho, que em sua maioria são práticos para levar à qualquer lugar.

São encontrados na literatura vários trabalhos que utilizam RA para diversas áreas como educação, medicina, entretenimento, arquitetura entre outros. Um exemplo de aplicação da RA na educação é o Livro de Realidade Aumentada para Crianças Portadoras de Necessidades Especiais (LIRA-ESPEC), aparentemente um livro tradicional, porém ele é composto de elementos da RA que potencializam os sentidos da criança com necessidades especiais ampliando sua capacidade de a aprendizagem através de estímulos visuais, sonoros e táteis [73]. A Figura 15 apresentação imagens deste livro.

Figura 15 - Livro de RA [73].

Outro exemplo aplicação de RA agora voltado à arquitetura e engenharia civil é o projeto realizado pela Rossi Residencial que resultou em um registro no livro dos recordes (*Guinness Book of World Records*) do maior marcador de RA do mundo [74]. A empresa criou um marcador do tamanho aproximado de um campo de *baseball* e o dispôs no local onde o empreendimento seria construindo, os clientes eram levados de helicóptero e sobrevoavam o marcador e com auxilio das técnicas de RA viam a construção entre meio as demais construções com se já estivesse finalizada e podiam apreciar os detalhes arquitetônicos da construção. A Figura 16 apresenta a visão que os clientes tinham.

Figura 16 - Prédio 3D visualizado durante sobrevoou [74].

Ambos os exemplos apresentados utilizam elementos básicos para realização da RA, como os encontrados em computador pessoal, porém o avanço tecnológico vem proporcionando o surgimento de dispositivos compactos com grande complexidade de processamento e com todos os dispositivos necessários como teclado e câmera embutidos em um único aparelho, são os chamados dispositivos *mobiles*. Estes equipamentos possibilitam novas formas de experimentar a RA através de aplicativos desenvolvidos para esta finalidade. Dentre estes tipos de equipamentos destacam-se os *smartphones, tablets* e *Google Glass* [75] que é um dos mais novos produtos da empresa norte americana Google.

2.5 Geradores de Ambientes Virtuais

A estrutura de um sistema de realidade virtual pode ser mostrada sob diferentes pontos de vista e graus de detalhamento [59]. A Figura 17 apresenta uma visão geral de um sistema de RV.

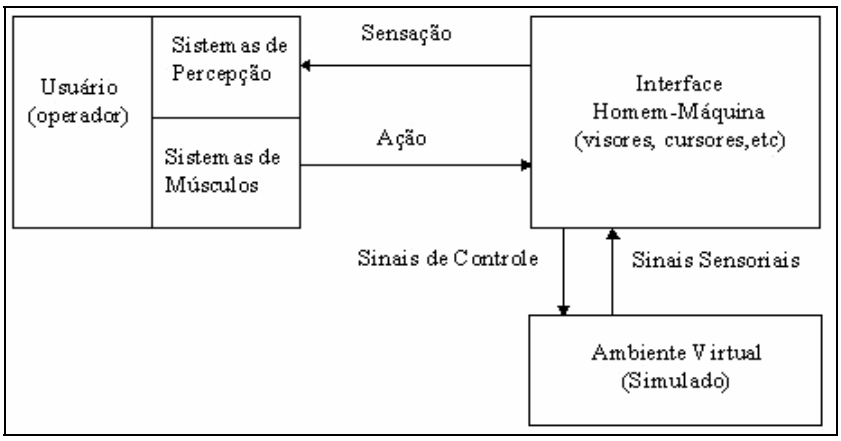


Figura 17 - Estrutura básica de uma aplicação de ambiente realidade virtual [59].

A interação do usuário com o Processador de Realidade Virtual é intermediada pelos dispositivos de entrada e saída. O Processador de RV primeiramente lê a entrada do usuário e acessa a Base de Dados para calcular as instâncias do mundo que se referem aos quadros a serem mostrados em sequência. Como não é possível prever as ações do usuário, os quadros devem ser criados e distribuídos em tempo real [59]. Esta estrutura é apresentada na Figura 18.

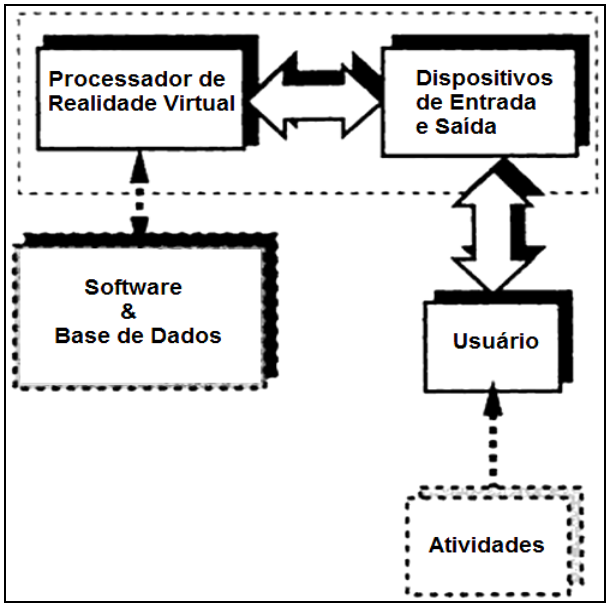


Figura 18 - Arquitetura de Sistema de Realidade Virtual [60].

O gerador de ambiente virtual é um sistema de computação de alto desempenho que contém uma base de dados referente ao mundo virtual, no qual existe a descrição dos objetos do ambiente virtual, junto com a descrição dos seus movimentos, comportamentos, efeitos de colisões, entre outros.

Na proposta de metamodelo de Palma (2001), os princípios observados na Figura 18 referente à estrutura de um sistema de RV são aplicados. O ambiente virtual é gerado com objetos virtuais. Os objetos poderiam pertencer a uma biblioteca de objetos virtuais de manufatura previamente codificados. A qualidade gráfica e o realismo do mundo virtual são diretamente relacionados com a riqueza da biblioteca. Esta proposta possui todos os elementos que compõe um gerador de ambientes virtuais, banco de dados de modelos e objetos virtuais, um processador de RV, dispositivos de entrada que englobam tanto os modelos de RdP que controlam o comportamento do ambiente, assim como, as interações que possam vir a ocorrer pelo usuário. Os dispositivos de saída podem ser entendidos como dispositivos que proporcionam a imersão dos usuários, no caso da proposta um CAVE, assim como óculos estéreos que complementam a sensação de imersão no ambiente virtualizado. E é com base no trabalho de Palma, que esta proposta foi fundamentada, buscando aprimoramento de sua ideia inicial.

2.6 Trabalhos Relacionados

O desenvolvimento baseado em modelos (MDD) não é um tema recente, já vem sendo estudado há anos por vários autores que propõem variações da abordagem MDD para aprimorar o desenvolvimento de software. Com intuito de se obter uma fundamentação sólida a respeito dos temas tratados neste trabalho foram pesquisados trabalhos relacionados que apresentam diversos exemplos de aplicações e modelos de desenvolvimento de software. A seguir são apresentados alguns trabalhos que serviram como base de estudos para construção deste trabalho.

Como citado anteriormente este não é um tema novo, Palma (2001), mesmo não citando diretamente o termo MDD da OMG trata em seu trabalho do desenvolvimento baseado em modelo propondo um metamodelo para o desenvolvimento de sistemas de simulação de eventos discretos com interface de Realidade Virtual (RV) aplicados a ambientes ou estações de trabalho de manufatura. O metamodelo de Palma (2001) é composto por módulos muito similares aos aplicados na construção da proposta deste trabalho, além de

também utilizar Rede de Petri para controlar a dinâmica da simulação [31]. O resultado final deste trabalho é um metamodelo para o desenvolvimento de simulação centralizada baseada em modelos de Redes de Petri com interface de RV distribuída que suporta ainda iteração de diferentes usuários.

Barbosa (2011) aborda em seu trabalho uma problematização da abordagem MDA que é a não existência de uma forma de garantir que transformações MDA sejam preservadoras de semântica, e nem que seus modelos envolvidos nas transformações sejam formais o suficiente para se permitir o uso de técnicas de verificação de equivalência, gerando críticas sobre a eficácia dessa abordagem [51]. A tese trata de abordagens consolidadas de métodos formais na arquitetura de MDA, tendo como contexto específico o desenvolvimento de software para sistemas embarcados com características de concorrências. Propõe extensões para parte da arquitetura MDA para que se possam construir modelos semânticos que representem aspectos estáticos e dinâmicos, ambos essenciais na semântica dos modelos envolvidos nas transformações e nos mecanismos de verificação de equivalência desses modelos [51]. Para prover uma avaliação do paradigma concorrente o autor utiliza modelos em Rede de Petri e Redes IOPT [51].

O trabalho de Silva (2013) apresenta um estudo sobre transformação modelo-modelo baseado em MDA, dando como exemplo a transformação de um modelo em Rede Petri que é utilizado como meio de tentar assegurar o correto andamento dos processos existentes dentro das transformações de modelo e da própria abordagem MDA [76].

Kulesza (2013) defende em sua tese que o MDD pode aumentar a produtividade do desenvolvimento de aplicações voltadas a TV Digital [77]. Para isto descreve em seu trabalho uma abordagem MDD para desenvolvimento e aplicações Ginga-NCL.

Lucrédio (2009) apresenta a tese de que a combinação entre o desenvolvimento orientado a modelos e a reutilização de software em um processo sistemático pode elevar e/ou melhorar os níveis de reutilização alcançados por uma organização de software, comparado ao processo tradicional de desenvolvimento baseado em código. O mesmo autor enfatiza que o MDD deve estar presente em todas as etapas de desenvolvimento de software, análise, projeto e a implementação [38].

Caliari (2007) faz uma análise dos conceitos de PIM e PSM da abordagem MDA, para isto utiliza três ferramentas que são analisadas a fim de verificar se realizam as transformações de acordo com os conceitos da MDA.

3 PROPOSTA DE DESENVOLVIMENTO DE AMBIENTES VIRTUAIS ORIENTADO A MODELOS

Este capítulo apresenta a proposta do metamodelo para geração de ambientes virtuais orientados a modelo através de ferramentas de transformação MDA. Serão apresentados também os conceitos aplicados para o desenvolvimento da proposta do metamodelo MDA e a análise de viabilidade desta proposta.

Para melhor entendimento do texto, foram sintetizados abaixo os termos comumente encontrados no contexto de MDA e amplamente utilizados no levantamento bibliográfico. Os termos são:

- Metamodelos - descrevem como devem ser construídos os modelos;
- Modelos - representam as instâncias dos metamodelos;
- Transformações - possuem um conjunto de regras que definem como um modelo de entrada pode ser transformado em outro modelo de saída;
- CIM - Modelo Independente de Computação;
- PIM - Modelo Independente de Plataforma;
- PSM - Modelo para Plataforma Específica;
- Código - É a sintaxe concreta de uma Linguagem de Programação;

3.1 Conceitos de MDA aplicados na proposta

O desenvolvimento de sistemas por meio do metamodelo proposto neste trabalho, inicialmente poderá gerar investimentos em ferramentas, é um processo que ainda requer esforços de desenvolvimento semelhante ao tradicional, mas com a linha do tempo poderá agregar valor ao adotante, pois terá um processo definido, uma coleção de documentos úteis para manutenção e atualização do sistema, visto que qualquer mudança ocorre desde o PIM. Outros fatores importantes são a possibilidade de se reutilizar constantemente classes já definidas, além de proporcionar a configuração de novos ambientes através da manipulando de objetos presentes dentro da biblioteca apenas orientada ao modelo comportamental da RdP.

Uma das características do MDA é a geração automática de código fonte, para realizar essa automação é necessário que os modelos sejam bem definidos e adequados aos padrões e regras da MDA, para isto deve se empregar à abordagem de desenvolvimento

de software definida pela OMG [49], na qual os modelos são ponto chave no processo de desenvolvimento.

Os artefatos utilizados são modelos gerados e interpretados por ferramentas de transformação, que podem ser transformados em outro modelo ou código fonte.

Segundo a OMG [49] existem duas abordagens para a geração automática de sistemas:

- Na primeira, um padrão de transformação é aplicado ao modelo para produzir novos/outros artefatos ou modelos específicos de uma tecnologia. Como exemplo, um modelo de informações de negócio pode ser transformado em um documento XML e/ou código C# ou Java, conforme exemplificado na Figura 19.

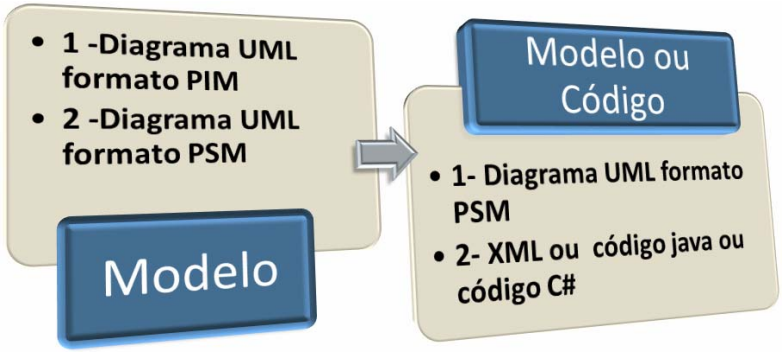


Figura 19 - Representação da primeira abordagem MDA. [Próprio Autor]

Na segunda abordagem é empregado um *Engine* de execução de modelo, implementado em alguma plataforma tecnológica específica. Este *Engine* executa diretamente o modelo tratando-o como um código fonte interpretável. O *Engine* pode ser capaz de produzir e consumir mensagens XML [49]. A segunda abordagem é representada na Figura 20.

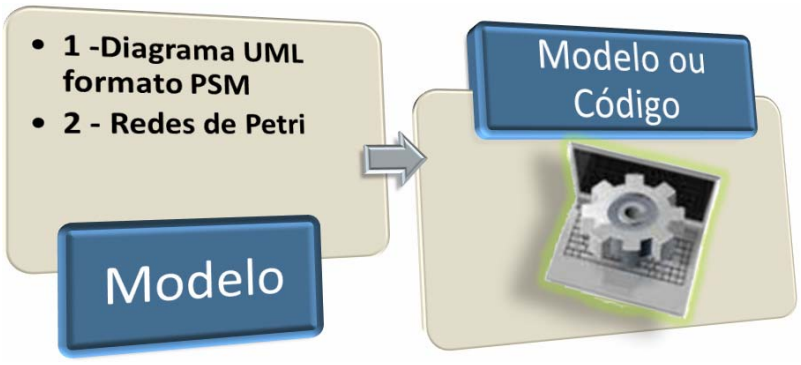


Figura 20 - Representação da segunda abordagem MDA. [Próprio Autor].

A proposta deste trabalho é a de aplicar as duas abordagens de MDA de forma complementar.

A primeira abordagem é utilizada para gerar as estruturas básicas dos elementos que compõem um ambiente. Nesta etapa são abstraídas as características de cada objeto necessário para viabilizar ambiente desejado, isto contempla a identificação dos objetos e definição de seus atributos e métodos. Estas atividades de identificação e definição dos elementos e suas relações são representadas por meio de diagramas da UML.

A segunda abordagem é empregada para definir os elementos e o comportamento do ambiente, nesta etapa são aplicadas as regras definidas na especificação do software que fornecem a dinamicidade ao ambiente. Os elementos e as regras de comportamento do ambiente podem ser representados através de modelos UML e de Redes de Petri.

O metamodelo proposto utiliza a combinação das duas abordagens da MDA. A primeira fornece a parte estrutural do ambiente através das transformações dos modelos, culminando na geração automática do código que dá origem ao arquivo de biblioteca de classes. Na segunda abordagem, é utilizada a biblioteca de classes e a especificação do software para elencar os objetos que serão instanciados para geração do cenário. Também é definido o comportamento dos objetos no ambiente e finalmente todos os modelos criados e transformados são interpretados e geram o ambiente virtual desejado. Todo fluxo de atividades da proposta do metamodelo será apresentado na seção que trata sobre o metamodelo.

A primeira abordagem de MDA adotada nesta proposta para a criação e transformação dos modelos é fundamentada nos conceitos da MDA do OMG com modelos que serão expressos em uma linguagem baseado no MOF (*Meta Object Facility).* O MOF usa metamodelos especificados em UML para descrever a linguagem de modelagem, assim como o inter-relacionamento entre objetos.

A partir do PIM, há ferramentas de desenvolvimento MDA que fazem o mapeamento conforme padronização da OMG e produz um ou mais Modelos Específicos de Plataformas (PSMs) em UML, um para cada plataforma de destino que o desenvolvedor definir. Na etapa seguinte o PSM é transformado gerando o código fonte. O código gerado pode receber ajustes quando necessário, principalmente classes de objetos que possuem métodos que exigem um tratamento específico. Aprimoramentos de código são suportados, pois as ferramentas que realizam transformações automáticas são fortemente centradas na

parte estrutural do código, dificilmente conseguem interpretar modelos que representam o comportamento dos objetos.

3.2 Metamodelo do MDA

O metamodelo proposto neste trabalho para o desenvolvimento de ambientes virtuais é apresentado na Figura 21 que mostra todo fluxo das atividades necessárias para geração de um ambientes baseado na MDA utilizando modelos UML e RdP.

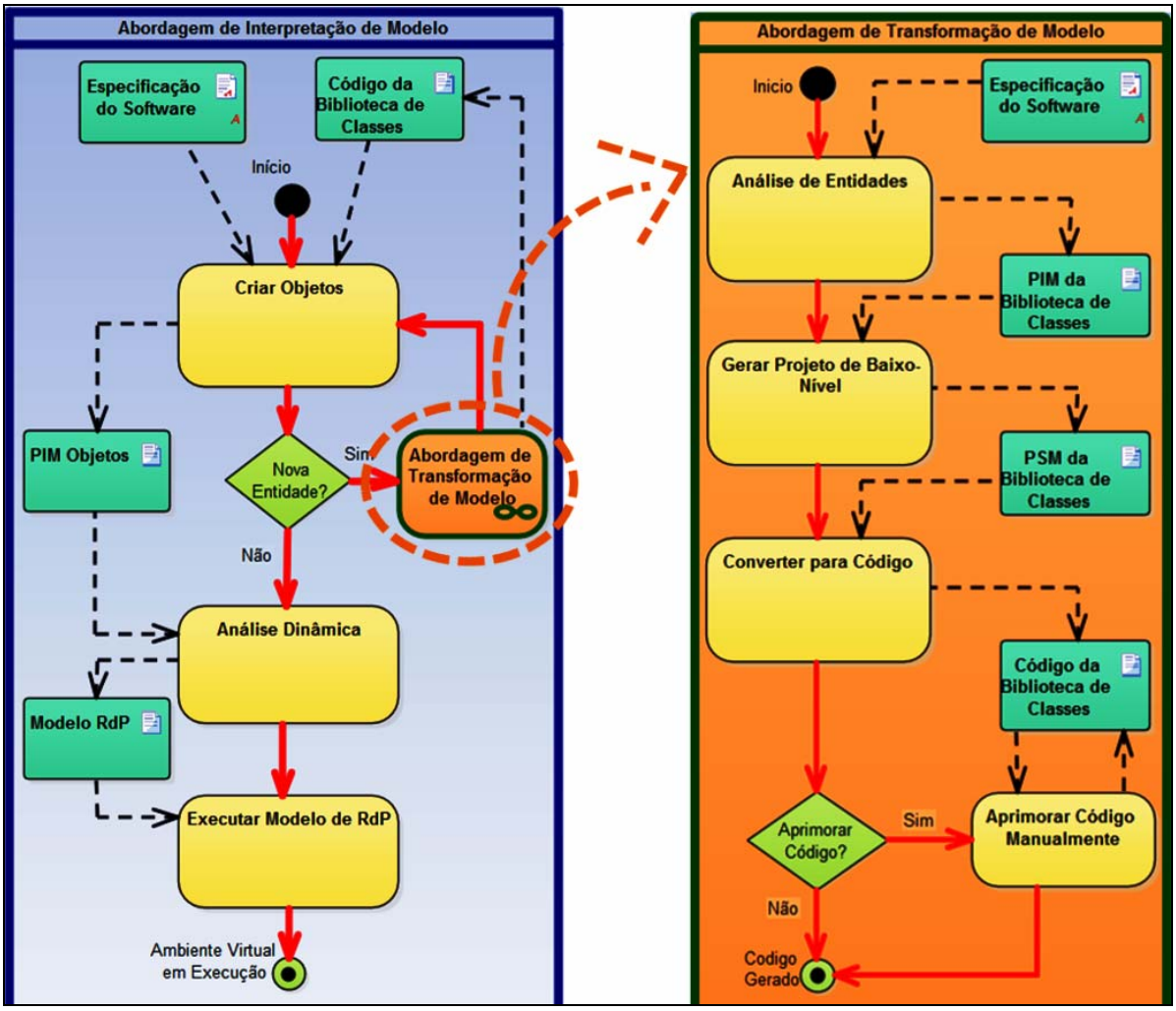


Figura 21- Fluxo de atividades e artefatos gerado do Metamodelo proposto para desenvolvimento de ambientes virtuais. [Próprio autor]

A Figura 21 apresenta dois diagramas de atividades da UML que contemplam todo o fluxo de atividades do metamodelo. O diagrama com a raia na cor laranja nominada “Abordagem de Transformação de Modelo” foi constituído com base na primeira

abordagem MDA apresentada na Figura 19. O diagrama com a raia na cor azul nominada "Abordagem de Interpretação de Modelo" representa a segunda abordagem MDA apresentada na Figura 20. A combinação dos dois diagramas representam todas as atividades que compõem o metamodelo proposto neste trabalho.

O fluxo de atividades do metamodelo inicia-se com a aplicação da Abordagem de Interpretação de Modelo. Partindo do principio de que já exista uma biblioteca de classe e o documento de especificação executa-se a atividade "Criar Objetos". Nesta atividade são elencados todos os elementos necessários para gerar um ambiente virtual, para isto são criados modelos compostos por objetos necessários para geração do ambiente virtual. Um exemplo deste tipo de representação pode ser feito através do diagramas de objetos da UML ou através de modelos próprios capazes de representar os elementos necessários.

Na sequencia, se não existir necessidade criação de nova classe inicia-se a "Análise Dinâmica", que é a atividade onde é modelado o comportamento do ambiente virtual. Nesta fase, através do modelo de RdP, os objetos modelados anteriormente são relacionados as transições da RdP que são responsáveis por executar os métodos dos objetos de acordo com os parâmetros inseridos. Por fim, o *Engine* executa o modelo RdP gerando o ambiente virtual modelado e configurado nas atividades anteriores.

Em duas situações pode ser necessário utilizar o fluxo diferente deste apresentado anteriormente. O primeira situação é quando for necessário criar a biblioteca de classes pela primeira vez e só existir a especificação do software. A outra situação é quando a biblioteca existir, mas houver a necessidade de criação de uma nova entidade. Nas duas situações é aplicada a Abordagem de Transformação de Modelo como apresentada no metamodelo proposto.

O fluxo da Abordagem de Transformação de Modelo, inicia-se com a Análise de Entidades, tendo com base a Especificação do Software. Nesta primeira atividade é criado o Modelo Independente de Plataforma (PIM), como por exemplo, um diagrama de classe genérico sem detalhes de uma linguagem específica. Em seguida na atividade "Gerar Projeto de Baixo-Nível" o PIM modelado passa por uma transformação automática dando origem um Modelo Específico de Plataforma (PSM). É nesta etapa que se define a plataforma que será aplicada para geração do ambiente virtual. Após a criação do PSM é possível "Converter para Código" o modelo gerado através de uma nova transformação automática de modelo para código. Se houver a necessidade de melhorar o código enriquecendo a implementação dos métodos com código que não foram possíveis de ser gerados através das

transformações, aplica-se a atividade “Aprimorar Código Manualmente” para que o código seja aprimorado e atenda a especificação do software. Ao fim do fluxo tem-se como resultado o código gerado.

Na sequencia são descritas cada uma das atividades e artefatos presentes nas abordagens MDA que compõem o metamodelo proposto.

3.2.1 Abordagem de Transformação de Modelo

3.2.1.1 Especificação do Software

A Especificação do Software é o documento que apresenta tudo que é necessário para o que o software seja implementado. No caso desta proposta, através deste documento é possível identificar os requisitos do sistema e determinar os recursos e funcionalidades necessários para constituir um ambiente virtual. São apresentadas as características do ambiente que se deseja desenvolver, os elementos físicos e virtuais que atuam no ambiente, assim como todo o suporte necessário para viabilizar o ambiente virtual pretendido, como, por exemplo: dispositivo para captura de imagem, dispositivos de visualização e iteração, motores, sensores e as bibliotecas de realidade virtual e/ou aumentada necessários para criar o cenário que se pretende simular.

Esta descrição do entendimento do sistema apresentado na especificação pode ser composta por modelos textuais e/ou gráficos. No entanto neste trabalho não será tratada a elaboração destes documentos de especificação, optou-se por não definir quais os padrões devem ser adotados para a geração dos documentos que compõem o levantamento inicial, pois não será analisado nenhum processo de conversão automática e de rastreabilidade de modelos gerados no levantamento requisitos. Fica a critério do analista de negócio/entidade estabelecer quais os documentos devem ser criados nesta fase inicial.

Apesar deste trabalho não abordar o processo de criação dos documentos da especificação, vale ressaltar que os artefatos gerados nesta fase são a base para o desenvolvimento do software e que as próximas fases adotam padrões de modelagem previamente estabelecidos para que seja possível realizar transformações da MDA, ou seja, deve fornecer um mecanismo para que os projetistas realizem a modelagem padronizada e livre de ambiguidades. Assim as especificações dos requisitos levantados serão refinadas na fase de análise dando origem ao modelo PIM.

3.2.1.2 Análise de Entidades

Na atividade de Análise de Entidades são construídos os modelos de alto nível empregado dentro da Abordagem de Transformação de Modelos. Estes modelos apresentam detalhes independentes de implementação, ou seja, independente de uma tecnologia ou linguagem específica. De acordo com a OMG, um modelo UML pode ser independente de plataforma (PIM) representar a funcionalidade e comportamento do negócio com muita precisão, mesmo sem incluir aspectos técnicos de uma tecnologia específica.

Com base nas informações da especificação, são construídos os diagramas estruturais (Classes, Pacotes, Componentes, etc.) das classes dos elementos que poderão compor um ambiente, através da uma ferramenta de suporte a MDA. O produto desta etapa é o PIM, modelo com alto nível de abstração e independente de qualquer tipo de plataforma. Nesta fase propõe-se a criação das classes dos objetos que poderão ser instanciados e farão parte do ambiente que se deseja criar, ou a reutilização de classes de objetos e elementos já existentes.

No caso de criação de novas classes, elas devem ser criadas de acordo com a análise das especificações dos objetos, todas as características devem ser transcritas para o modelo, seus atributos, métodos e relacionamentos. A Figura 22 apresenta uma classe da biblioteca de classes construída nesta primeira etapa de modelagem independente de plataforma.

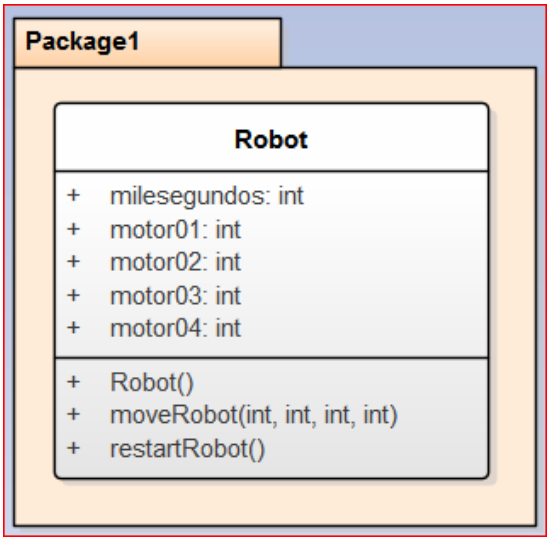


Figura 22 - Exemplo de uma classe do PIM.[Próprio Autor]

Apesar de a UML oferecer suporte para modelar o comportamento dos objetos como: Diagramas de Sequência, Estados, Atividades, Colaboração, etc., as

ferramentas de suporte a abordagem MDA não fornecem suporte para transformações com estes diagramas. É possível adotar ferramenta comercial ou desenvolver um software específico para converter estes diagramas, que poderá oferecer códigos parciais dos métodos das classes, caso o usuário do metamodelo venha ter este recurso.

3.2.1.3 Gerar Projeto de Baixo Nível

Nesta atividade é definida uma plataforma específica, ou seja, o diagrama de classe anteriormente genérico passa por uma transformação automática que tem como resultado o PSM de uma linguagem específica.

Para realizar esta transformação de PIM para PSM é necessário ter definido qual a plataforma será utilizada na transformação, pois esta definição é que orienta o mapeamento necessário para a transformação entre modelos. São inseridos métodos *defaults*, porém agora com todas as características específicas de linguagem definida na transformação de PIM para PSM, definições dos parâmetros e outros detalhes que se fazem necessário que estejam presente no código. A Figura 23 apresenta um exemplo do resultado da transformação do PIM da biblioteca de classe para PSM onde plataforma definida foi à linguagem Java.

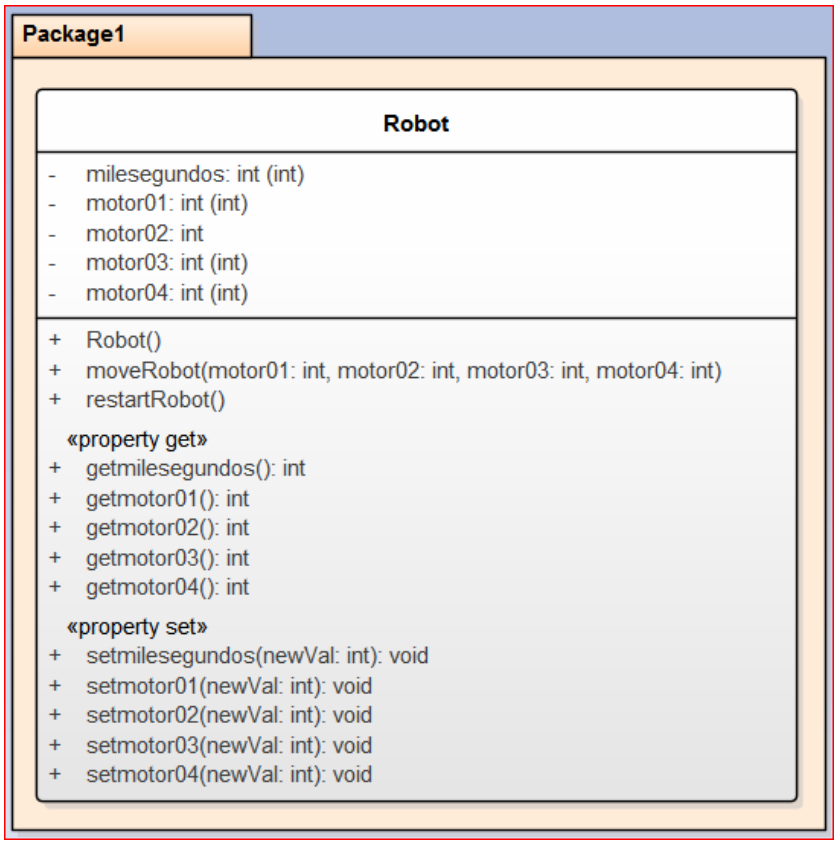


Figura 23 – Parte do PSM. Classe da biblioteca de classes após transformação para linguagem Java. [Próprio Autor]

Os modelos e as transformações apresentadas são realizados por meio de ferramenta comercial com suporte a MDA similar as ofertadas no mercado, porém, estas mesmas transformações podem ser realizadas de forma manual, respeitando todas as regras de transformação exigidas pela MDA, de uma forma que possibilite o mapeamento de todos os elementos entre os resultados das transformações. Outra forma de realizar estas transformações seria por meio de criação de ferramentas próprias, que assim como da forma manual, atendessem todas as regras pertinentes exigidas pela MDA. Porém ambas as formas de transformação anteriormente apresentadas tornam-se inviáveis quando se considera o custo temporal e financeiro. O esforço empreendido para realizar a transformação manual ou para a criação de ferramenta de transformação seria bastante grande e exigiria um período de tempo.

As transformações são responsabilidade da ferramenta MDA, ao modelador cabe à tarefa de modelar o software e determinar no momento certo qual a tecnologia será utilizada para geração do código-fonte. As transformações e a criação dos mapeamentos são de responsabilidade da ferramenta que fornece suporte a transformação de MDA, este processo pode ser considerado uma caixa preta, o que é importante são os resultados obtidos com as ferramentas.

3.2.1.4 Converter para código

A conversão para código é a etapa onde a partir do PSM da biblioteca de classes o código fonte é gerado automaticamente para a linguagem definida na etapa de Projeto de Baixo-Nível. A fase de conversão de modelo em código é a ultima da Abordagem de Transformação de Modelo. A Figura 24 apresenta o resultado da geração de código a partir do modelo PSM da biblioteca de classes da Figura 23. É possível observar que os elementos textuais da representação gráfica da classe "*Robot*" quando transformados são transcritos para código da plataforma especificada na fase de transformação de PIM para PSM. No exemplo da Figura 24 a imagem do código, na linha três tem-se o inicio da criação da classe *Robot* que finaliza na linha trinta e seis:

```
3   public class Robot {...
36  }
```

Também é apresentado nas linhas de quatro a oito as variáveis representadas no PSM da classe *Robot*:

```
4   private int milliseconds;
5   private int motor01;
```

6 private int motor02;

7 private int motor03;

8 private int motor04;

Nas linhas dezoito e dezenove do código da Figura 24 é apresentada a transcrição do método moveRobo:

33 public moveRobot(int motor01, int motor02,

34 int motor03, int motor04){

35 }

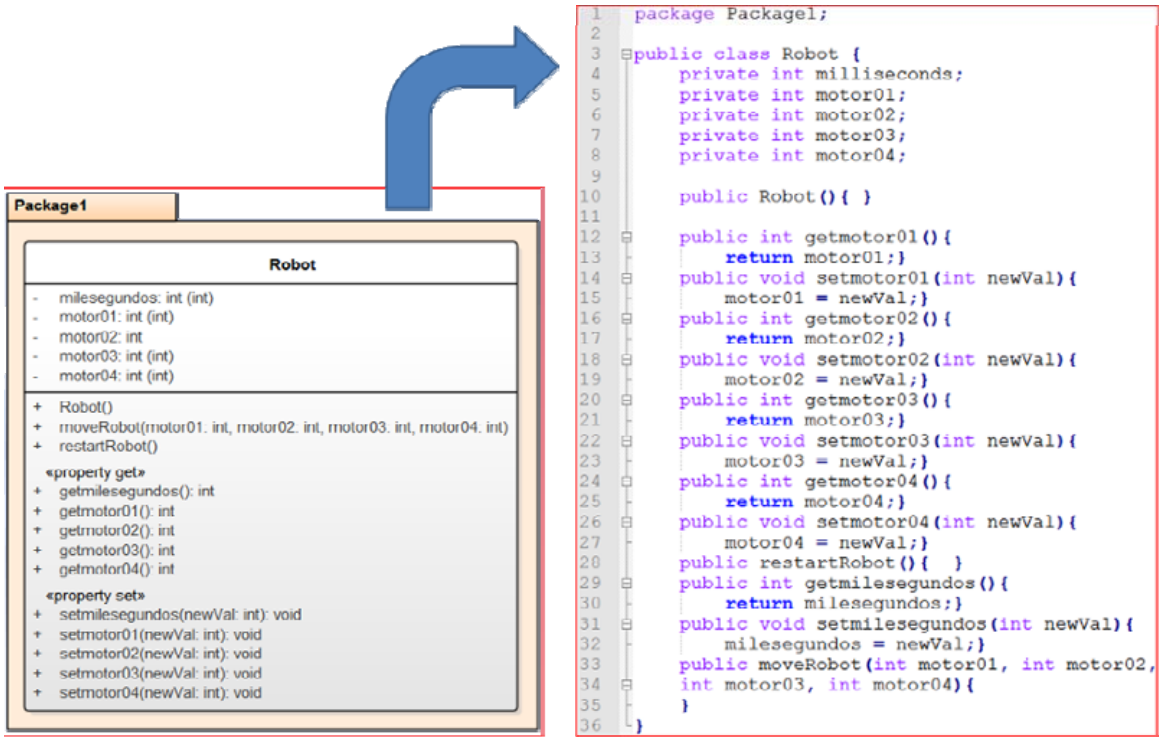


Figura 24 - Código Java gerado a partir das transformações da MDA. [Próprio Autor]

3.2.1.5 Aprimorar Código Manualmente

Está é uma atividade facultativa, só é realizada quando o código fonte necessitar de alguma adequação em seus métodos e parâmetros. Nesta atividade de aprimoramento de código métodos com características específicas são implementados nas classes. Este refinamento do código é necessário porque alguns métodos possuem especificidades como portas de comunicação, funções específicas de alguma *interface* de

controle que não é possível representar modelos UML, e que por sua vez não puderam passar por transformações automáticas para geração de código.

O código gerado pode ser compilado/interpretado por uma IDE específica da linguagem definida nas atividades anteriores, esta compilação pode dar origem a um arquivo de biblioteca de classes dos elementos e objetos que poderão ser instanciados no ambiente desejado.

O código-fonte da biblioteca de classes é um dos artefatos para Abordagem de Interpretação de Modelo. Nesta abordagem utiliza-se um *Engine* para execução de modelos criados e transformados na Abordagem de Transformação de Modelo, juntamente com a especificação do software que servirão de base para os modelos que serão criados na Abordagem de Interpretação de Modelo.

3.2.2 Abordagem de Interpretação de Modelo

3.2.2.1 Criar Objetos

Esta é a primeira atividade da Abordagem de Interpretação de Modelo que compõe o metamodelo proposto neste trabalho. Nesta etapa são selecionados os elementos que comporão o ambiente, isto é, os objetos que serão instanciados para que o ambiente virtual seja gerado. Para isto é analisada a Especificação do Software e a Biblioteca de Classes.

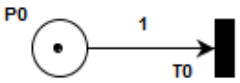
Uma forma de elencar os objetos do ambiente é através da modelagem do diagrama de objetos da UML. Uma alternativa pode ser utilizar um editor específico capaz de elencar os elementos necessários para criação do cenário. Ambas as opções devem ser capazes também de criar um arquivo .xml com os elementos, objetos do ambiente.

Esta atividade necessita da biblioteca de classe com a coleção de classes com seus atributos e métodos que poderão ser instanciados em tempo de execução e os documentos da especificação do software.

3.2.2.2 Análise Dinâmica

A atividade de análise da dinâmica de funcionamento do ambiente dá origem à modelagem comportamental do ambiente, que é modelada através de Rede de Petri. Nesta etapa é criado o modelo que descreve a dinâmica do comportamento do ambiente que

se deseja gerar de acordo com o comportamento e funcionalidade que cada objeto deve realizar. Para a criação do modelo de RdP utiliza-se um Editor de Rede de Petri que disponha entre seus recursos a possibilidade de salvar o modelo criado em formato de arquivo .xml como no exemplo apresentado na Figura 25. A análise dinâmica tem o objetivo de entender todo o comportamento do ambiente a ser gerado e representar este comportamento através da modelagem na RdP.




```
1   <?xml version="1.0" encoding="ISO-8859-1"?><pnml>
2   <net id="Net-One" type="P/T net">
3     <token id="Default" enabled="true" red="0" green="0" blue="0"/>
4   <place id="P0">
24  <transition id="T0">
50  <arc id="P0 to T0" source="P0" target="T0">
63    </net>
64    </pnml>
```

Figura 25 - Estrutura básica do arquivo .xml de um modelo de RdP. [Próprio Autor]

Neste modelo podem-se empregar as nomenclaturas para lugares e transições fornecidas pelo editor (exemplo *P0*, *P1*...*Pn* para lugares e *T0*, *T1*...*Tn* para as transições) como apresentado na Figura 25. No entanto para facilitar a tarefa de configuração, recomenda-se empregar nomes sugestivos aos objetos e funcionalidades representados no modelo, como mostrado na Figura 26.


```
1   <?xml version="1.0" encoding="ISO-8859-1"?><pnml>
2   <net id="Net-One" type="P/T net">
3     <token id="Default" enabled="true" red="0" green="0" blue="0"/>
4   <place id="P0">
24  <place id="P1">
44  <transition id="Iniciar Autenticacao">
70  <arc id="Iniciar Autenticacao to P1" source="Iniciar Autenticacao " target="P1">
83  <arc id="P0 to Iniciar Autenticacao" source="P0" target="Iniciar Autenticacao ">
96    </net>
97    </pnml>
```

Figura 26 - Substituição da nomenclatura padrão por nomes específicos. [Próprio Autor]

Os métodos e parâmetros são associados às transições do modelo de RdP moldando assim os respectivos comportamentos e características descritas na especificação do software. Como por exemplo, a transição "Iniciar Autenticacao" será associada com a funcionalidade fazerLogin() do Objeto como apresentado na Figura 26.

As associações são realizadas através do relacionamento das transições da RdP com os métodos dos objetos elencados no diagrama de objetos, que serão instanciados em tempo de execução para geração do ambiente virtual. Uma transição pode ser associada a mais de um método do mesmo objeto ou de objetos diferentes, a rede também permite disparar mais de uma transição, ou seja, executar mais de um método simultaneamente.

3.2.2.3 Executar Modelo de RdP

Esta atividade é a ultima da Abordagem de Interpretação de Modelo, ela é responsável pela interpretação dos modelos por meio de arquivos .xml. Um *Engine* implementada em uma plataforma específica utiliza o modelo RdP, o digrama de objetos e a biblioteca de classes para gerar o ambiente virtual inicialmente planejado. O *Engine* é uma ferramenta construída em uma linguagem de programação específica capaz de entender e executar tanto as regras de uma RdP como a estrutura de classes com seus métodos e parâmetros.

4 APLICAÇÃO DA PROPOSTA DE DESENVOLVIMENTO ORIENTADO POR MODELO

Para validar a proposta deste trabalho é apresentada neste capítulo sua aplicação no desenvolvimento e geração de um ambiente virtual, especificamente de Realidade Aumentada-RA.

No ambiente de RA os objetos virtuais e reais coexistem, trabalham e interagem. O ambiente de RA proposto neste trabalho é composto de um braço robótico real, uma rampa virtual, uma caixa virtual e os marcadores de RA que servem de referencia para a renderização dos objetos 3D. A dinâmica do ambiente utiliza os elementos reais e virtuais para realizar uma tarefa, como se fosse um único ambiente. O braço robótico real que coleta uma caixa virtual e a leva ao ponto mais alto da rampa virtual. O braço robótico deixa a caixa na parte superior da rampa virtual, o objeto caixa então desce da extremidade mais alta da rampa até a parte mais baixa pela força da gravidade, quando a caixa chega novamente na parte inferior da rampa, o robô coleta novamente dando continuidade ao ciclo.

Primeiramente, são apresentados os materiais e métodos empregados para o desenvolvimento deste ambiente especificado acima.

4.1 Recursos utilizados nesta proposta de metamodelo MDA

O metamodelo de MDA proposto neste trabalho apresentado na Figura 21, requer algumas ferramentas de suporte a transformação de modelos. Assim, para viabilizar este metamodelo é necessário utilizar ferramentas existentes (proprietária e/ou *opensorce*) e/ou outras que podem ser construídas conforme necessidade, para que seja possível visualizar todos os processos descritos, passando pela Abordagem de Transformação de Modelo até a geração do ambiente virtual na Abordagem de Interpretação de Modelo.

4.1.1 Ferramenta CASE para modelagem UML

A criação e a transformação dos modelos deste trabalho dentro desta abordagem são fundamentadas através do MDA do OMG com modelos que serão expressos em uma linguagem baseado no *Meta Object Facility* (MOF). O MOF usa metamodelos

especificados em UML para descrever a linguagem de modelagem, assim como o inter-relacionamento dos objetos.

A ferramenta de desenvolvimento de MDA utilizada neste trabalho é a *Enterprise Architect* (EA) da Sparx, que segunda a OMG e a própria fornecedora da ferramenta, suporta o MOF.

Conforme o OMG um modelo UML pode ser independente de plataforma ou específico, ou seja, cada padrão ou aplicação MDA é baseada, normativamente, em um Modelo Independente de Plataforma (PIM), que quando transformado de acordo com mapeamentos padronizados pela OMG produz um ou mais Modelos Específicos de Plataformas, (PSMs). Na próxima etapa, a ferramenta gera o código a partir do PSM.

Com esta ferramenta é possível trabalhar com os dois níveis de abstração de modelos existentes na abordagem MDA e a geração automática de código. Ela permite a transformação de modelos para várias tecnologias diferentes, além de fornecer suporte para criar novas regras para novas plataformas tecnológicas. A EA trabalha com engenharia avante e também fornece suporte a engenharia reversa. Com ela é possível criar modelos de negócio, *workflows* e diagramas da UML, é uma ferramenta que fornece apoio ao desenvolvimento de *software*.

Nesta proposta é utilizada uma versão *Trial* fornecida pelo site da empresa. Com a EA é criado o PIM na atividade Análise de Entidades, o PSM da atividade Gerar Projeto de Baixo Nível e o código gerado na atividade Converter para código. Atividades estas pertencentes à Abordagem de Transformação de Modelo. Uma das principais vantagens da adoção desta ferramenta para o projeto é devido o suporte ao armazenamento dos modelos em arquivos .xml.

4.1.2 Ferramenta para modelagem de Rede de Petri

Para modelagem de RdP é utilizada o editor *Platform Independent Petri net Editor* (PIPE), uma ferramenta *opensource* desenvolvida em Java criado por um grupo de projetos do *Deparment of Computing*, *Imperial College London*, e permite a realização de análises manuais de funcionamento do modelo, tem *interface* amigável e apresenta rapidez na execução de testes nos modelos criados. Outro ponto positivo é possibilidade de salvar os modelos criados em formatos de arquivos .xml. Ela é aplicada na atividade Análise Dinâmica,

para construção do modelo comportamental do ambiente. A Figura 27 apresenta como deverá ser estruturado o arquivo .xml gerado pelo editor de RdP.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?><pnml>
2  <net id="Net-One" type="P/T net">
3   <token id="Default" enabled="true" red="0" green="0" blue="0"/>
4  <place id="P0">
24 <transition id="T0">
50 <arc id="P0 to T0" source="P0" target="T0">
63  </net>
64  </pnml>
```

Figura 27 - Estrutura básica do arquivo .xml de um modelo de RdP. [Próprio Autor]

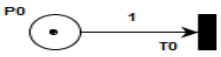


Figura 28 - Representação gráfica de RdP da Figura 27.

Analisando as figuras anteriores é possível identificar em ambas os elementos da RdP. Os elementos, lugar (*place*) *P0*, transição (*transition*) *T0* e o arco (*arc*) "*P0 to T0*" que indica sua origem (*source*) *P0* e o destino (*target*) *T0* da representação gráfica são encontrados nas *tags* (marcações) do arquivo xml.

```
4  <place id="P0">
5  <graphics>
6   <position x="45.0" y="30.0"/>
7   </graphics>
8  <name>
9   <value>P0</value>
10 <graphics>
13  </name>
14 <initialMarking>
15  <value>Default,1</value>
16 <graphics>
19  </initialMarking>
20 <capacity>
23  </place>
```

Figura 29 - Dados contidos na *tag place*. [Próprio Autor]

Quando a *tag place* é expandida como apresentado na Figura 29, podem-se identificar os dados relacionados ao lugar *P0*, como seu posicionamento no painel de edição, que no exemplo indica x= "45.0" e "y= 30.0". Na linha 9 encontrasse o nome do lugar *P0* e na linha 15 encontrasse logo após o valor *Default*, o número 1, este número indica a quantidade

de marcas (*tokens*) do lugar *P0*. Estes dados do lugar serão carregados e utilizados pelo *Engine* na configuração e na execução do modelo RdP.

Quanto às transições, a Figura 30 apresenta os dados relativos à transição *T0*, dentre os dados contidos nas *tags* de transição, destaca-se a *position* e *name*. *Position* contem as coordenadas da transição com relação à área de edição dos modelos, que no exemplo possui os valores x= “135.0” y= “30.0”. A *tag name* possui o valor relacionado ao nome da transição, que no caso do exemplo é *T0*. Estes são os dois dados de todas as transições do modelo, que serão utilizados pelo *Engine*, tanto para configuração como para a execução do modelo.

```
24  <transition id="T0">
25  <graphics>
26    <position x="135.0" y="30.0"/>
27  </graphics>
28  <name>
29    <value>T0</value>
30  <graphics>
33  </name>
34  <orientation>
37  <rate>
40  <timed>
43  <infiniteServer>
46  <priority>
49  </transition>
```

Figura 30 - Dados da *tag transition*. [Próprio Autor]

O elemento arco da RdP também é comtemplado pelo arquivo .xml, como apresentado na Figura 31. Os dados relevantes para o *Engine* é o *id* do arco, que no exemplo da figura possui o valor *id= "P0 to T0" source= "P0" target= "T0"*, a o segundo valor da *tag inscription*, na figura apresenta logo após o valor “*Default*” o valor “1”, este valor indica o peso do arco, que conjuntamente com um elemento lugar como entrada pode determinar se uma transição está ou não habilitada e quantas marcas deverá ser consumida do lugar de entrada/origem, ou quando conjuntamente com um elemento transição com saída pode determinar a quantidade de marcas que o lugar de destino receberá a após o disparo da transição.

Outros dados relevantes para *Engine* se encontra no exemplo da Figura 31 nas linhas 59 e 60, estas linha indicam a posição inicial e final do arco no painel de edição do modelo. Na linha 59 logo após o *id= “000”* encontram-se os valores *x= “71” y= “42” curvePoint= “false”* que indicam a coordenada inicial do arco e o *curvePoint= "false"* indica

que não existe pontos de curva na parte inicial do arco. A linha 60 com *id= "001"* apresenta os dados da coordenada final do arco dentro do editor de RdP, neste caso os valores são *x= "141" y= "42"* para coordenada final e *false* para existência de ponto de curva na parte final do arco.

```
50  <arc id="P0 to T0" source="P0" target="T0">
51  <graphics/>
52  <inscription>
53  <value>Default,1</value>
54  <graphics/>
55  </inscription>
56  <tagged>
59  <arcpath id="000" x="71" y="42" curvePoint="false"/>
60  <arcpath id="001" x="141" y="42" curvePoint="false"/>
61  <type value="normal"/>
62  </arc>
```

Figura 31 - Dados da *tag arc*. [Próprio Autor]

Estas características estruturais devem ser comtempladas pelo arquivo .xml gerado pelo editor modelo de RdP. Portanto o Editor de RdP deve ser capaz de trabalhar com o formato *Petri Net Markup Language-PNML* [32], que é baseado em *Extensive Markup Language- XML* [33], que é um formato XML especifico para RdP. Pode-se perceber na primeira e ultima linha do código apresentado na Figura 31 a notação *pnml*, indicando que arquivo segue o formato padrão PNML.

Após um estudo inicial de ferramentas editoras de RdP chegou-se a ferramenta *open source* PIPE (*Platform Independent Petri Net Editor*) como a mais adequada à esta proposta. É um editor que além da modelagem gráfica permite a simulação do modelo, fornecendo com isso a possibilidade de testar o modelo antes mesmo de ser posto realmente em uma execução direta e também pelo fato de que este editor suporta o formato *PNML*.

4.1.3 Ambiente de Desenvolvimento Integrado (IDE-Integrated Development Environment)

A ferramenta de compilação de código utilizada para realização deste estudo de caso é a IDE Eclipse, uma ferramenta *opensource* utilizada para desenvolvimento de software, baseada em Java. Neste trabalho ela é utilizada na atividade Aprimoramar do Código e que recebe o código estrutural elaborado no EA para proceder com a programação do conteúdo interno de alguns métodos específicos como o de comunicação com arquivos .dll

específicos para o controle dos motores. A IDE ainda gera a Biblioteca de Classes, que nesta aplicação da proposta é um arquivo .jar utilizado para criar os objetos, e também é entrada para *Engine*.

4.1.4 Editor de Objetos do Cenario – EOC

Para elencar os objetos que farão parte do ambiente virtual deve-se utilizar uma ferramenta de modelagem com suporte a geração de arquivos .xml como por exemplo o EA. Porem optou-se neste trabalho pela criação da própria ferramenta de modelagem visual com recursos intuitivos de seleção de objetos. A ferramenta é denominada de EOC (Editor de Objetos do Cenário), construída em linguagem Java, capaz de elencar os elementos que serão instanciados no ambiente gerar, além de suportar leitura de arquivo .jar e escrita de arquivo .xml que representa o modelo dos objetos modelados na ferramenta, necessários para gerar o ambiente virtual especificado.

Para utilizar a ferramenta é necessário carregar o da biblioteca de classes, em seguida serão exibidos na lateral esquerda da janela da ferramenta ícones representantes de cada classe. Estes ícones foram atribuídos a cada uma das classes na atividade de aprimoramento do código, para proporcionar ao usuário um efeito intuitivo das classes disponíveis, porém, não é obrigatório que as classes tenham ícones, a falta deste não afetará a geração do o ambiente virtual.

Para criar um objeto no ambiente basta clicar com o botão esquerdo do mouse sobre o ícone e arrastar para área de edição e soltar o botão. Quando o ícone é solto surge uma opção de adicionar um nome a este objeto. Este nome de objeto, assim como a classe a qual pertence será utilizado na atribuição dos métodos das classes às transições da RdP.

Por fim, o editor tem a opção de salvar a modelagem dos objetos criados como um arquivo .xml com os dados de cada objeto elencado.

A Figura 32 apresenta o EOC já com o um arquivo de biblioteca de classes carregado e com dois objetos já selecionados e nominados.

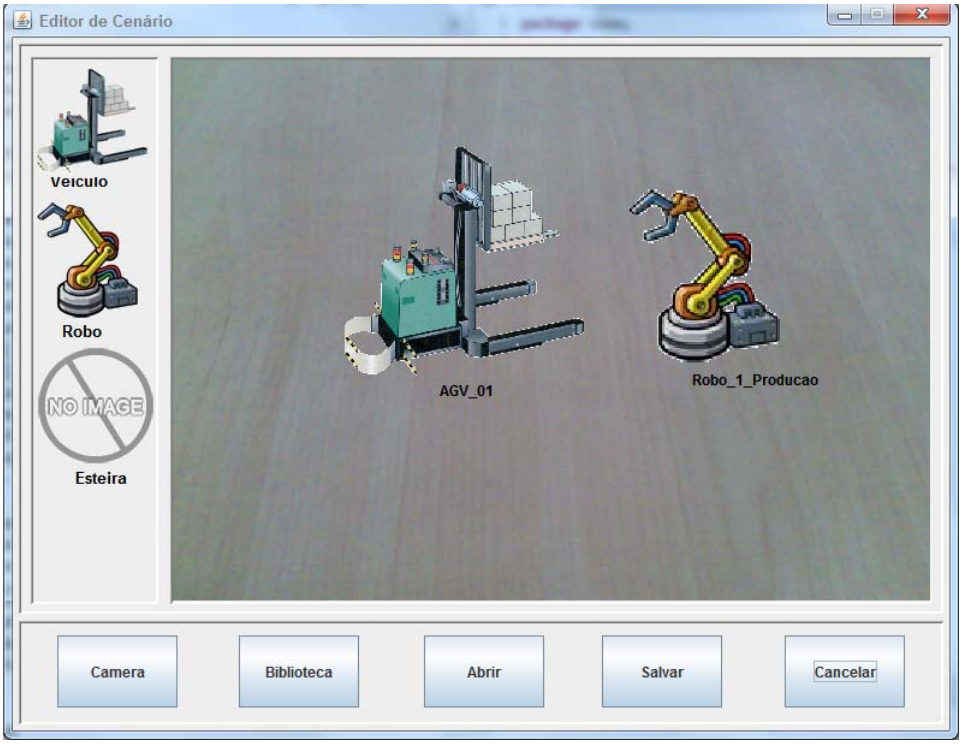


Figura 32 - Editor de Cenário – EC. [Próprio Autor]

O funcionamento do EOC ocorre da seguinte forma. Partindo do princípio de que uma câmera já esta instalada no computador, o primeiro passo é acionar a câmera pressionando o botão “Camera” na *interface* da ferramenta, isto acionara a câmera e iniciará a exibição de sua imagem no painel central do EOC.

Após a câmera liga é necessário carregar a biblioteca de classes que contem as classes que poderão ser utilizadas para criar os objetos do ambiente. Para carregar o arquivo pressiona-se o botão Biblioteca na parte inferior da ferramenta, isso abrirá uma janela para escolha do arquivo. Quando o arquivo for carregado na ferramenta surgirão os ícones relacionados às classes que estão contidas no arquivo. No exemplo da Figura 32 foram carregadas três classes como pode ser observado através dos ícones.

Na sequência primeiro ícone representa a classe veículo, o segundo a classe robô e o terceiro é a representação de uma classe a qual não possui um ícone que a represente. Assim ocorrerá com outras classes na mesma situação, que não possuam um ícone.

Para elencar os elementos que comporão o ambiente, é necessário clicar com o botão esquerdo do mouse sobre o ícone da classe desejada e arrasta para a posição desejada. No exemplo da Figura 32 foram arrastados os ícones da classe Veiculo e classe Robo. Assim que os ícones são soltos no painel tem-se a opção de adicionar um nome a este

objeto selecionado. Estes objetos selecionados serão instanciados em tempo de execução pelo *Engine*.

Após a criação de todos os objetos é necessário que este modelo de objetos seja armazenado no computador para que possa ser utilizado posteriormente. Isto é realizado através do botão salvar, que ao ser clicado abre a janela padrão do sistema operacional para que possa escolher o local e um nome para este arquivo. Quando esta ação é realizada é gerado um arquivo .xml que contem todos os dados necessários dos elementos que comporão o ambiente. A Figura 33 é apresenta a estrutura do arquivo gerado pelo EOC, no exemplo é possível verificar a existência dos dois objetos apresentado na Figura 33. Na linha 2 do arquivo inicia-se o corpo do arquivo que vai até as linha 17 onde é fechado o bloco principal, entre as linhas 3 e 9 e entre as linha 10 e 16 são apresentados os dois objetos que foram selecionados, dentro de cada bloco de objeto existe os dados de posição x e y da tela onde o objeto 3D será renderizado caso seja utilizada esta forma de renderização, o nome que o objeto recebeu e a classe a qual ele pertence.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ambiente id="AmbienteTeste7">
3    <objeto id="robo">
4      <graphics>
5        <position x="403" y="307" />
6      </graphics>
7      <name>Robo_1_Producao</name>
8      <classe>Robo</classe>
9    </objeto>
10   <objeto id="veiculo">
11     <graphics>
12       <position x="235" y="293" />
13     </graphics>
14     <name>AGV_1</name>
15     <classe>Veiculo</classe>
16   </objeto>
17 </ambiente>
```

Figura 33 - Estrutura do arquivo .xml gerado pelo EOC. [Próprio Autor]

4.1.5 Configurador de Ambiente

Esta é uma funcionalidade desenvolvida para relacionar as transições da RdP com os objetos elencados que faram parte do ambiente desejado, isto é, serão instanciados pelo *Engine* em tempo de execução conforme modelado na RdP. Recebe como entrada o arquivo .xml com o modelo de RdP criado pelo Editor de RdP na atividade “2.b-

Analise dinâmica do Ambiente", arquivo .xml do modelo criado pelo Editor de Cenário na atividade "6-Edição do Cenário do Ambiente" e o arquivo .jar criado na atividade "6-Compilação" a partir dos códigos gerados na Abordagem de Transformação de Modelo. A Figura 34 apresenta com um modelo de RdP ficar quando carregado no Configurador de Ambiente. Após todas as transições configuradas esta funcionalidade gera em memória uma RdP configura com todos os dados necessários para geração do ambiente especificado. Tem-se também a possibilidade de gerar um arquivo .xml com todos os dados configurados anteriormente, que pode ser carregado posteriormente no *Engine* para ser executado, desde que todos os dados constantes no modelo estejam presentes no computador conforme configurado no modelo RdP.

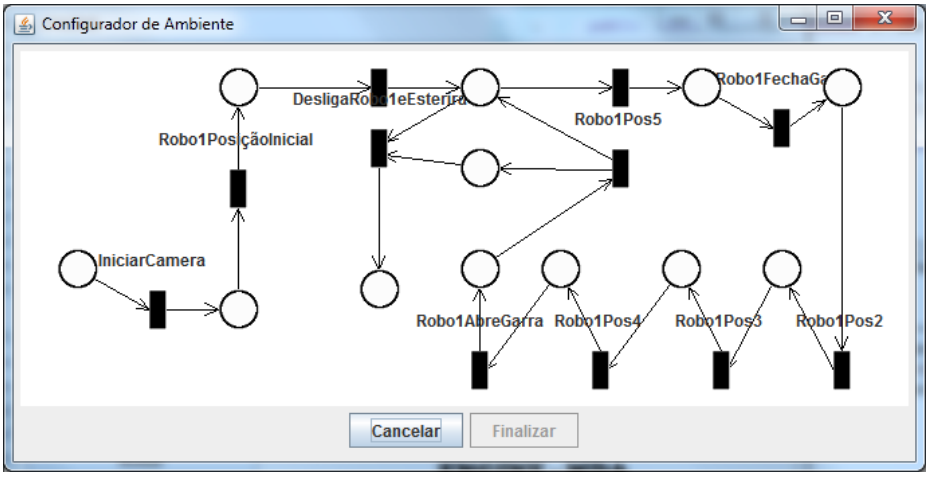


Figura 34 - Configurador de Ambiente. [Próprio Autor]

Para configurar a transição click sobre seu desenho com o botão esquerdo do mouse, uma nova janela será exibida como a apresentada na Figura 35.

Uma transição pode ser associada a mais de um objeto e também pode disparar mais de um método simultaneamente de um único objeto ou de objetos diferentes.

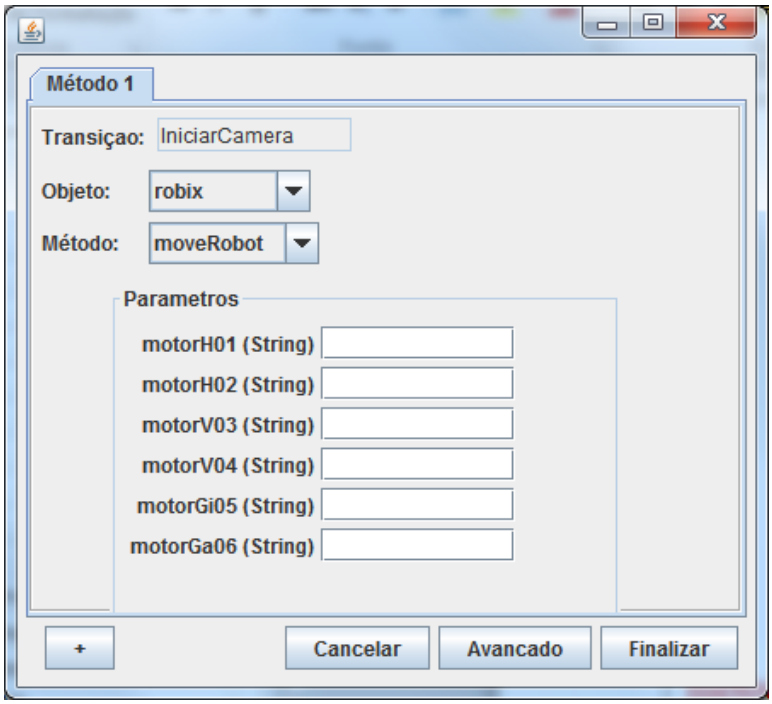


Figura 35 - *Interface* de configuração de Transição / Objeto(s) / Método(s) / Parâmetro(s). [Próprio Autor]

Na Figura 35 é possível observar que o campo “Transição” é preenchido automaticamente com o nome da transição que foi selecionada. Se for necessário adicionar outro método para que essa mesma transição o dispare, pressione o botão com o sinal de adição que fica no canto inferior esquerdo. Primeiro passo para realizar a configuração é a escolha do objeto, assim que o objeto é escolhido, a ferramenta busca os métodos que pertence ao objeto e os apresenta no campo método. Se o método escolhido possuir parâmetros, eles serão exibidos no painel “Parametros”.

A configuração de transição conta com um recurso para mediação de conflitos, este recurso pode ser acessado pressionando o botão “Avançado”. Esta opção não é obrigatória para o funcionamento da rede, porém, se existir algum conflito na RdP que necessite ser tratado, isto deverá ser realizando através desta função. Esta função apresenta as possibilidades resolução de conflito por meio de probabilidade ou prioridade. Situações de conflito na RdP ocorrem quando existem mais de uma transição que possuem um mesmo lugar como entrada.

Quando a configuração da transição estiver completa pressione o botão “Finalizar” para que os dados inseridos sejam armazenados em memória, a *interface* de

configuração será finaliza, permitindo assim possa realizar a configuração de outra transição, até que se configurem todas.

Sempre que uma transição é configurada a cor desta transição é alterada, a cor preta é trocada pela cor vermelha, isto serve para indicar quais as transições que foram e que faltam ser configuradas.

Quando todas as transições estiverem configuradas o botão “Finalizar” da *interface* “Configurador de Ambiente” passará para o estado habilitado, podendo então finalizar a configuração do modelo de RdP do ambiente.

4.1.6 Engine de Execução de Modelo RdP

O *Engine* é uma ferramenta interpretadora de modelos de RdP, desenvolvida em linguagem de programação Java pelos alunos Márcio de Abreu Moreira e Wilson Hissamu Shirado, mestrandos do curso de Pós-Graduação em Ciência da Computação da Universidade Estadual de Londrina. Desenvolvida para atender os conceitos de RdP, sendo capaz de interpretar um modelo de RdP e executa-lo segundo as regras conceituais de RdP ordinária.

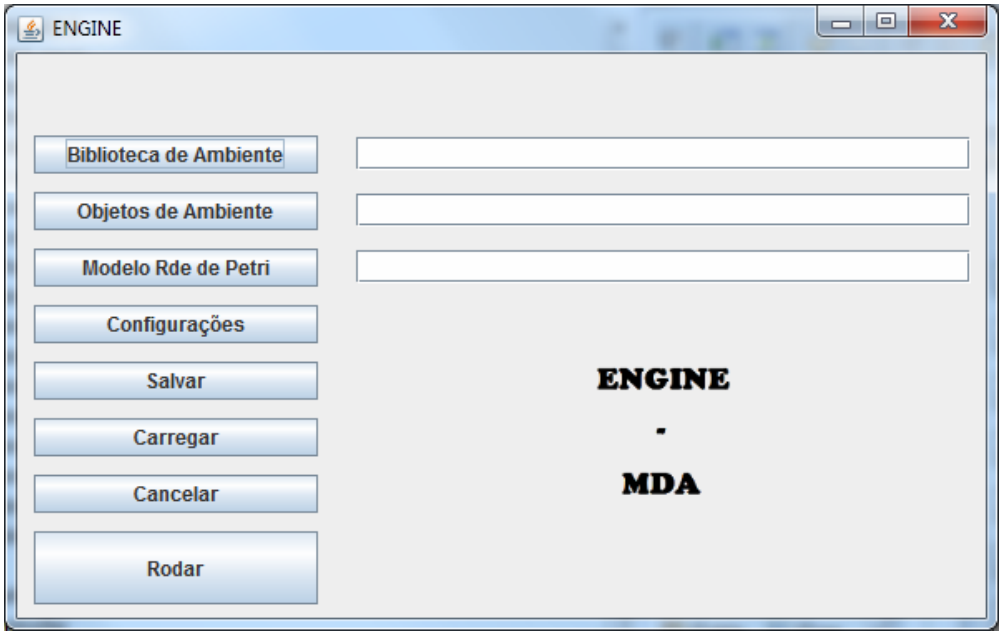


Figura 36 - *Interface* do *Engine*. [Próprio Autor]

Recebe como entrada um arquivo .xml de um modelo de RdP configurado no Configurador de Ambientes e um arquivo de Biblioteca de Classes gerado na Abordagem de Transformação de Modelo. Quando o *Engine* é executado, inicia uma série de verificações internas em todos os elementos da RdP, lugares, marcações, arcos e transições. Se identificada

a possibilidade de disparo da transição, o *Engine* dispara a transição e faz todas as alterações necessárias de acordo com as regras de uma RdP, como mudanças de *tokes* entre os *places* de acordo com os valores dos arcos. Isto acontece sucessivamente enquanto existir a possiblidade de disparo de transição.

A cada disparo de transição, uma ação é realizada, pois cada transição pode estar relacionada a um objeto e algum de seus métodos, isto reflete em uma reação no ambiente gerado.

Em geral o *Engine* é responsável por toda existência do ambiente, de acordo com o que é configurado no Configurador de Ambiente. Um exemplo é o disparo de uma transição "*X*" relacionada ao método "moverRobo()" do objeto Robo1, isto acarreta na realização do movimento do Robo1 para uma posição previamente configurada no Configurador de Ambientes. Outro exemplo é a transição "*Y*" relacionada ao método de acionamento da *webcam*, liga a *webcam* quando a transição é dispara. Quando não houver mais transições habilitadas para disparo o *Engine* finaliza a execução.

4.1.7 Outros recursos envolvidos

Para geração do ambiente virtual além das ferramentas de apoio a abordagem MDA já apresentados também foram utilizados outros recursos descritos abaixo:

- Braço robótico é um modelo construído com o kit Robix RCS-6 fabricado pela empresa norte americana Rascal™ (Figura 37), possui seis servos-motores controlados por uma interface hardware que recebe os sinais do computador através da porta paralela e os transferem para os motores dando movimento ao robô. O robô possui um software de controle que utiliza arquivos .dll próprios e arquivos nativos do sistema operacional Windows. De acordo com os documentos do software do robô, controla sem utilizar seu software de controle. As funcionalidades/métodos do robô são encontradas no seu manual de utilização que é um arquivo .txt que acompanha o software de instalação do sistema de controle. Neste estudo de caso não serão utilizadas todas as funcionalidades apenas as que possibilitam movimento ao robô.

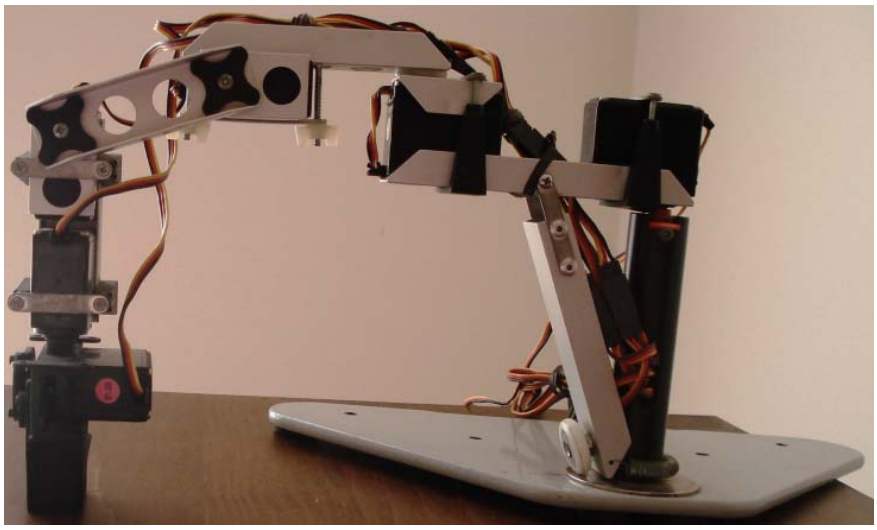


Figura 37 - Robix RCS-6. [Próprio Autor]

- Rampa tridimensional (Figura 38) de transporte por gravidade, um produto é colocado na sua parte superior e é descolado para parte inferior pela força gravitacional. Não possui motores para realizar sua função. É um objeto 3D com as características sólidas habilitadas, o que não permite transpassa-la. A abstração deste modelo 3D não recebe métodos e atributos, simplesmente é criada uma classe Rampa para que possam ser relacionada a diferentes objetos rampa de mesma classe, para que possam ser instanciados.

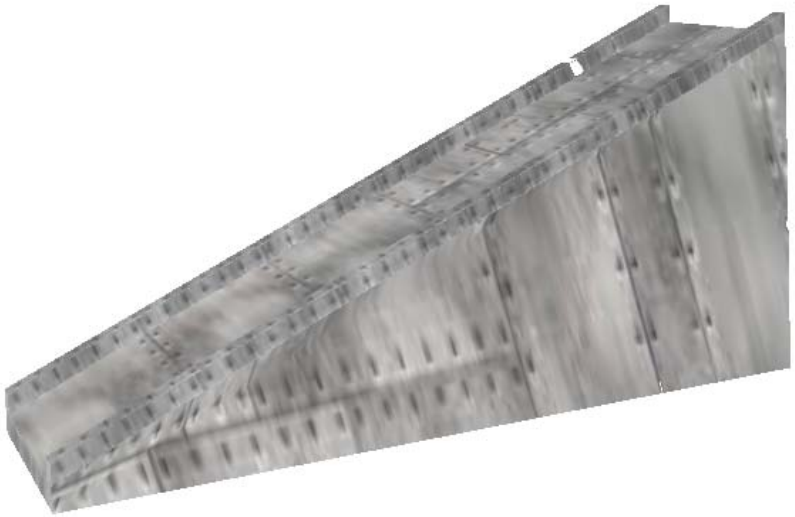


Figura 38 - Rampa 3D. [Próprio Autor]

- Caixa tridimensional (Figura 39) que representa um produto que será movido pelo braço robótico da base até o alto da esteira, o produto se deslocará da parte mais alta para a base da esteira devido à força gravitacional aplicada a estes objetos.

Figura 39 - Caixa 3D. [Próprio Autor]

- Blender 3D uma opção *opensource* que permite além da construção de objetos 3D, animação, aplicação de efeitos físicos, exportação para vários formatos incluindo o .vrml, que é o formato utilizado neste caso, dentre outras opções. Ambos os objetos tridimensionais foram construídos com a ferramenta de modelagem Blender 3D.
- Biblioteca de RA NyARToolkit [2] baseada na biblioteca ARToolKit [1]. A versão usada neste trabalho foi desenvolvida para linguagem Java. A NyARToolkit possui todas as classes para captura de imagem, identificação de marcadores e renderização de objetos 3D. Também são utilizados três marcadores padrão da biblioteca ARToolKit, patt.hiro e patt.kanji, e outro que faz parte da ferramenta SACRA [78], um sistema de autoria de realidade aumentada. As configurações de relacionamento dos objetos 3D com seus respectivos marcadores são feitas diretamente nos arquivos internos da biblioteca de RA, incluindo os parâmetros da webcam. A opção de utilizar marcadores ao invés das coordenadas do cenário é

devido a fato de que a RA com marcadores facilita o desenvolvimento, e a maioria das bibliotecas possui esta forma de interação como padrão, não exigindo desenvolvimentos adicionais.

4.2 Desenvolvimento do Sistema

Para desenvolvimento sistema é adotado o fluxo da Figura 21, que apresenta o metamodelo proposto neste trabalho para criação de ambientes virtuais, apoiado pelos recursos apresentados nas seções anteriores.

4.2.1 Especificação do Software – Ambiente de Realidade Aumentada

Nesta etapa inicial não é utilizado uma ferramenta específica, podendo ser registrado o texto em qualquer editor de texto, ou mesmo registrado em documentos escritos manualmente.

Para este estudo de caso é proposto à geração de um ambiente de Realidade Aumentada-RA. O ambiente é composto dois objetos virtuais (rampa e caixa) e um dispositivo real (braço robótico) que coexistem e interagem. O braço robótico real coleta a caixa virtual na parte baixa, no solo, por meio de um marcador e a solta na parte superior da rampa virtual. O objeto caixa então desce da extremidade alta da rampa até a mais baixa pela força da gravida, quando a caixa chega novamente na parte inferior da rampa o robô a coleta novamente dando continuidade ao ciclo.

4.2.2 Analise das Entidades

De acordo com a descrição do ambiente desejado é construído no EA um PIM com as classes que definem os elementos que comporão o ambiente. Nesta parte da aplicação do metamodelo é criada a representação estrutural através da modelagem de diagrama de classes das entidades. Cada classe é criada com seus atributos e métodos conforme necessário para viabilizar a geração do ambiente virtual, como por exemplo para o controle do robô como no exemplo apresentado na Figura 40.

Por ser um equipamento que utiliza arquivos .dll para sua execução e comunicação com a *interface* e *hardware*, é necessário a criação de uma interface entre as funções do arquivo .dll e os métodos da classe Robot.

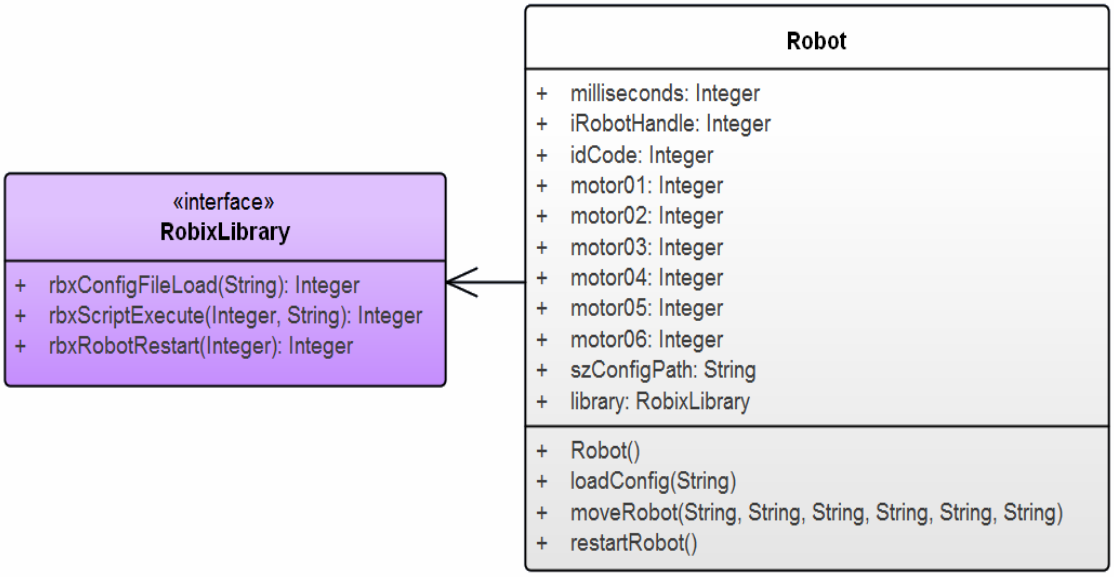


Figura 40 - PIM da Classe Robot e interface RobixLibrary. [Próprio Autor]

Para execução do ambiente de realidade aumentada é utilizada uma biblioteca de RA, que já possui todos os processos necessários de captura de imagem, detecção de marcadores, renderização dos objetos 3D, etc., dando suporte à criação do ambiente. Para utilizar esta biblioteca RA é criada uma classe capaz de fornecer todos os procedimentos necessários para este estudo de caso. Esta classe de invocadora a biblioteca é chamada de classe “Executora_RA”, que além do seu construtor possuí um método que instanciará as funções da biblioteca de RA.

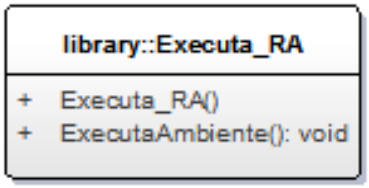


Figura 41 - Classe Executora_RA. [Próprio Autor]

4.2.3 Projeto de Baixo-Nivel

Com o PIM estrutural criado é possível submete-lo a transformação para PSM. Para este estudo de caso foi definida para transformação do modelo PIM a tecnologia específica Java, tendo em vista que, o *Engine* também foi desenvolvido na plataforma

específica Java para atender a Abordagem de Interpretação de Modelo em que os modelos são interpretados e executados por um *Engine* de um a tecnologia específica. Com o novo modelo gerado, para língua específica Java.

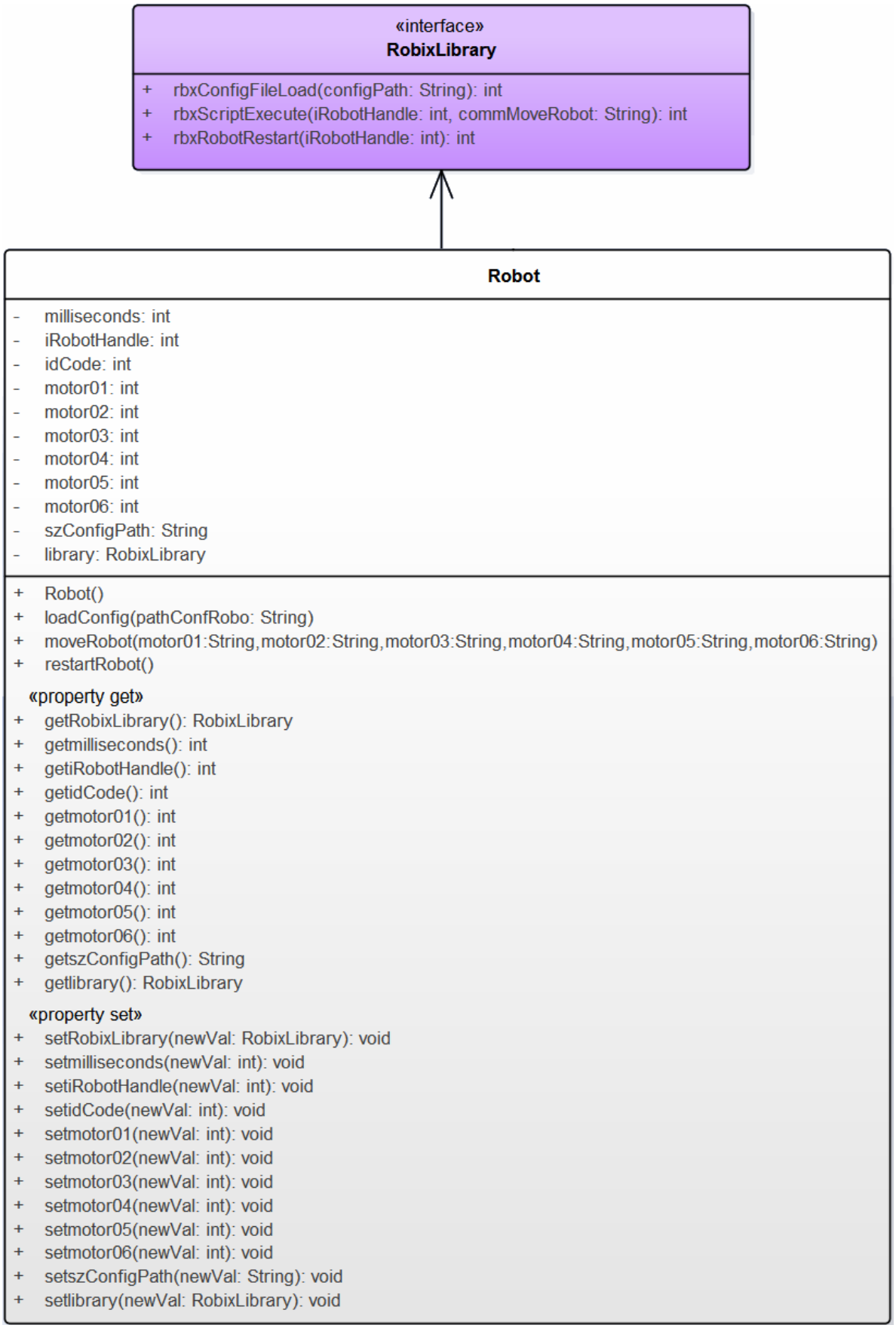


Figura 42 - PSM da Classe Robot e interface RobixLibrary. [Próprio Autor]

4.2.4 Converter para Código

Na etapa de codificação a ferramenta EA transformou automaticamente o modelo PSM gerado da transformação do PIM para PSM, em um código da linguagem específica Java.

```
1 package Library;
2
3 /**
4  * @author Marcio
5  * @version 1.0
6  * @created 18-fev-2015 04:29:27
7  */
8 public class Robot {
9
10     private int milliseconds;
11     private int iRobotHandle;
12     private int idCode;
13     private int motor01;
14     private int motor02;
15     private int motor03;
16     private int motor04;
17     private int motor05;
18     private int motor06;
19     private String szConfigPath;
20     private RobixLibrary library;
21
22     public Robot(){ }
23
24     public int getmilliseconds(){
25         return milliseconds;
26     }
27
28     /**
29      *
30      * @param newVal
31      */
32     public void setmilliseconds(int newVal){
33         milliseconds = newVal;
34     }
35
36     public int getiRobotHandle(){
37         return iRobotHandle;
38     }
39
40     /**
41      *
42      * @param newVal
43      */
44     public void setiRobotHandle(int newVal){
45         iRobotHandle = newVal;
46     }
47
48     public int getidCode(){
49     ...
```

```
135 ...
136  /**
137   *
138   * @param newVal
139   */
140  public void setszConfigPath(String newVal){
141      szConfigPath = newVal;
142  }
143
144  public RobixLibrary getlibrary(){
145      return library;
146  }
147
148  /**
149   *
150   * @param newVal
151   */
152  public void setlibrary(RobixLibrary newVal){
153      library = newVal;
154  }
155
156  /**
157   *
158   * @param pathConfRobo
159   */
160  public loadConfig(String pathConfRobo){
161
162  }
163
164  /**
165   *
166   * @param motor01
167   * @param motor02
168   * @param motor03
169   * @param motor04
170   * @param motor05
171   * @param motor06
172   */
173  public moveRobot(String motor01, String motor02,
174                   String motor03, String motor04,
175                   String motor05, String motor06){
176
177  }
178
179  public restartRobot(){
180
181  }
182 }//end Robot
```

Figura 43 - Trechos de código da Classe Robot. [Próprio Autor]

4.2.5 Aprimoramar do Código e Manualmente

Com o código criado pode-se utilizada tanto a ferramenta EA como a ferramenta IDE Eclipse para editar métodos que não foram completamente implementados através do modelo. Neste estudo de caso é utilizada a IDE Eclipse que é utilizada para realizar a compilação do código para um arquivo .jar.

Figura 44 - Trechos de código da Classe Robot com implementações realizadas na IDE Eclipse. [Próprio Autor]

4.2.6 Criar Objetos

Para realização desta atividade neste estudo de caso foi utilizada a ferramenta EOC que fornece recursos para geração do modelo de objetos de acordo com a especificação. O arquivo da biblioteca de classes é carregado no EOC que fornece recurso necessário para elencar os objetos que serão instanciados pelo Engine em tempo de execução, com foi apresentado na seção 4.1.4. Após a criação do modelo de objetos ele deve ser salvo em arquivo .xml com os dados dos elementos selecionados que serão configurados no Configurador de Ambientes.

4.2.7 Analise dinâmica do ambiente

Com o modelo de objetos criado analisa-se a especificação do ambiente virtual para criar o modelo de RdP que controla o comportamento de todos os elementos do ambiente. Para isto foram utilizadas as ferramentas PIPE para modelar a RdP e o Configurador de Ambientes para relacionar os objetos às transições da RdP.

Em um primeiro momento a RdP é modelada dando origem ao modelo apresentado na Figura 45 com todas as ações necessárias para que seja gerado o ambiente virtual especificado.

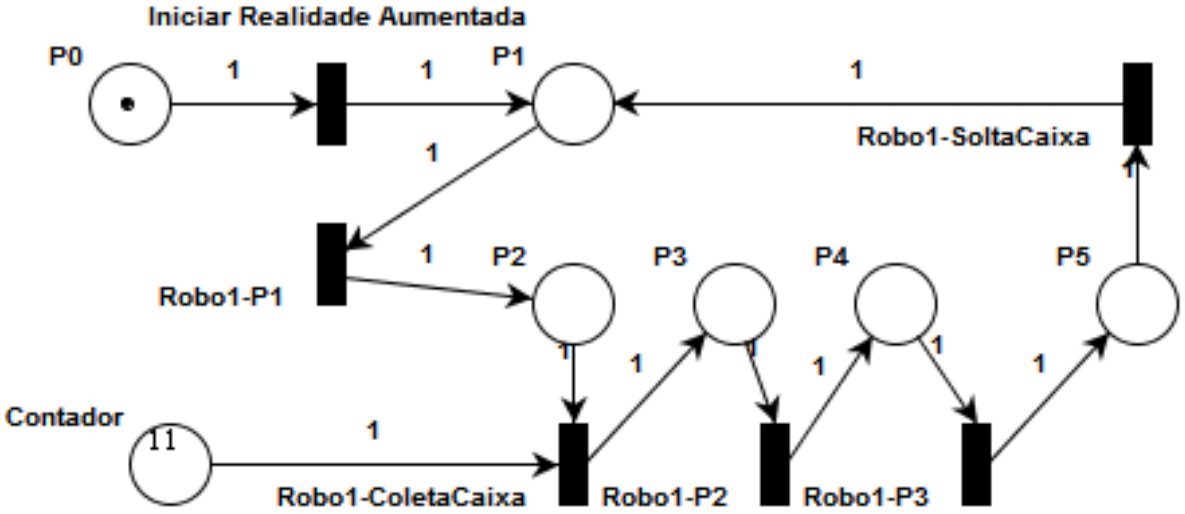


Figura 45 - PIM Comportamental do ambiente especificado. [Próprio Autor]

Após a montagem da RdP utilizou-se o Configurador de Ambientes para realizar a configuração das transições/objetos/métodos/parâmetros de cada transição.

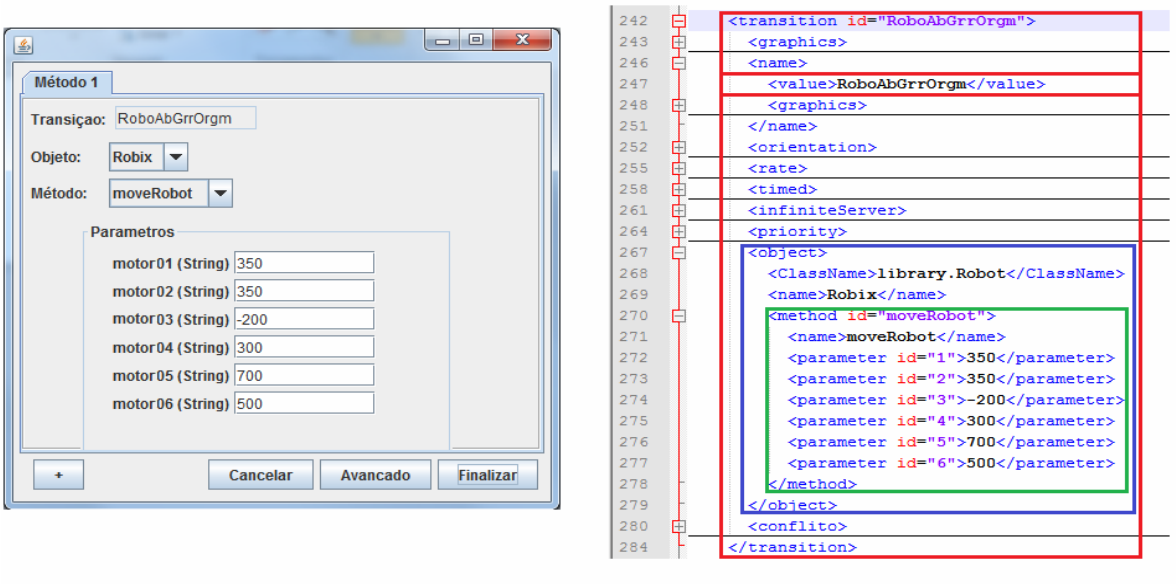


Figura 46 - À direita a interface de configuração de transição/objeto/metido/parâmetros, à esquerda trecho do arquivo .xml com configurações realizadas. [Próprio Autor]

Com todos os objetos devidamente relacionados às transições obteve-se o modelo apresentado na Figura 45, que segue o seguinte fluxo:

A transição “Iniciar Realidade Aumentada” quando disparada executa o método vinculado a ela que inicia todos os processos do funcionamento da câmera e biblioteca de RA, verificação dos marcadores e renderização dos objetos 3D sobre os marcadores. Em seguida, quando a transição “Robo1-P1” é disparada executa o método com os parâmetros inseridos na etapa de “Configuração de Ambiente”, esta ação envia um sinal para que o “Robo1” que o faz se ficar na posição inicial. Quando a transição “Robo1-ColetaCaixa” é disparada executa o método a ela relacionada que envia as coordenadas para o “Robo1” que fará com que realize a coleta do objeto virtual “caixa”. As transições “Robo1-P2” e “Robo1-P3” quando disparadas posicionam o “Robo1” nas coordenadas configuradas. Até que por fim quando a transição “Robo1-SoltaCaixa” é disparada o método a ela relacionada envia as coordenadas necessárias para quo o “Robo1” solte a “caixa” virtual, de acordo com as configurações realizadas na etapa de “Configuração de Ambiente”.

A transição “Robo1-ColetaCaixa” só será disparada se existir marcas nos lugares “*P2*” e “*Contador*”. Quando não existir marcas no lugar “Contador” a transição não estará apta a ser disparada, portanto finaliza a dinâmica do ambiente virtualizado.

4.2.8 Execução do Engine

Com o modelo RdP completamente configurado o *Engine* pode ser executado iniciando os processos contínuos de verificação do estado da RdP e disparo de transições habilitadas. É um processo cíclico que só termina quando não existir mais transições habilitadas. O resultado final é o apresentado na Figura 47, onde é possível observar todos os elementos virtuais e reais que foram elencados no modelo de objetos. Cabe a observação de que quando os objetos estão sendo elencados não existem uma diferenciação entre reais e virtuais, todos são tratados da mesma forma. Uma classe implementada para controlar um robô deve ser modelada de forma que possibilite controlar tanto um objeto robô real com um virtual. Com isto tornasse possível experimentar geração de variados ambientes, combinando objetos reais e virtuais ou semente reais ou somente virtuais de forma síncrona e combinada para experimentação de equipamentos e treinamentos.

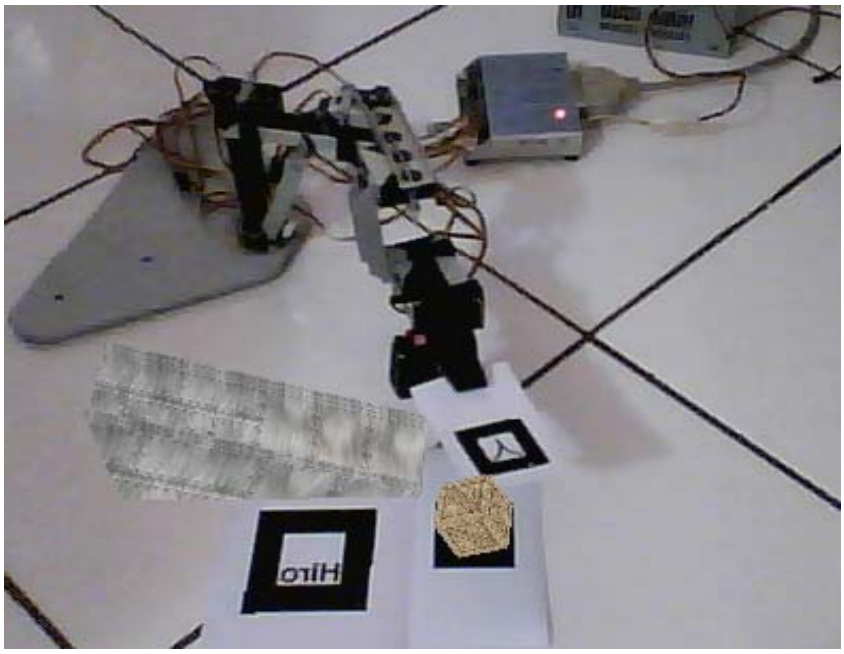


Figura 47 - Ambiente de Realidade Aumentada desenvolvido utilizando o metamodelo MDA proposto neste trabalho. [Próprio Autor]

5 CONCLUSÃO

Uma realidade muito presente nos projetos de *software*, é o fato que muitas vezes os modelos criados caem no desuso e acabam se tornando obsoletos, não são efetivamente implementados e nem atualizados quando ocorrem alterações no projeto, portanto os modelos não são refletidos no código e consequentemente não há reuso, ou seja, há retrabalho para o desenvolvimento de novos ambientes, os sistemas remendam códigos gerando sistemas confusos e de difícil manutenção.

Assim, com o emprego do metamodelo proposto dentro do paradigma de Desenvolvimento Orientado a Modelo – MDD, com o efetivo emprego através da aplicação da abordagem de Arquitetura Orientada a Modelo – MDA, possibilitou a criação de modelos e códigos correspondentes durante toda a vida do *software* e valorizou a utilização de modelos para desenvolvimento. A geração de código do comportamento dos objetos não foi realizada com transformação automática do modelo para código, mas nem por isso perde a correspondência entre modelos e código e vice-versa, pois a *interface* de comunicação é ditada pelos métodos das classes definidas no modelo PIM e PSM, ou seja, nenhum método poderá ser criado deliberadamente pelo programador, será necessário defini-los nos níveis superiores de PIM e PSM.

O metamodelo dentro do contexto de ambientes virtuais proporcionou distinguir o controle do ambiente realizado pelo modelo de RdP e da criação e composição dos objetos que participam do ambiente através de modelos descritos em UML. Assim, facilitou o reaproveitamento dos objetos para criação de outros ambientes sem novas modelagens, apenas redefinindo a dinâmica do ambiente, através da reutilização dos objetos contidos na Biblioteca de Classes, criando novos ambientes orientados pelo modelo de RdP, ou seja diminui a codificação manual, minimiza riscos de falha humana na programação, acelerando o desenvolvimento e colaborando para a manutenção, além de proporcionar a reutilização dos modelos existentes em novos projetos.

A proposta de utilizar uma abordagem de MDA da OMG traz a formalização conceitual através dos modelos que pode ser criado através de ferramentas que fornecem suporte para as transformações de PIM para PSM e de PSM para Código, agregado ao fato de adotar uma das linguagens de modelagem mais difundida na área de Engenharia de Software, a UML. Por outro, lado verificou-se que as ferramentas fornecem bom processo de

transformação automática na parte estrutural do *software*, mas carecem de transformações automáticas do modelo comportamental.

Por fim, neste trabalho foram desenvolvidos um metamodelo e um estudo de caso que demonstrou a aplicabilidade no desenvolvimento de ambientes virtuais minimizando a complexidade na construção de *software* geradores destes ambientes, com a aplicação de conceitos e técnicas de Desenvolvimento Orientado a Modelo, através da aplicação efetiva da abordagem MDA.

REFERÊNCIAS

[1] KATO, H. Documentation. *ARToolKit*. Disponivel em: <http://www.hitl.washington.edu/artoolkit/documentation/>. Acesso em: 21 abr. 2015.

[2] NYATLA. NyARToolkit project. *NyARToolkit*. Disponivel em: <http://nyatla.jp/nyartoolkit/wp/>. Acesso em: 21 abr. 2015.

[3] BRAGA, R. F. et al. Estudo comparativo de toolkits de Realidade Virtual e Aumentada visando aplicação educacional. *Workshop de Desafio da Computação Aplicada à Educação*. [S.l.]: [s.n.]. 2012. p. 138-147.

[4] BARROS, E. M. D. F. *VirTraM: Um Framework para o Desenvolvimento de Treinamentos Utilizando Realidade Virtual em Dispositivos Móveis*. Universidade Federal do Ceará. Fortaleza, p. 128. 2005.

[5] DIAS, L. et al. GIS2R — Augmented reality and 360° panoramas framework for geomarketing. *8th Iberian Conference on Information Systems and Technologies (CISTI)*, 19-22 June 2013. 1-5.

[6] ALCARAZO, S. B.; ROJO, J. C.; LAGUNA, S. Á. http://www.lookar.net/. *Look*, 2011. Disponivel em: <http://www.lookar.net/>. Acesso em: 14 Janeiro 2015.

[7] KLEPPE, A.; WARNER, J.; BAST, W. *MDA-Explained:* The Model Driven Architecture: Practice and Promise. ISBN 032119442X. ed. [S.l.]: Addison Wesley, 2003.

[8] SCHMIDT, D. C. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer Society Press*, Los Alamitos, CA, USA, v. 39, n. 2, p. 25-31, feb 2006. ISSN 0018-9162.

[9] VOELTER, M. A. G. I. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. *11th International Software Product Line Conference, SPLC 2007*. Kyoto: IEEE. 2007. p. 233-242.

[10] AMELLER, D. et al. Handling non-functional requirements in Model-Driven Development*:* An ongoing industrial survey. *23rd International Requirements Engineering Conference (RE)*. Ottawa, ON: IEEE. 2015. p. 208-213.

[11] AMELLER, D.; FRANCH, X.; CABOT, J. Dealing with Non-Functional Requirements

in Model-Driven Development. *18th IEEE International Requirements Engineering Conference (RE)*. Sydney, NSW: IEEE. 2010. p. 189-198.

[12] MURATA, T. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541-580, 1989.

[13] OLIVEIRA, C. D. *Associação de redes de Petri com objetos virtuais e reais para controle de ambientes virtuais imersivos e telepresença*. São Carlos: USP, 2008.

[14] VALE, L. D. N.; JULIA, S. *Especificação de Testes Funcionais usando Redes de Petri a Objetos para Softwares Orientados a Objetos*. Universidade Federal de Uberlândia - Faculdade de Computação - Programa de Pós-Graduação em Ciência da Computação. Uberlândia-MG, p. 138. 2009.

[15] LACHTERMACHER, L. et al. *Transformando o Diagrama de Atividade em uma Rede de Petri*. Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro. 2008. (ISSN 0103-9741).

[16] ANDRADE, V. C. *Transformação de modelos de Diagrama de Sequencia UML contemplando restrições de tempo e energia para Rede de Petri Temporal*. Curitiba - PR: Dissertação (Mestrado em Informática) Universidade Federal do Paraná, 2013.

[17] BACALÁ, S. J. *Arquitetura de software baseada numa abordagem UML/Redes de Petri com prevenção de bloqueio mortal em Sistemas de Tempo Real*. Universidade Federal de Uberlândia. Uberlândia-MG. 2003.

[18] BOUSETTA, B.; EL BEGGAR, O.; GADI, T. A Model Transformation Approach for Code Generation From State Machine Diagram. *IADIS International Journal on Computer Science and Information Systems*. [S.l.]: IADIS. 2014. p. 1-15.

[19] EL BEGGAR, O.; BOUSETTA, ; GADI, T. Automatic code generation by model transformation from sequence diagram of system's internal behavior. *International Journal of Computer and Information Technology*. Morocco: IJCIT. 2012. p. 129-146.

[20] LTD., S. S. P. The Ultimate Design & Build Platform. *SPARS Systems*, 2015. Disponivel em: <http://www.sparxsystems.com.au/>. Acesso em: 01 abr 2015.

[21] IBM. Rational Rose family. *IBM*, 2015. Disponivel em: <http://www-03.ibm.com/software/products/pt/ratirosefami>. Acesso em: 01 abr 2015.

[22] VARGAS, T. C. D. S. *Suporte à Edição de UML 2 Ambiente SEA*. Florianópolis: Universidade Federal de Santa Catarina, 2008.

[23] SAMPAIO, F. F. Modelagem Dinâmica Computacional e o Processo de Ensino-Aprendizagem - Algumas questões para reflexão. *Conferencia Internacional de Informatica e Educaçao do Chile - TISE'98*. [S.l.]: [s.n.]. 1998.

[24] MELLAR, H. E. A. *Learning with Artificial Worlds:* Computer Based Modelling. London: Falmer Press, 1994.

[25] SANTOS, A. D. C. K. D.; VARGAS, A. P.; MENDIZABAL, O. M. A. M. C. A. B. C. W. O ModelCiências: um portal para o projeto Modelagem Semiquantitativa e Quantitativa na Educação em Ciências. *Educar em Revista [online]*, Curitiba, p. 217-235, 2003. ISSN ISSN 0104-4060.

[26] SOUSA, G. C. M. *Desenvolvimento de Máquinas deExecução para Linguagens de Modelagem Específicas de Domínio:* Uma Estratégia Baseada em Engenharia Dirigida por Modelos. Goiânia: Universidade Federal de Goiás, 2012.

[27] OMG. *Unified Modeling Languagem Infrastructure Specification, v.2.3*. Object Management Group. [S.l.], p. 1. 2009.

[28] FOWLER, M. *UML Distilled:* A Brief Guide to the Standard Object Modeling Language. 3. ed. Boston: Addison Wesley, 2003.

[29] BARROS, J. P. M. P. R. Introdução à modelagem de sistemas utilizando redes de Petri. *Instituto Politécnico de Beja, ,* 2001.

[30] PETERSON, J. L. Petri Nets. *Computing Surveys*, v. 9, n. 3, set 1977.

[31] PALMA, J. G. *Metamodelo para a modelagem e simulação de sistemas a eventos discretos, baseado em Redes de Petri e Realidade Virtual:* uma aplicação em sistema de manufatura. São Carlos: Tese (Doutorado em Engenharia Mecânica) Escola de Engenharia de São Carlos - USP, 2001.

[32] KINDLER, E.; WEBER, M. The Petri Net Markup Language. *Lecture Notes in Computer Science*, p. 124-144, 2003.

[33] ISO-8879. *Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML)*. International Organization For Standarzation. [S.l.]. 1986.

[34] SOUZA, C. A. C. D. et al. Lingagem XML. *Revista de Informática Aplicada*, v. II, n. imes universidade, p. 86-93, jul/dez 2006. ISSN 02.

[35] NASCIMENTO, A. E. K. *Intercâmbio de Dados entre Aplicativos utilizando XML/XSLT*. Universidade Federal do Rio Grande do Sul. Porto Alegre, p. 81. 2001.

[36] VARA, J. M.; MARCOS, E. A framework for model-driven development of information systems: Technical decisions and lessons learned. *Journal of Systems and Software, Elsevier Inc.*, 85, n. 10, out. 2012. 2368–2384.

[37] FRANCE, R.; RUMPE, B. Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering (FOSE '07), Ieee*, 2, maio 2007. 37–54.

[38] LUCRÉDIO, D. *Uma Abordagem Orientada a Modelos para Reutilização de Software*. São Carlos-SP: Tese (Doutorado em Ciência da Computação e Matemática Computacional) Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, 2009. 19-232 p.

[39] TEPPOLA, S.; PARVIAINEN, P.; TAKALO, J. Challenges in Deployment of Model Driven Development. *2009 Fourth International Conference on Software Engineering Advances, IEEE*, set 2009. 15–20.

[40] STHAL, T.; VOLTER, M. *Model-Driven Software Development - Technology, Engineering, Management*. [S.l.]: Wiley, 2006.

[41] VOLTER, M.; BETTIN, J. Patterns for Model-Driven Software-Development, 2004.

[42] MELLOR, S. J. E. A. Model-Driven Development. *IEEE Software*, 2003.

[43] HAILPERN, B.; TARR, P. Model-driven development : The good , the bad , and the ugly, v. 45, n. 3, p. 451–461, 2006.

[44] SELIC, B. The pragmatics of model-driven development. *IEEE Software*, v. 20, n. 5, p. 19–25, set. 2003.

[45] AMEEDEEN, M. A.; BORDBAR, B.; ANANE, R. Model interoperability via Model Driven Development. *Journal of Computer and System Sciences*, v. 77, 2011.

[46] MODELWARE. *MDD Maturity Model*. [S.l.]. 2006.

[47] OMG, O. M. G. MDA - The Architecture Of Choice For A Changing World. *OMG*, 2015. Disponivel em: <http://www.omg.org/mda/>. Acesso em: 23 mar 2015.

[48] BORGES, H. P. et al. *Uma Arquitetura Baseada em Modelos*. Federal Institute of Education, Science and Technology of Maranhão. São Luís, p. 50.

[49] OMG, O. M. G. *MDA - Guide. rev. 2.0*. Object Management Group. [S.l.], p. 15. 2014. (OMG Document ormsc/2014-06-01).

[50] BROWN, A. An introduction to Model Driven Architecture Part I: MDA and today's

systems. *IBM-Developer Works*, 17 feb. 2004. 1-16.

[51] BARBOSA, P. E. E. S. *MDA-Veritas: Uma Arquitetura MDA Estendida para Transformações de Sistemas Concorrentes Preservadoras de Semântica*. Universidade Federal de Campina Grande. Campina Grande, p. 214. 2011.

[52] OMG, O. M. G. *MDA - Guide Version 1.0.1*. Object Management Group. [S.l.], p. 62. 2003. (omg/2003-06-01).

[53] CALIARI, G. L. P. *Transformações e mapeamentos da MDA e sua implementação em três ferramentas*. São Paulo: Dissertação(Mestrado em Sistemas Digitais)Escola Politécnica da Universidade de São Paulo, 2007. 137 p.

[54] SINGH, Y.; SOOD, M. The Impact of the Computational Independent Model for Enterprise Information System Development. *International Journal of Computer Applications*, v. II, n. 8, p. 21-26, dez. 2010. ISSN 0975 – 8887.

[55] LTD, S. S. P. The Ultimate Design & Build Platform. *SPARS Systems*, 2015. Disponivel em: <http://www.sparxsystems.com.au/>. Acesso em: 01 abr 2015.

[56] CHANGE VISION, I. Astah. *Astah*, 2015. Disponivel em: <astah.net>. Acesso em: 01 jul 2015.

[57] NETTO, A. V. et al. Realidade Virtual e suas aplicações na área de Manufatura, Treinamento, Simulação e Desenvolviemnto de Produto. *G&P - Gestão e Produção*, São Carlos, v. 5, n. 2, p. 104-116, Ago. 1998.

[58] AUKSTAKALNIS, S.; BLATNER, D. *Silicon Mirage:* The art and science of virtual reality. Berkeley: Peachpit Press, 1992.

[59] KIRNER, C.; PINHO, M. S. Introdução à Realidade Virtual. *Livro do Mini-curso - In: Workshop de Realidade Virtual - São Carlos*, p. 1-40, 1997.

[60] SENA, D. C. D. A. J. P. D. C. Uso de realidade virtual como visualizador e avaliador do uso de simulação de evento discreto de sistemas de manufatura. *VI Simpósio de Engenharia de Produção da Região Nordeste*, Campina Grande, jun 2011. 1-9.

[61] TORI, R.; KIRNER, C.; SISCOUTO, R. *Fundamentos e Tecnologia de Raealidade Virutal e Aumentada*. Belém: SBC, 2006. 395 p.

[62] CRUZ-NEIRA, C. A. A. The CAVE: Audio visual experience automatic virtual environment. *Communications of the ACM*, 35, n. 6, 1992. 64-72.

[63] KIRNER, C. *Mãos colaborativas em ambientes de realidade misturada*. Piracicaba:

UNIMEP, 2004.

[64] ROBERTSON, G. G.; CARD, S. K.; MACKINLAY, J. D. Three views of virtual reality: Nonimmersive virtual reality. *Computer(IEEE)*, 26, n. 2, fev 1993. 81-83.

[65] BOTEGA, L.; CRUVINEL, C. P. E. Realidade Virtual: Histórico, Conceitos e Dispositivos. In: COSTA, R. M. E. M. C.; RIBEIRO, M. W. S. *Pré-simpósio, XI Symposium on Virtual - Aplicações de Realidade Virtual e Aumentada* -. Porto Alegre: SBC, 2009. p. 8-30.

[66] EUROPA, @. Een kijkje in de 3D CAVE studio. *@Ford Europa*, p. 18, jun/ago 2013.

[67] RODELLO, I. A. et al. Realidade Virtual e Aumentada Aplicada na Área de Negócios: casos na área de Marketing e de Projeto e Desenvolvimento de Produtos. *Tendências e Técnicas em Realidade Virtual e Aumentada*, v. 3, p. 43-59, mai 2013.

[68] TORI, R.; KIRNER, C. Fundamentos de Realidade Virtual. In: KIRNER, C.; TORI, R. E. S. R. *Fundamentos e Tecnologia de Realidade Virtual e Aumentada - VIII Symposium on Vitrual Reality*. Belém-PA: Editora SBC - Sociedade Brasileira de Computação, 2006. Cap. 1, p. 2-21.

[69] AZUMA, R. A Survey of Augmented Reality. *In Presence: Teleoperators and Virtual Environments* , v. 6, n. 4, p. 355-385, Aug 1997.

[70] INSLEY, S. Obstacles to General Purpose Augmented Reality. *ECE 399H, Information Security & Cryptography*, dez 2003.

[71] MILGRAM, P. E. A. Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum. *Telemanipulator and Telepresence Technologies, SPIE,* v. 2351, p. 282-292, 1994.

[72] BIANCHINI, C. D. P.; SILVA, L. Sistemas de Realidade Aumentada Móvel Suportados por Computação em Nuvem. *Tendências e Técnicas em Realidade Virtual e Aumentada*, v. 4, p. 9-32, mai 2014.

[73] KIRNER, C. et al. *Livro de Reaidade Aumentada para Crianças Portadora de Necessidades Especiais (LIRA-ESPEC)*. [S.l.]: 5ª Mostra Acadêmica UNIMEP, 2007.

[74] ROSSI, R. Largest augmented reality mark. *Officially Amazing Guinness World Records*, 2010. Disponivel em: <http://www.guinnessworldrecords.com/world-records/largest-augmented-reality-mark>. Acesso em: 29 mar. 2015.

[75] PEDERSEN, I.; TRUEMAN, D. “Sergey Brin is Batman”*:* Google’s Project Glass & the

instigation of computer adoption in popular culture. *CHI'13*. Paris, France: [s.n.]. 2013.

[76] SILVA, S. C. D. et al. Um Estudo Sobre Transformacoes Modelos-Modelos Baseado na Arquitetura Dirigida Por Modelos. *Anais da Semana de Informática CESIT/UEA*, Manaus-AM, v. 1, n. 1, p. 10, 2013. ISSN 2319-0418.

[77] KULESZA, R. *Uma Abordagem de Desenvolvimento Orientado a Modelos para a Integração entre Projetos de Mídia e Software no Domínio de Aplicações de TV Digital*. Recife: Tese (Doutorado em Ciência da Computação) Universidade Federal de Pernambuco, 2013.

[78] KIRNER, C. Funcionamento e Utilização do Sistema de Autoria Colaborativa com Realidade Aumentada - SACRA. *CKirnes*, 2011. Disponivel em: <http://www.ckirner.com/sacra/>. Acesso em: 01 maio 2015.

[79] TANENBAUM, A. S. *Sistemas Operacionais Modernos*. [S.l.]: Prentice Hall (Pearson), 2007.

TRABALHOS PUBLICADOS PELO AUTOR

1. Márcio de Abreu Moreira, Jandira Guenka Palma, Jéssica Tomaz da Silva, Vanessa Matias Leite, Wilson Hissamu Shirado, **Implementation of MoProSoft in a small company with MPS.BR-F**, 8th IADIS International Conference on Information Systems (IS 2015), Madeira-Portugal, março/2015, (Qualis B4)
2. Márcio de Abreu Moreira, Jandira Guenka Palma, Wilson Hissamu Shirado, Cristina Yassue Morimoto, **Modelagem de Negócio como apoio ao Desenvolvimento Ágil de Software**, 12th CONTECSI USP Congresso Internacional de Gestão da Tecnologia e Sistemas de Informação, São Paulo, May 20 to 22/2015, (Qualis B4)
3. Márcio de Abreu Moreira, Jandira Guenka Palma, Sylvio Barbon Júnior, Wilson Hissamu Shirado, **Estudo Comparativo entre Algoritmos das Transformadas Discretas de Fourier e Wavelet**, Revista Brasileira de Computação Aplicada (RBCA - ISSN 2176-6649), Outubro/2015, volume 7, número 3 (Qualis B4).
4. Wilson Hissamu Shirado, Jandira Guenka Palma, Márcio de Abreu Moreira, **Automação de testes funcionais em sistema embarcados via Processamento Digital de Imagens**, Revista Engenharia de Software Magazine (ISSN- 1983-1277), Edição 82, Janeiro/2016 (Qualis-C).
5. Wilson Hissamu Shirado, Jandira Guenka Palma, Márcio de Abreu Moreira, Tânia Eiko Eishima, **Model Driven Architecture: Entendendo e aplicando os Conceitos. Engenharia de Software Magazine**, Revista Engenharia de Software Magazine (ISSN-1983-1277), Aceito para publicação em 2015 (Qualis-C).