Práticas e Experiências no Ensino de Engenharia Dirigida por Modelos

Paulo Henrique M. Maia[1], Fabiano Gadelha[1], Marcos Borges[1], Laryssa Lima Muniz[1], Amanda Souza da Silva[1], Janaide N. de S. Ximenes[1]

[1]Mestrado Acadêmico em Ciências da Computação – Universidade Estadual do Ceará (UECE)
Av. Dr. Silas Munguba, 1700 – Caixa Postal 60.714.903 – Fortaleza – CE – Brasil

pauloh.maia@uece.br

{mandinhaestatistica,fabianojgf,nogueirajanaide}@gmail.com

{laryssa.muniz,marcos.borges}@aluno.uece.br

Abstract. *Model-driven Engineering (MDE) is a software development approach that aims at generating source code from model specification and transformation. Although it has succeed in industry, we believe that MDE has not become yet a de facto standard for implementing systems due to, among other factors, the way it has been taught in university. This paper reports an experience on teaching an MDE course in an Academic Master's Program in Computer Science at State University of Ceará, describing the adopted methodology, the faced difficulties and proposed solutions, in addition to presenting the results of an evaluation of the course carried out with egressed students.*

Resumo. *A Engenharia Dirigida por Modelos (Model-driven Engineering - MDE) é uma abordagem de desenvolvimento de software que visa produzir código a partir da especificação e transformação de modelos. Apesar de sua aplicação bem sucedida na indústria, acreditamos que a MDE ainda não se tornou um padrão de fato na construção de sistemas devido, dentre outros fatores, à forma que ela tem sido ensinada nas universidades. Este artigo relata a experiência no ensino de uma disciplina de MDE no Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará, descrevendo a metodologia adotada, dificuldades enfrentadas e soluções encontradas, além de apresentar os resultados de uma avaliação da disciplina realizada com os alunos egressos.*

1. Introdução

Nas últimas décadas, várias linguagens de programação, processos, técnicas, *frameworks* e ferramentas têm sido propostos para apoiar a concepção e implementação de sistemas de informação, atividades consideradas caras e complexas [Neto and de Oliveira 2013]. Elas visam, dentre outros fatores, facilitar a comunicação entre as partes interessadas e agilizar o desenvolvimento e testes desses sistemas, proporcionando, assim, a entrega de produtos que atendam às exigências das empresas [Booch 2004, Krogstie 2012, Whittle et al. 2013, Whittle et al. 2015]. Entretanto, essas soluções nem sempre conseguem atacar a crescente complexidade de tais sistemas devido ao contínuo e rápido surgimento de novas tecnologias, o que acarreta uma necessidade de interoperabilidade e

MAIA, P. H. M.; GADELHA, F.; BORGES, M.; MUNIZ, L. L.; SILVA, A. S.; XIMENES, J. N. S.;
Práticas e Experiências no Ensino de Engenharia Dirigida por Modelos
iSys | Revista Brasileira de Sistemas de Informação, Rio de Janeiro, vol. 9, No. 2, pp. 106-135, 2016

evolução de sistemas existentes para incorporá-las. Uma alternativa para o tratamento desse problema é abstrair os sistemas por meio de modelos.

Modelos são artefatos utilizados na Engenharia de Software para capturar e representar o conhecimento adquirido durante o processo de desenvolvimento de software. Eles auxiliam na comunicação com os clientes e usuários, apoiando a concepção do software [Brambilla et al. 2012]. Além disso, modelos também ajudam analistas a entender problemas complexos e projetar suas soluções potenciais por meio de abstração. Portanto, sistemas de software podem se beneficiar muito do uso de modelos e técnicas de modelagem [Selic 2003].

Em virtude das vantagens obtidas com a utilização de modelos, uma nova tendência emergiu considerando modelos não apenas como artefatos de documentação, mas como elementos centrais no processo de Engenharia de Software. Com o uso de técnicas complexas, tais como metamodelagem e transformações de modelos, é possível obter geradores de código-fonte ou interpretadores de modelos, tornando possível a criação ou execução automática do software com base nesses modelos.

A Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE) é uma abordagem que conduz o processo de desenvolvimento de software por meio da construção e transformações de modelos. A MDE fornece meios para adaptar novas tecnologias com sistemas legados e permite a integração entre diferentes tecnologias, propiciando uma maior agilidade no desenvolvimento de sistemas de informação modernos [Schmidt 2006].

De acordo com Arisholm *et al.* (2006), a MDE permite os engenheiros de software trabalharem em um nível mais alto de abstração nos estágios de desenvolvimento de software, trazendo, consequentemente, benefícios como portabilidade, produtividade e manutenibilidade. Selic (2003) e France e Rumpe (2007) afirmam que a MDE pode fornecer técnicas e ferramentas para amenizar a dificuldade de projetar sistemas complexos.

A abordagem MDE possui três elementos principais: (i) o *modelo* é uma abstração de um sistema frequentemente utilizado para substituir o sistema que se deseja construir/desenvolver/projetar [Ludewig 2003], ou seja, é uma representação simplificada da realidade criada para um determinado propósito. Um modelo conceitual do sistema de informação é projetado usando um metamodelo orientado a objetos [da Silva 2010]; (ii) o *metamodelo* auxilia na simplificação do manuseio, análise e reflexão sobre um determinado assunto de interesse, facilitando a compreensão da complexidade de um sistema. Pode ser conceituado como uma abstração de algo com o propósito de entendimento antes de construí-lo [Rumbaugh et al. 1991]. Um metamodelo, que também é um modelo, formaliza os conceitos fornecidos pela linguagem de modelagem que é definida pela sua sintaxe concreta e sintaxe abstrata; por fim, (iii) as *transformações de modelos*, segundo Miller and Mukerji (2003), são o processo de converter um modelo em outro modelo do mesmo sistema, tendo como um dos seus principais objetivos a redução do esforço e de erros ao automatizar a construção e modificação de modelos. Tais transformações podem ser classificadas como Modelo para Modelo (*Model-To-Model* - M2M) ou Modelo para Texto (*Model-To-Text* - M2T) [Brambilla et al. 2012].

Apesar da MDE ser uma abordagem bem estabelecida e aplicada com sucesso em vários setores industriais, como automotivo, aeroespacial, telecomunicações e sistemas

de informação [Hutchinson et al. 2011][Whittle et al. 2014], ela ainda não se tornou um padrão de fato na implementação de sistemas [Mussbacher et al. 2014]. Whittle *et al.* (2014) mencionam que uma possível explicação está em como a MDE tem sido ensinada nas universidades. Eles apontam que os conceitos de MDE são geralmente ensinados de forma fragmentada, em diferentes disciplinas e em momentos distintos do curso, o que faz com que muitos alunos (e futuros desenvolvedores) tenham dificuldades em lidar com abstração, modelagem e geração de código de uma maneira geral. Esse pensamento é corroborado por Cowling (2003), que afirma que os conceitos de modelagem e estrutura de software não recebem atenção suficiente na grade curricular de um curso de Engenharia de Software. Por fim, de acordo com Clarke *et al.* (2009), o uso da MDE motiva estudantes a pensar e criar modelos corretos e completos em fase de projeto, uma vez que vários deles irão ser usados para gerar a implementação resultante.

Existem diversos relatos sobre o ensino da MDE em diferentes universidades, seja em cursos de graduação ou pós-graduação [Blay-Fornarino 2008, Clarke et al. 2009, Burden et al. 2012, Batory et al. 2013, Schmidt et al. 2014, Poruban et al. 2014]. Contudo, não foi encontrado nenhum artigo que discuta a experiência e mostre as práticas realizadas em uma disciplina de MDE no Brasil. Particularmente, há poucas disciplinas de Engenharia de Dirigida por Modelos (com suas variantes nomenclaturas) nas universidades brasileiras. Em uma busca na Plataforma Sucupira[1], apenas dois cursos de pós-graduação em Computação tinham uma disciplina da MDE em sua grade curricular (a pesquisa em cursos de graduação não foi realizada devido à imensa quantidade de cursos), o que colabora para a falta de conhecimento por parte dos profissionais de software sobre esse assunto e a pouca adesão dessa prática nas empresas brasileiras.

Particularmente para a área de Sistemas de Informação, o ensino de MDE pode ser um fator diferencial para a formação do aluno, uma vez que o faz refletir sobre melhores práticas para a modelagem de sistemas, conhecer ferramentas e modelos que podem ser usados para diferentes domínios, e potencializa sua capacidade técnica para o desenvolvimento de soluções independentes de plataforma e linguagem de programação. Além disso, do ponto de vista acadêmico, o ensino da Engenharia Dirigida por Modelos oportuniza o aluno a explorar melhor o tema em um curso de pós-graduação, gerando pesquisas que podem resultar em produtos úteis para o mercado, favorecendo assim uma maior integração universidade-empresa.

O objetivo deste artigo é apresentar um relato da experiência do ensino da Engenharia Dirigida por Modelos em uma disciplina homônima do Mestrado Acadêmico em Ciência da Computação (MACC) da Universidade Estadual do Ceará (UECE). Para tanto, a metodologia adotada é descrita, os principais conceitos e ferramentas de MDE utilizados na disciplina são detalhados, e uma avaliação da mesma feita pelos alunos que a cursaram é apresentada. Também são discutidas neste artigo as principais dificuldades e soluções adotadas durante o ensino dessa disciplina.

Este relato é importante pois disponibiliza uma experiência de sucesso no ensino de MDE que, apesar de ter sido realizado no contexto de um curso de pós-graduação em Ciência da Computação, pode ser adaptado ou replicado para outros cursos de pós-graduação e graduação em áreas correlatas, como Sistemas de Informação. Adicional-

[1] https://sucupira.capes.gov.br/sucupira/

mente, este trabalho apresenta detalhes técnicos sobre as etapas de um processo MDE, o que é útil não só para professores, mas também para alunos que buscam material mais especializado sobre o tema, o qual não é fácil de ser encontrado na literatura.

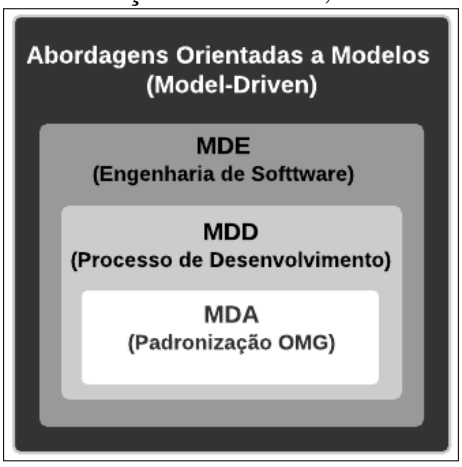
O restante deste artigo está organizado nas seguintes seções. Seção 2 descreve os principais conceitos da MDE, as etapas e elementos de um processo MDE. Seção 3 apresenta a disciplina de Engenharia Dirigida por Modelos do MACC e sua metodologia, além de descrever um exemplo da aplicação do processo MDE abordado na disciplina, enfatizando a modelagem, transformação de modelos e geração de código. A Seção 4 apresenta a avaliação da disciplina. A Seção 5 mostra as dificuldades encontradas e soluções adotadas durante a disciplina, enquanto a Seção 6 discute os principais trabalhos relacionados. Finalmente, a Seção 7 apresenta as conclusões e identifica os possíveis trabalhos futuros.

2. Engenharia Dirigida por Modelos

A Engenharia Dirigida por Modelos é uma abordagem na qual modelos estão no centro de desenvolvimento de artefatos de software [Schmidt 2006, France and Rumpe 2007]. Segundo Hofstader (2006), a MDE parte do princípio de modelar um sistema abstrato, no qual existe maior compreensão do que o software faz ou deve fazer e, a partir disso, desenvolver o sistema sem a necessidade de codificá-lo, ou seja, desenvolvendo-o em alto nível.

O principal objetivo da MDE é aumentar o nível de abstração utilizando modelos, ao invés de interagir manualmente com todo o código-fonte, ficando o desenvolvedor protegido dessas complexidades nas diversas plataformas de programação. A definição de MDE expande o conceito de Desenvolvimento Orientado por Modelos (*Model-Driven Development* - MDD), que engloba a definição da Arquitetura Orientada por Modelos (*Model-Driven Architecture* - MDA [OMG 2016]), conforme mostra a Figura 1.

Figura 1. Relação entre MDE, MDD e MDA

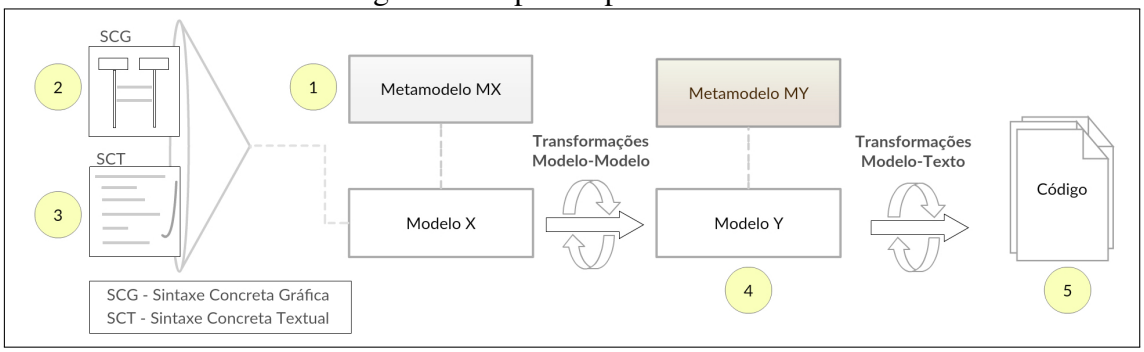


Um processo de MDE pode ser visto na Figura 2 e contempla, geralmente, cinco etapas: (1) metamodelagem, (2) geração da sintaxe concreta gráfica, (3) geração da sintaxe concreta textual, (4) transformação entre modelos (M2M) e, por fim, (5) a geração de código (M2T) [Brambilla et al. 2012]. Vale ressaltar que não necessariamente todas as

fases devem ocorrer. Por exemplo, a descrição da sintaxe gráfica nem sempre é realizada. Contudo, no contexto deste trabalho, foi escolhido abordar um processo mais completo para que os alunos possam exercitar todas as possíveis etapas.

Cada uma dessas etapas é descrita em mais detalhes a seguir.

Figura 2. Etapas do processo MDE



2.1. Metamodelagem

O metamodelo consiste em um modelo que define a linguagem de modelagem utilizada por outro modelo. De acordo com Bézivin (2005), os metamodelos descrevem os diversos tipos de elementos contidos no modelo e a forma como eles são relacionados e restringidos.

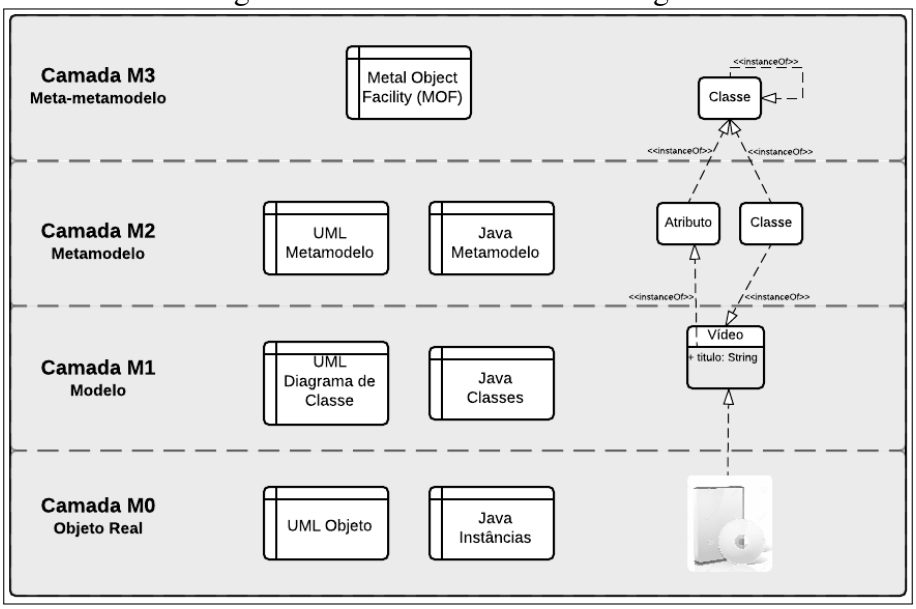
A metamodelagem abrange quatro instâncias, conforme ilustrado pela Figura 3: objeto real (Camada M0), modelo (camada M1), metamodelo (camada M2) e meta-metamodelo (camada M3). De acordo com a Figura 3, fica evidenciado que os modelos representam objetos do mundo real, enquanto um metamodelo pode ser considerado uma maior abstração, enfatizando as propriedades do modelo. Segundo Brambilla *et al.* (2012), em um sentido prático, metamodelos constituem basicamente a definição de uma linguagem de modelagem que proporciona uma maneira de descrever toda a classe de modelos que podem ser representados por essa linguagem. Os meta-metamodelos podem ser considerados como modelos que descrevem metamodelos.

Em qualquer nível no qual a prática da metamodelagem é considerada, pode-se afirmar que um modelo está em conformidade com seu metamodelo da mesma forma que um programa de computador está em conformidade com a gramática da linguagem de programação em que o mesmo está escrito [Brambilla et al. 2012]. Assim, na Figura 2, o modelo X está em conformidade com o metamodelo MX, por exemplo.

2.2. Modelos

Um modelo é uma abstração de um sistema frequentemente utilizado para substituir o sistema real e representa uma visão parcial e simplificada do sistema [Ludewig 2003]. Portanto, a criação de vários modelos é normalmente necessária para melhor representar e compreender o sistema que se deseja construir. Neste trabalho serão utilizados dois modelos para ilustrar o processo de MDE: Diagrama de Sequência (DS) da UML (*Unified Modeling Language*) e *Labelled Transition Systems* (LTS). A escolha de ambos os modelos justifica-se pelo fato de ser comum encontrar trabalhos

Figura 3. Conceito de Metamodelagem



na literatura que usam a transformação de um DS para LTS com diferentes finalidades [Cartaxo et al. 2007, Ziadi et al. 2011], como por exemplo testes baseados em modelos.

Os DSs descrevem uma sequência de ações que ocorrem em um sistema, exibindo o comportamento dinâmico do mesmo. Têm como objetivo estabelecer a interação entre objetos por meio da troca de mensagens entre eles para a realização de determinada tarefa ou funcionalidade especificada em um caso de uso. A Figura 4 apresenta um diagrama de sequência simples contendo troca de mensagens entre um ator e dois objetos do sistema. Vale notar que a mensagem MensagemSincrona3 está dentro de um container do tipo *loop*, que indica que aquela mensagem se repetirá enquanto uma determinada condição for satisfeita.

O LTS [Keller 1976] é um modelo formal que provê uma descrição integral do conjunto de todos os possíveis comportamentos do sistema. É constituído por um conjunto de estados (sendo um deles o estado inicial) e transições entre esses estados rotuladas por ações, como mostra a Figura 5.

Formalmente, um LTS é uma tupla *(S, A, T,* q_0*)*, na qual:

- *S* é um conjunto finito não vazio de estados;
- *A* é um conjunto finito não vazio de rótulos;
- *T* define o conjunto de transições rotuladas entre os estados;
- q_0 representa o estado inicial;

2.3. Sintaxe Concreta

Segundo Fondement (2007), a sintaxe concreta pode ser compreendida como uma linguagem superficial que funciona como uma interface que faz a ponte entre as instâncias dos conceitos da linguagem e permite que o ser humano possa produzir ou ler instâncias de acordo com a estrutura definida pelo modelo. Sintaxe concreta pode ser de natureza gráfica ou textual, mas muitas vezes tem-se uma mistura de ambos [Brambilla et al. 2012].

Figura 4. Exemplo de Diagrama de Sequência

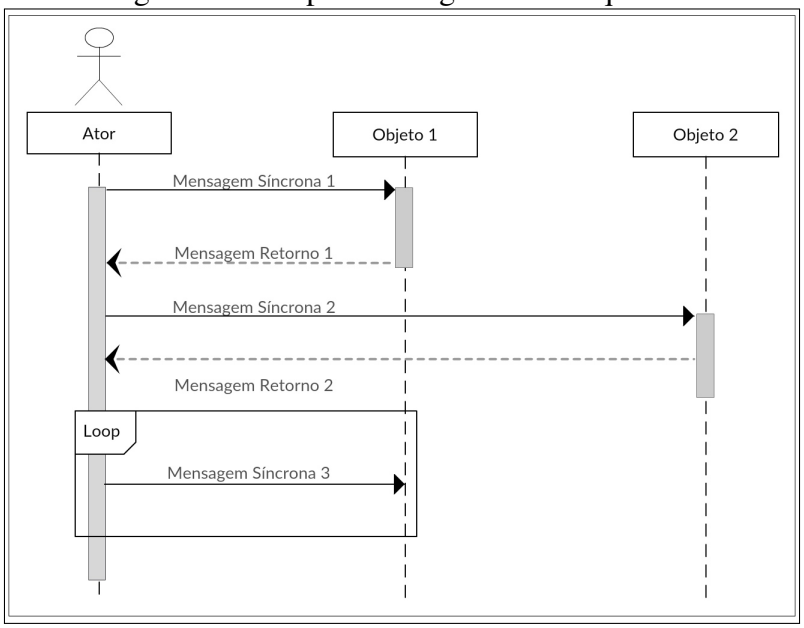
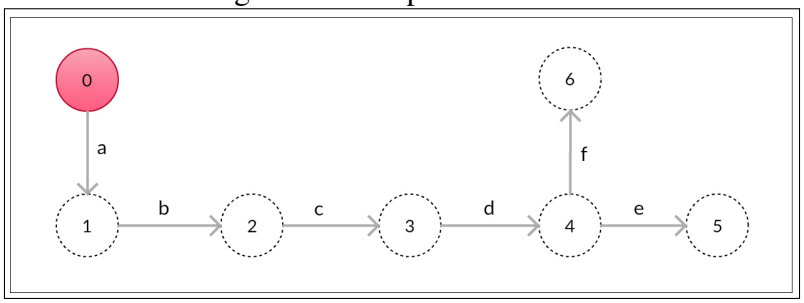


Figura 5. Exemplo de um LTS



Na definição da sintaxe concreta gráfica, cada elemento do metamodelo deve possuir uma representação visual, na qual define-se um ícone, imagem ou figura geométrica e indica-se propriedades a serem exibidas na interface. Essas ferramentas fazem uma distinção clara entre entidades e relacionamentos, tornando assim possível desenhar apenas um tipo do gráfico. Esse tipo de linguagem é chamada linguagem baseada em conexões.

A sintaxe concreta textual pode ser genérica ou específica [Brambilla et al. 2012]. No primeiro caso, ela pode ser aplicada a todo tipo de modelo, como por exemplo, um formato textual para especificar diagramas de objetos. Apesar de abrangente, traz como desvantagem o fato de não poder ser ajustada para lidar com as especificidades de alguns domínios. Já no segundo caso, a sintaxe é expressa em forma de regras, estando em conformidade com uma gramática do tipo *Extended Backus-Naur-Form* (EBNF), que pode ser processada por compiladores para geração de processadores de texto para a linguagem criada. Esse tipo de sintaxe é apropriado para uma linguagem específica de domínio (*Domain Specific Language* - DSL).

Uma DSL é uma linguagem que fornece, por intermédio de notações e de abstrações apropriadas, um poder de expressividade focado em, e geralmente restrito a, um determinado domínio de problema [Van Deursen and Klint 2002]. Para criar uma DSL é

necessário especificar um metamodelo da linguagem que represente de modo rigoroso a sintaxe da própria linguagem. Uma DSL contém sintaxe e semântica no mesmo nível de abstração que o domínio do problema oferece.

2.4. Transformações

A transformação de modelos é a geração automática de um modelo destino a partir de um modelo origem utilizando um conjunto de regras [Kleppe et al. 2003]. As transformações de modelos utilizadas neste trabalho são: Modelo para Modelo (M2M) e Modelo para Texto (M2T).

M2M é o método que realiza a transformação de um modelo de entrada em um modelo de saída, que podem ou não ser instâncias do mesmo metamodelo. O procedimento para transformação dos modelos é composto por 4 etapas: (i) criação dos metamodelos de entrada e de saída; (ii) definição das regras de transformação; (iii) criação da instância do modelo de entrada; e (iv) geração do modelo de saída [Brambilla et al. 2012].

M2T é o método de transformação de modelos no qual um código-fonte é gerado a partir de um modelo, também conhecido como Modelo para Código (*Model-To-Code* - M2C). Recebe um modelo de entrada e realiza a geração dos arquivos de código-fonte para uma determinada linguagem de programação.

3. A Disciplina e sua Metodologia

A disciplina de Engenharia Dirigida por Modelos tem sido lecionada no MACC há dois anos, uma vez por ano. A primeira turma teve 13 alunos, enquanto a segunda teve seis. A disciplina possui quatro créditos (68 horas/aula no total) e sua ementa pode ser obtida no site do curso[2]. O livro-texto trabalhado na disciplina é o de Brambilla *et al.* (2012).

Segundo Schmidt *et al.* (2014), há três possíveis abordagens para ensinar MDE: (i) *puramente teórica*, na qual apenas conceitos de MDE são explicados. Esse caso geralmente acontece quando MDE é parte de uma disciplina maior (Engenharia de Software, por exemplo), e não há tempo suficiente para entrar em detalhes; (ii) *apoiada por ferramenta*, na qual uma ferramenta, geralmente o Eclipse Modeling Framework (EMF), é utilizada para modelar o sistema e gerar classes Java a partir dos modelos; (iii) *prática*, na qual os principais conceitos são estudados e aplicados, como por exemplo, na criação de um gerador de código a partir de ferramentas apropriadas.

De acordo com essa denominação, a disciplina de MDE discutida no presente trabalho encaixa-se na abordagem prática, uma vez que a metodologia da disciplina a divide em duas partes: a primeira, mais introdutória e teórica, visa explicar os principais conceitos (conforme mostrado na seção 2) e identificar onde e como a MDE tem sido aplicada na indústria e na academia por meio da leitura e discussão de artigos. Na segunda, os alunos praticam cada uma das etapas de um processo de MDE utilizando diferentes ferramentas e linguagens. A nota da disciplina é uma média ponderada das notas das atividades realizadas nessas duas etapas.

A seguir essas duas etapas que compõem a disciplina são melhor detalhadas.

[2] http://www.uece.br/macc/index.php/disciplinas

3.1. Etapa teórica

Nesta etapa, a disciplina tem seu foco voltado para aspectos introdutórios e de fundamentação relativos à MDE e seu uso. Seguindo os capítulos do livro-texto da disciplina, são abordados assuntos como conceitos e princípios de MDE, casos de uso e ferramentas de MDE, Arquitetura Dirigida por Modelos, MDE ágil e sua adoção na indústria, e processo de MDE.

Ao término da exposição de cada tópico, são indicados pelo professor artigos relacionados ao assunto abordado, a partir dos quais os alunos deverão realizar a leitura e, posteriormente, uma discussão em sala de aula. Além disso, para cada tópico há um artigo para o qual deve-se desenvolver uma resenha crítica, que é avaliada quanto à estrutura do texto, utilização correta da linguagem e domínio sobre o assunto exposto. Por fim, os alunos devem escolher um artigo sobre um dos tópicos explorados e realizar uma apresentação do mesmo para a turma.

A nota dessa etapa é formada pela soma das notas das resenhas e da apresentação e tem peso 1 na composição da nota final.

3.2. Etapa prática

A disciplina, nessa etapa, passa a ter um foco mais prático, na qual os alunos irão aplicar os conhecimentos adquiridos na etapa teórica no desenvolvimento de um projeto, indicado pelo professor, utilizando MDE. O projeto, que é o mesmo para todos os alunos, tem como objetivo praticar as atividades envolvidas na abordagem MDE e consiste em cinco partes: metamodelagem, construção da sintaxe concreta gráfica e textual, transformação entre modelos e geração de código, conforme apresentado na Seção 2.

Cada uma das cinco partes do projeto é entregue em forma de implementação em uma ferramenta adequada para aquela atividade e possui um *deadline* específico para entrega. Ao final, o projeto desenvolvido é avaliado pelo professor quanto ao funcionamento e corretude, além de ser usado para avaliar o aluno quanto ao seu domínio sobre o conhecimento técnico utilizado na concepção do mesmo.

A nota dessa etapa é formada pela soma das notas das implementações e tem peso 2 na composição da nota final.

3.2.1. Exemplo Prático

Esta subseção detalha um exemplo da aplicação dos conhecimentos e atividades que compõem o processo de MDE, ilustrado na Figura 2, ensinado na disciplina. O exemplo mostrado a seguir trata do projeto construído por alunos que cursaram a disciplina em sua mais recente edição.

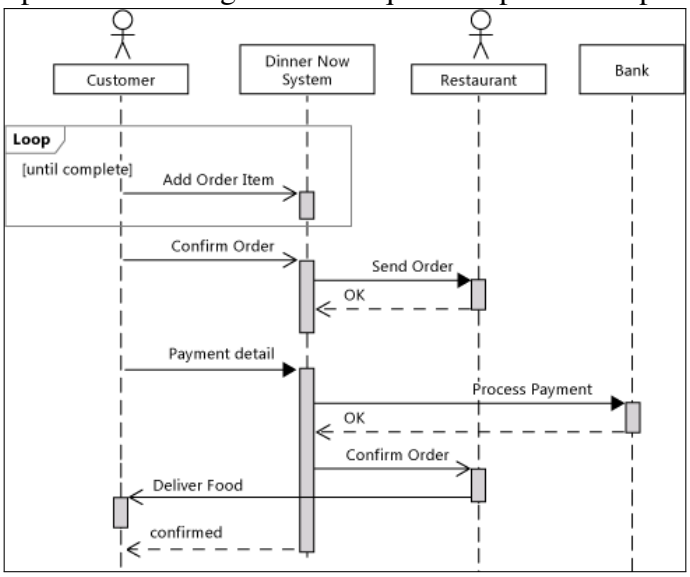
O projeto consistiu em modelar uma versão simplificada do Diagrama de Sequência da UML contendo apenas objetos, atores, linhas de vida, mensagens e *loop*. A escolha desse modelo se deu pelo fato do DS ser bastante conhecido e utilizado para modelagem de comportamento de software, visando assim facilitar a aplicação das etapas do processo. Para tanto, na etapa de metamodelagem é construído um metamodelo que deve ser uma representação fiel da estrutura de uma instância do DS, enquanto nas etapas de constru-

ção da sintaxe concreta gráfica e textual são construídas representações com elementos gráficos e textuais para o metamodelo concebido na etapa anterior.

Esse modelo do DS é então submetido às etapas de transformação. Para a transformação entre modelos (M2M), foi adotado como modelo destino o *Labelled Transition System*, de forma que seja construído um transformador para realizar a conversão de uma instância do DS em uma instância do LTS. Conforme mencionado na seção 2.2, a escolha desses modelos se deu pela utilização de ambos conjuntamente, como pode ser encontrado em artigos na literatura [Cartaxo et al. 2007, Ziadi et al. 2011]. Já na etapa de transformação de modelo em texto (M2T), também conhecida como geração de código, o DS é transformado em um conjunto de classes Java, procedimento comumente encontrado em ferramentas UML.

A Figura 6 ilustra o exemplo do DS que descreve uma funcionalidade de realização e entrega de pedidos de um restaurante, o qual será utilizado no decorrer desta seção. O DS contém: quatro entidades, sendo os dois atores, **Customer** e **Restaurant**, e dois objetos, **Dinner Now System** e **Bank**, tendo cada uma dessas entidades uma linha de vida associada; um conjunto de mensagens de requisição e resposta trocadas entre as entidades; além de uma estrutura de *loop* referenciando uma única mensagem.

Figura 6. Exemplo base do Diagrama de Sequência aplicado ao processo de MDE



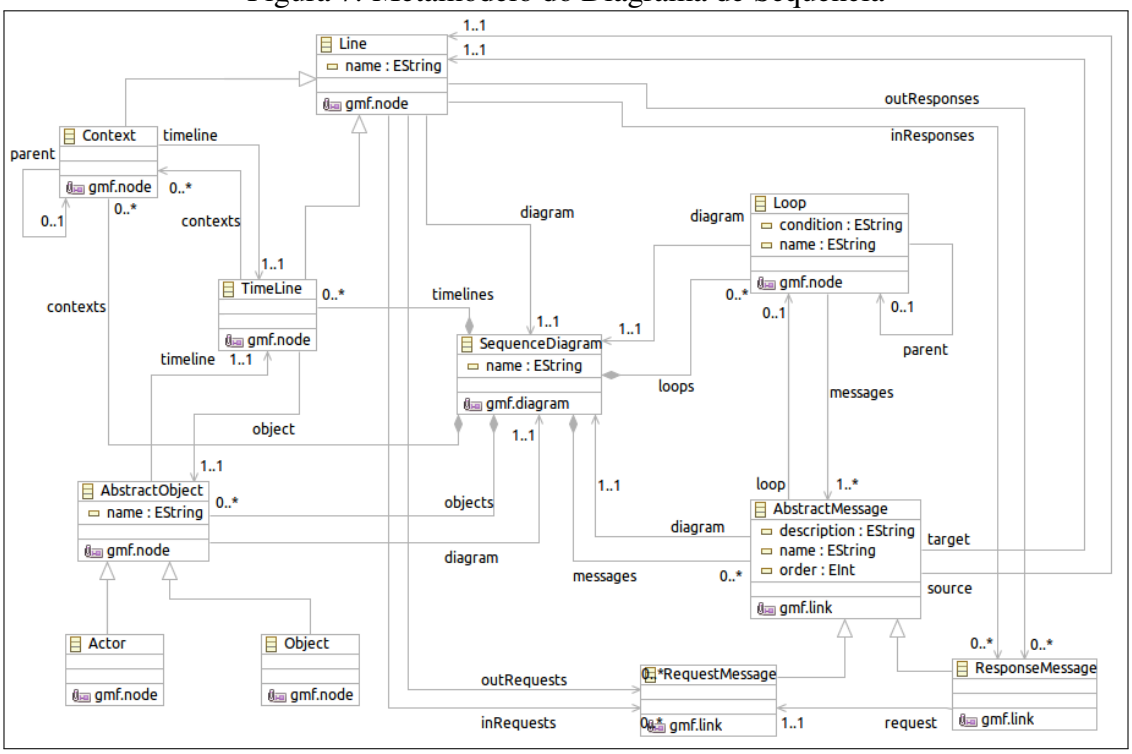
A seguir são descritas, em detalhes, cada uma das atividades desenvolvidas na construção do projeto da abordagem prática da disciplina.

Metamodelagem: O processo de metamodelagem foi realizado utilizando o Eclipse Modeling Framework (EMF)[3], que é uma extensão da IDE Eclipse[4], desenvolvido para simplificar o carregamento, manuseio e armazenamento de modelos. O EMF permite a modelagem do domínio utilizando um metamodelo graficamente. Pelo editor pode-se criar um diagrama com classes e seus relacionamentos que representam elementos dos

[3] Eclipse EMF: http://www.eclipse.org/modeling/emf/docs/
[4] Eclipse: http://www.eclipse.org

modelos que se deseja manipular. Por fim, um metamodelo com extensão *.ecore* é gerado automaticamente. O Ecore é baseado em um padrão de descrição de metamodelo, que por sua vez é um subconjunto da EMOF (Essential MOF)[5], um subconjunto do padrão da *Object Management Group* (OMG [6]) [Brambilla et al. 2012].

Figura 7. Metamodelo do Diagrama de Sequência



O metamodelo do DS concebido nesse estudo de caso foi desenvolvido tomando como base os metamodelos referentes ao mesmo diagrama, que encontram-se descritos em [Li et al. 2014] e [Shen et al. 2008]. A Figura 7 ilustra o metamodelo do DS simplificado, desenvolvido como um componente utilizando a ferramenta EMF.

No metamodelo ilustrado na Figura 7 é possível identificar os elementos pertencentes ao contexto do diagrama de sequência e seus devidos relacionamentos. A entidade **SequenceDiagram**, que representa a estrutura maior ou o próprio diagrama, é composta por coleções de objetos das entidades **AbstractObject**, que deriva em **Actor** e **Object**; **AbstractMessage**, que tem derivações em **RequestMessage** e **ResponseMessage**; **Line**, que é especializada por **TimeLine** e **Context**; e por fim objetos do tipo **Loop**.

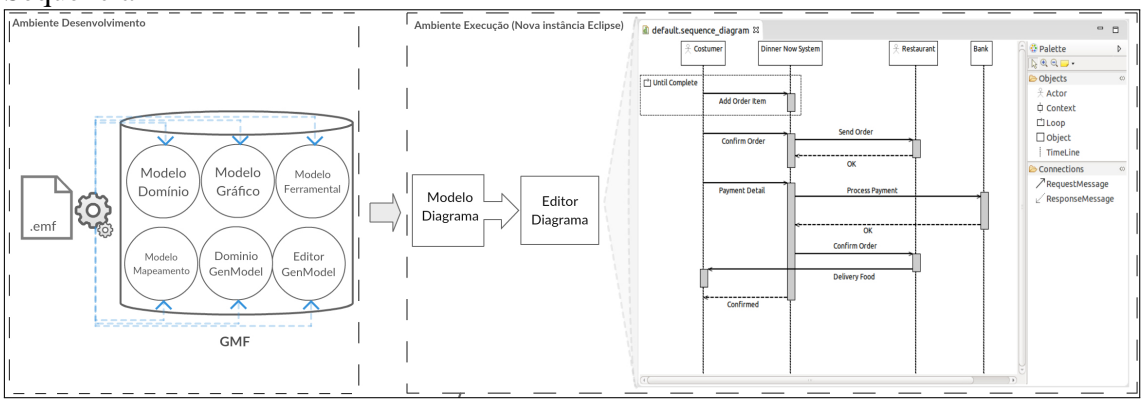
Sintaxe Concreta Gráfica: Para definir a sintaxe gráfica do DS, faz-se necessário definir a correspondência dos elementos concretos existentes no metamodelo Ecore com seus respectivos elementos gráficos. Pode-se utilizar a ferramenta Eugenia[7] baseada no Eclipse, que fornece um conjunto de anotações sobre o metamodelo para gerar mode-

[5] Meta-Object Facility (MOF): http://www.omg.org/mof/
[6] OMG: http://www.omg.org/
[7] EuGENia: http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/

los de baixo nível suportados pelo EMF, e o *Graphical Modeling Framework* (GMF)[8] [Kolovos et al. 2010]. As anotações são baseadas na linguagem Emfatic[9] (extensão *.emf*), que permite trabalhar em mais alto nível escondendo detalhes complexos do GMF. O GMF é um framework *open source* amplamente utilizado no desenvolvimento dirigido por modelos, implementado em EMF [Herrmannsdoerfer 2011].

A Figura 8 mostra a estrutura geral para obter a sintaxe concreta gráfica, na qual um arquivo *.emf* gera seis partes fundamentais para a construção do editor GMF. Segundo El Kouhen *et al.* (2012), essas partes são: modelo de domínio baseado em *ecore*; modelo gráfico baseado em *Graphical Editor Framework* (GEF); modelo ferramental, que contém as ferramentas do editor, tais como menus e paletas; o modelo de mapeamento, que faz as ligações entre a semântica e as representações gráficas; domínio *genmodel*, o qual gera o código do modelo de domínio com EMF; e editor *genmodel*, arquivo final para gerar o editor gráfico GMF. Segundo Herrmannsdoerfer (2011), o modelo de mapeamento é transformado em um modelo diagrama, o qual pode ser inicializado no editor diagrama (onde o diagrama de sequência foi gerado).

Figura 8. Esquema básico para geração da Sintaxe Concreta Gráfica do Diagrama de Sequência



Sintaxe Concreta Textual: Nesse exemplo, para a criação da sintaxe concreta textual do modelo DS, foi utilizada a ferramenta Xtext[10], um *framework* para construção de linguagens, especialmente DSLs. O Xtext é integrado com a ferramenta EMF e utiliza os arquivos *.ecore* e *.genmodel* para gerar a linguagem especificada. Além do *Ecore*, o Xtext cria um *parser*, um *serializer* e um editor no Eclipse para o teste de sua nova linguagem ao fazer a montagem do projeto. Para a concepção da linguagem foi utilizado o mesmo objeto *ecore* construído no tópico 3.2.1 a partir do metamodelo da Figura 7.

A sintaxe concreta textual criada pode ser considerada uma DSL para o DS. Uma vez criada a DSL para o domínio do problema, já é possível criar instâncias para a utilização ou mesmo validação do modelo ou da linguagem criada. A Figura 9 mostra, de maneira resumida, um exemplo da utilização da sintaxe textual criada para o exemplo base na Figura 6. Na linha 1 tem-se a declaração do diagrama de sequência, nomeado

[8] GMF: http://www.eclipse.org/modeling/gmp/
[9] Emfatic Language: http://www.eclipse.org/epsilon/doc/articles/emfatic/
[10] Xtext: https://eclipse.org/Xtext/documentation/

de *sd*, e a linha 2 descreve o conjunto de mensagens que compõem o diagrama *sd*. Da linha 3 à 9, tem-se a definição de cada uma dessas mensagens, detalhando sua descrição, origem, destino, e *loop* associado. Na linha 12 representa-se o conjunto de objetos que fazem parte de *sd*, onde da linha 13 à 21 encontram-se as definições de cada um desses objetos, detalhando suas *timelines* associadas. A linha 23 detalha o conjunto de *loops* contido em *sd*, enquanto da linha 24 à 28, tem-se a definição de cada um desses *loops*, detalhando suas condições de parada e as mensagens que o compõem. Para simplificação do exemplo, alguns detalhes foram omitidos.

Figura 9. Exemplo de Sintaxe Concreta Textual do Diagrama de Sequência

lts_test.dsllts lts_test.dsllts test.dslsd »2

```
1  SequenceDiagram sd {
2      messages {
3          RequestMessage req1 {
4              description "Add Order Item"
5              source CustomerTL
6              target DinnerNowSystemTL
7              loop lp
8              diagram sd
9          }
10         //Demais declarações foram omitidas
11     }
12     objects {
13         Actor Customer {
14             timeline CustomerTL
15             diagram sd
16         },
17         Object DinnerNowSystem {
18             timeline DinnerNowSystemTL
19             diagram sd
20         }
21         //Demais declarações foram omitidas
22     }
23     loops {
24         Loop lp {
25             condition 'Until Complete'
26             messages (req1)
27             diagram sd
28         }
29     }
30     //Demais declarações foram omitidas
```

Transformação de Modelos: Segundo Muller *et al.* (2005), transformações entre modelos tornam possível a construção de pontes entre diversas linguagens de modelagem para problemas específicos. Eles podem ser comparados com os compiladores partindo do princípio em que eles podem traduzir ou converter modelos de um metamodelo em modelos oriundos de um outro metamodelo. Um exemplo disso seria a transformação de um modelo de classes em um modelo relacional.

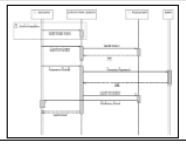
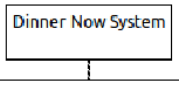
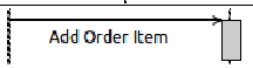
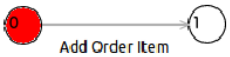
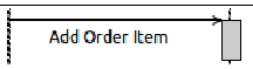
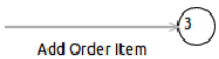
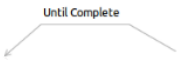
Nesta etapa, será apresentada a forma na qual um DS pode ser transformado em um LTS utilizando o *ATLAS Transformation Language* (ATL)[11], uma linguagem para a realização de transformações de modelos no contexto de Arquitetura Dirigida por Modelo [OMG 2016]. A linguagem consiste de regras (*rules*) que podem ser divididas em *Called Rules*, *Lazy Rules* e *Matched Rules*, que se baseiam fortemente na *Object Constraint Language* (OCL), e de auxiliadores (*helpers*), que são recursos da linguagem dedi-

[11] ATL: http://www.eclipse.org/atl/documentation/

cados para complementar transformações do modelo que não podem ser satisfeitos com OCL [Brambilla et al. 2012].

Primeiramente, foi realizado um mapeamento entre elementos de ambos os modelos, o qual encontra-se ilustrado na Figura 10. Na linha 1 tem-se a equivalência entre os modelos de fato, visto por meio de **SequenceDiagram** e **LTS**. Na linhas 6 e 7 encontra-se a equivalência entre objetos do tipo **RequestMessage** e **ResponseMessage** com uma união entre objetos **Transition** e **State**. Por fim, na linha 8, pode-se observar a equivalência entre um objeto **Loop** com um objeto **Transition**.

Figura 10. Mapeamento de equivalência de entidade entre o DS e LTS

Diagrama de Sequência		**LTS**	
Entidade	**Sintaxe Gráfica**	**Entidade**	**Sintaxe Gráfica**
SequenceDiagram		LTS	
Actor		--	--
Object		--	--
TimeLine		--	--
Context		--	--
RequestMessage (É a primeira mensagem do DS)		InitialState + Transition + State	
RequestMessage (Não é a primeira mensagem do DS)		Transition + State	
ResponseMessage		Transition + State	
Loop		Transition	

De acordo com esse mapeamento, verificou-se a necessidade de se construir um conjunto de *Matched Rules* e *Lazy Rules* para converter automaticamente cada elemento com relação de equivalência. A Figura 11 contempla a principal *Rule*, que converte um objeto do tipo **SequenceDiagram** em um objeto **LTS**. Em mais detalhes, na linha 1 e 2 são descritos os metamodelos envolvidos na transformação. Na linha 8 define-se o módulo que conterá as regras de transformação. Na linha 9 informa-se que será criado um LTS com nome OUT a partir de um **SequenceDiagram** nomeado de IN. A linha 20 inicia a definição da regra de transformação. Entre as linhas 21 e 26 são realizadas as operações de equivalência, nas quais a partir de um objeto *s* do tipo **SequenceDiagram** será obtido

um objeto *l* do tipo LTS, e que o elemento *messages* em *s* será tranformado em *transitions* em *l*. Entre as linhas 27 e 32 são feitas operações imperativas, sendo que na linha 29 é feita uma busca pelo primeiro objeto da coleção *messages* em *s*, para transformar no objeto *initialState* em *l*.

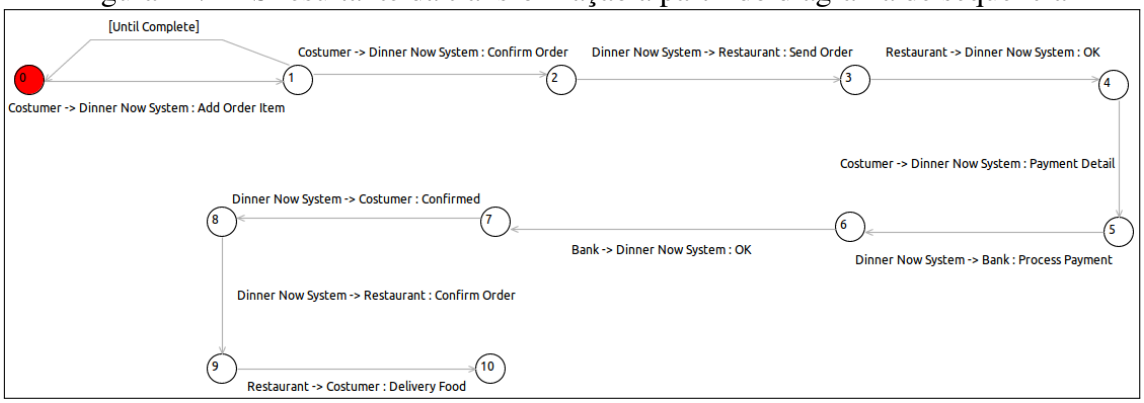
Após a execução da transformação do modelo, tem-se como resultado um modelo LTS, o qual pode ser visto no diagrama editor na Figura 12.

Figura 11. *Rule* principal utilizada para transformar o Diagrama de Sequência em LTS

SequenceDiagram2LTS.atl

```
1  -- @path LTS=/lts/model/lts.ecore
2  -- @path SequenceDiagram=/sequence/model/sequence.ecore
3  --Detalhes Omitidos
8  module SequenceDiagram2LTS;
9  create OUT : LTS from IN : SequenceDiagram;
10
20 rule SequenceDiagram2LTS {
21     from
22         s : SequenceDiagram!SequenceDiagram
23     to
24         l : LTS!LTS(
25             transitions <- s.messages
26         )
27     do {
28         --Creating Initial State
29         for (msg in s.messages -> select(msg | msg.oclIsTypeOf(SequenceDiagram!RequestMessage) and msg.order = 1)) {
30             l.initialState <- thisModule.RequestMessage2InitialState(msg);
31         }
32         --Detalhes Omitidos
82     }
83 }
35 --Detalhes Omitidos
```

Figura 12. LTS resultante da transformação a partir do diagrama de sequência



Geração de Código Para o presente exemplo, foi considerada apenas a geração de código na linguagem Java a partir do DS da Figura 6 utilizando a ferramenta Acceleo[12]. O Acceleo é uma ferramenta de código aberto disponibilizada como um *plugin* para o Eclipse e tem como objetivo permitir a geração de código e artefatos de texto, promovendo assim transformações M2T.

Dentre os diversos arquivos criados para realizar transformação M2T, pode-se destacar o *generatedMainClass.mtl*, que representa um *template* de geração de código da classe *Main* e pode ser visto na Figura 13. A linha 2 contém a declaração do módulo. Entre as linhas 4 e 7 é realizada a importação de todos os módulos dos quais o presente

[12] Acceleo: http://wiki.eclipse.org/Acceleo

módulo é dependente. Na linha 9 tem-se a declaração do *template* responsável pela geração da classe *Main*. A linha 10 indica que será criado um arquivo para esse *template*, não somente texto. A linha 11 escreve no arquivo o nome da *package* da classe, e a linha 13 inicia a construção. A linha 14 faz uma chamada ao *template* que gera o método *main()*. As linhas 16 e 17 encerram o arquivo gerado e *template* respectivamente. Na linha 19 inicia-se o *template* de geração do método *main* e entre as linhas 21 e 24 são executados o *templates* que preenchem o corpo do método.

Figura 13. Template para geração da classe Main

*generateMainClass.mtl

```
1  [comment encoding = UTF-8 /]
2  [module generateMainClass('sequence')]
3
4  [import org::eclipse::acceleo::module::mde::common::queries/]
5  [import org::eclipse::acceleo::module::mde::common::generateMainClassVariableDeclare/]
6  [import org::eclipse::acceleo::module::mde::common::generateMainClassVariableSetting/]
7  [import org::eclipse::acceleo::module::mde::common::generateMainClassCallDiagramMethod /]
8
9  [template public generateMainClass(aSequenceDiagram : SequenceDiagram)]
10 [file (aSequenceDiagram.getFileName(), false, 'UTF-8')]
11 [aSequenceDiagram.getPackage()/]
12
13 public class Main {
14     [aSequenceDiagram.generateMainMethod()/]
15 }
16 [/file]
17 [/template]
18
19 [template public generateMainMethod(aSequenceDiagram : SequenceDiagram)]
20 public static void main (String [('[]')/]args) {
21     [aSequenceDiagram.generateMainClassVariableDeclare()/]
22
23     [aSequenceDiagram.generateMainClassVariableSetting()/]
24     [aSequenceDiagram.generateMainClassCallDiagramMethod()/]
25 }
26 [/template]
```

Como resultado da execução dos *templates*, a Figura 14 apresenta um dos códigos gerados. Nessa figura observa-se a estrutura exatamente como descrita no arquivo *generatedMainClass.mtl*, no qual se tem a definição da classe *Main* e o método *main()* contendo a declaração e inicialização de objetos, a definição das referências entre os objetos e chamada ao método principal definido no DS.

O arquivo completo com os todos os códigos relativos ao exemplo mostrado nesta seção encontra-se disponível para *download* no repositório de código do Grupo de Engenharia de Software e Sistemas Distribuídos (GESAD)[13].

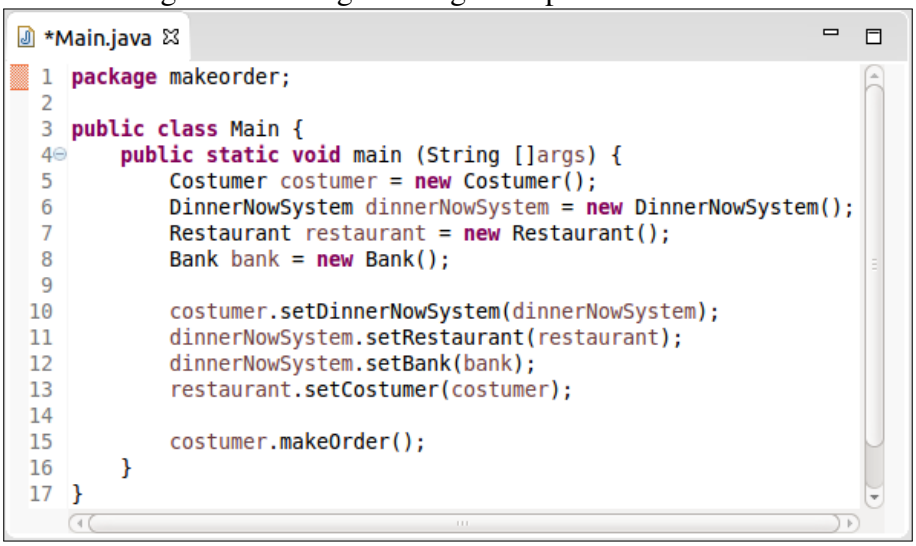
4. Avaliação da Disciplina

Para poder avaliar a metodologia de ensino aplicada na disciplina, foi elaborado um questionário, que foi disponibilizado na internet para todos os alunos que participaram das duas versões da disciplina. O processo de avaliação aconteceu de acordo com os seguintes passos:

1. Identificação do público-alvo
2. Elaboração das perguntas do questionário
3. Envio do questionário para os participantes
4. Análise dos dados coletados

[13] http://gesad.uece.br/codigos-fonte/

Figura 14. Código fonte gerado para a classe Main



O questionário utilizado nessa avaliação foi inspirado no apresentado em [Clarke et al. 2009] e possuía, ao todo, 19 perguntas divididas em quatro seções: uma com seis perguntas sobre a percepção do aluno em relação à modelagem de sistemas e sua experiência técnica, outra com seis perguntas sobre a metodologia adotada, a terceira com seis questionamentos em relação ao uso das ferramentas trabalhadas na disciplina, e a última contendo apenas uma pergunta subjetiva sobre críticas ou sugestões à disciplina.

Algumas perguntas tinham como resposta apenas as opções Sim ou Não, enquanto em outras foi utilizada uma escala *Likert*, consistindo de Discordo Plenamente, Discordo, Neutro, Concordo, e Concordo Plenamente. O questionário, que foi respondido por 18 pessoas (o universo era de 19 pessoas), pode ser visto no anexo A.

Vale ressaltar que embora seja uma disciplina da área de Engenharia de Software, alunos de outras áreas de pesquisa, como Redes de Computadores, Sistemas Distribuídos e Algoritmos, também a cursaram e estão dentre os respondentes do questionário.

4.1. Resultados Obtidos

A Figura15 mostra o resultado das cinco primeiras perguntas. 61,1% dos respondentes disseram que já tinham experiência com modelagem de sistemas (Q1) e 50% trabalham com desenvolvimento de software (Q2). Mesmo assim, 94,4% responderam que a disciplina de MDE melhorou sua compreensão em relação à importância da modelagem de software (Q3) e 77,8% responderam que a disciplina melhorou sua forma de usar ou fazer a modelagem de sistema (Q4). Além disso, 66,7% informaram que pretendem usar conceitos, ferramentas ou linguagens MDE na sua pesquisa ou trabalho (Q5).

Quando perguntados se a disciplina impactou ou influenciou na sua formação profissional ou acadêmica (Q6), 81,2% concordaram ou concordaram plenamente, enquanto 18,2% se disseram neutros, conforme mostra a Figura 16. Isso é um dado animador e corrobora os dados das questões 3, 4 e 5, mostrando que a disciplina ajudou no aprimoramento técnico e/ou científico dos alunos.

Em relação à metodologia adotada (parte teórica e prática), a Figura 17 mostra que

Figura 15. Resultado das questões 1 a 5

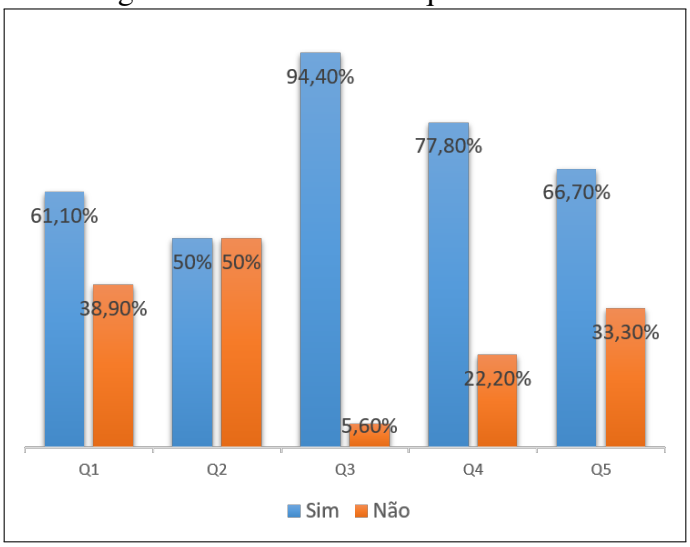
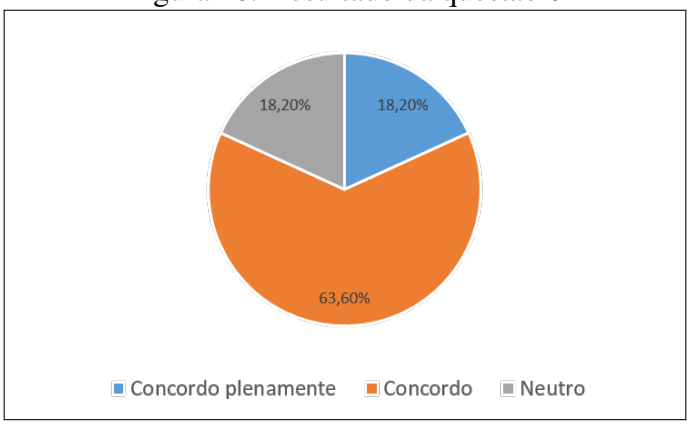


Figura 16. Resultado da questão 6



a mesma foi aprovada por 100% dos alunos (Q7), enquanto 88,9% dos alunos concordam que a metodologia ajudou na compreensão dos conceitos de MDE (Q8), e 11,1% se disseram neutros. Isso demonstra uma excelente aceitação dos alunos em relação à forma como essa disciplina foi ministrada.

Levando em conta a qualidade do material adotado na disciplina, 100% dos alunos concordaram que os artigos ajudaram a entender os conceitos e a aplicação da MDE (Q9). Sobre as ferramentas adotadas (Q10), 90,9% concordaram que a escolha delas foi adequada (dos quais mais da metade afirmaram que concordavam plenamente), enquanto 9,1% ficaram neutros. Esses dados são ilustrados na Figura 18.

Finalizando a segunda seção do questionário, a Figura 19 aponta que 83,3% concordaram que os trabalhos teóricos (resenhas, resumos, e apresentações) foram motivantes (Q11), enquanto 5,6% se disseram neutros e 11,1% discordaram. Essa taxa de discordância pode estar relacionada ao fato que, geralmente, a parte teórica é menos interessante se comparada com a parte prática. Por outro lado, 94,4% concordaram que os trabalhos práticos (implementações) foram motivantes (Q12), enquanto 5,6% discordaram plenamente, o que também é normal, uma vez que nem todo aluno se interessa por tarefas de

desenvolvimento.

Figura 17. Resultado das questões 7 e 8

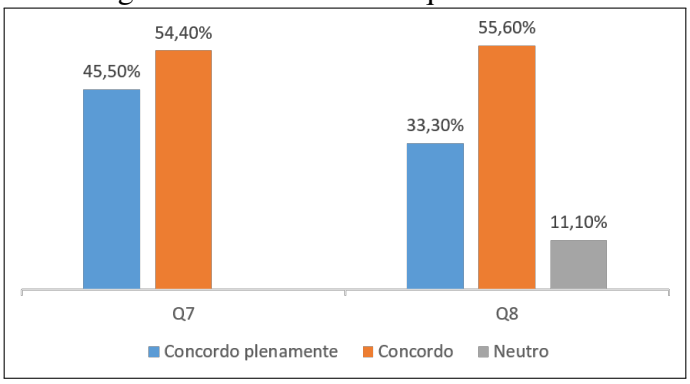
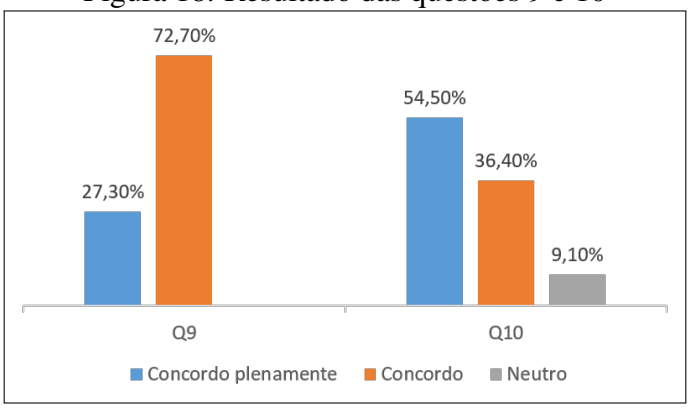


Figura 18. Resultado das questões 9 e 10



A Figura 20 mostra os resultados das perguntas contidas na terceira parte do questionário, as quais eram relacionadas à facilidade de uso das ferramentas e à documentação disponível. Nota-se que, em geral, poucos alunos consideraram as ferramentas fáceis de usar (colunas Concordo Plenamente e Concordo), tendo o Eugênia a maior porcentagem de concordância (38,9%) e o ATL a mais baixa (apenas 16,7%). Por outro lado, os alunos também não consideraram as ferramentas de todo difíceis de usar (colunas Discordo e Discordo Plenamente), sendo o ATL o mais difícil (38,9%) e o Acceleo o menos difícil (22,3%). A maioria dos alunos opinou que as ferramentas não eram tão fáceis de usar, mas também não eram tão difíceis (coluna Neutro), o que provavelmente indica que eles tiveram alguma dificuldade no início, mas logo conseguiram se adaptar. Isso vale especialmente para o Acceleo, para o qual 50% dos respondentes se enquadra nessa situação.

Metade dos respondentes (50%) concordou que a documentação disponível sobre as ferramentas ajudou no aprendizado das mesmas, enquanto apenas 16,7% não concordaram e os outros 33,3% se disseram neutros. Mesmo sendo útil, essa documentação não facilitou totalmente a utilização das ferramentas, uma vez que, como mostrado anteriormente, poucos alunos acharam-nas fáceis de usar.

Por fim, foi perguntado se os alunos teriam alguma sugestão ou crítica a fazer de forma a melhorar a disciplina. Apenas 3 pessoas responderam e suas respostas encontram-

Figura 19. Resultado das questões 11 e 12

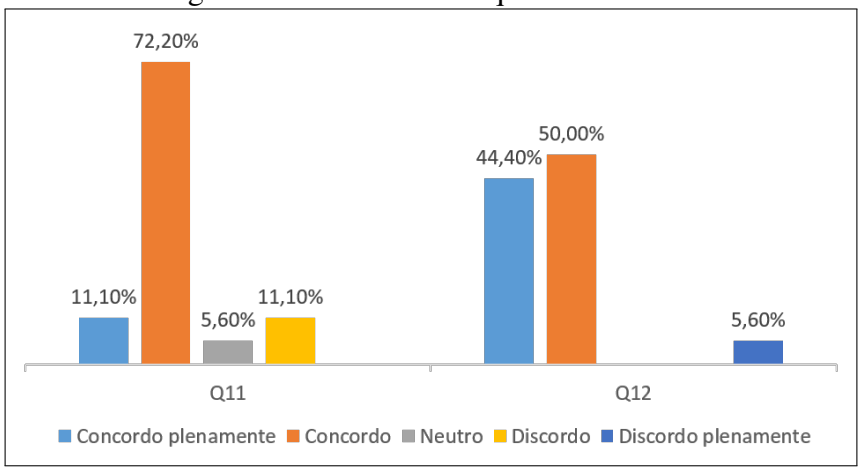


Figura 20. Resultado das perguntas sobre as ferramentas

Ferramenta	Conc. Plenamente	Concordo	Neutro	Discordo	Disc. Plenamente
EMF	0%	33,3%	33,3%	22,2%	11,1%
Eugênia	0%	38,9%	33,3%	16,7%	11,1%
Xtext	5,6%	22,2%	38,9%	22,2%	11,1%
ATL	0%	16,7%	44,4%	27,8%	11,1%
Acceleo	0%	27,8%	50%	16,7%	5,6%

se na Figura 21. Essas respostas foram utilizadas para melhorar a disciplina, conforme será mostrado na Seção 5.

4.2. Ameaças à validade

Nesta seção os principais riscos à validade que podem afetar os resultados obtidos neste trabalho são discutidos.

Primeiramente, pode-se apontar ameaças quanto à amostragem. Como a disciplina foi lecionada, até o momento da escrita deste artigo, apenas duas vezes, o total de alunos que responderam ao questionário foi de 18 pessoas. Apesar desse número ser relativamente pequeno, ele totaliza quase todos os alunos que cursaram a disciplina, uma vez que apenas um aluno não respondeu à pesquisa.

Outra ameaça que pode ser levantada diz respeito à criação do questionário e o teor das perguntas. Apesar do questionário se basear em outro disponível em [Clarke et al. 2009], o mesmo foi estendido com novas questões para capturar informações relevantes para o contexto deste trabalho. Para minimizar esse risco, as questões foram elaboradas de acordo com as instruções apresentadas em [Kitchenham and Pfleeger 2002], incluindo questões objetivas e subjetivas.

Por fim, o nível de confiança dos alunos em relação às suas respostas também pode ser considerado uma ameaça à análise dos resultados, uma vez que eles podem não se sentir à vontade para fazer alguma crítica à disciplina. Para minimizar esse risco, o questionário foi respondido anonimamente.

Figura 21. Sugestões e críticas à disciplina

Sugestão/Crítica	Descrição
S1	Alguns dos primeiros artigos introduziram o assunto ainda de uma maneira complexa. Talvez seja interessante sugerir os artigos seguindo uma ordem crescente de complexidade.
S2	Em cada parte das etapas da prática, é interessante que, se possível, alunos das turmas anteriores dêem seus relatos de experiência e possíveis atalhos (configurações e entendimento das ferramentas), pois desta forma o nível de conhecimento adquirido se torna muito melhor e mais rápido para quem está iniciando. Além dos possíveis resultados serem bem mais elaborados, uma vez que o tempo "perdido" nos entendimentos gerais das ferramentas, pode ser agregado a qualidade dos trabalhos em cada etapa.
S3	Por ser uma disciplina relativamente nova na UECE e seu ensino ainda ser pouco difundido em outras universidades, acredito que os materiais e resultados produzidos pelos alunos ao longo da disciplina poderiam reunidos e disponibilizados como uma biblioteca da disciplina. Isso serviria de base para os alunos das turmas seguintes, bem como ajudaria na própria evolução da disciplina. Além disso poderia servir como base de conhecimento para a difusão de disciplinas de temática semelhante em outras instituições.

5. Dificuldades e Soluções Adotadas

O ensino de MDE ainda é um desafio, tanto para docentes quanto para discentes. Nesta seção são abordadas as dificuldades encontradas para lecionar e aprender sobre os conceitos, ferramentas e práticas da MDE, e as lições aprendidas nesse processo.

Apesar de existir um vasto material que pode ser utilizado na primeira parte do curso, uma vez que há muitas publicações disponíveis sobre o assunto, nem sempre o material é didático o suficiente para introduzir o aluno no assunto. Adicionalmente, há uma grande dificuldade na parte prática do conteúdo. Isso se deve principalmente a três fatores: (i) pouco material que mostre exemplos implementados das etapas do processo de MDE, o que faz com que o processo de aprendizado seja muito baseado em tentativa e erro e, por consequência, mais demorado; (ii) dificuldade com a configuração das ferramentas, causado especialmente por documentação oficial pouco intuitiva e ausência de tutoriais, como foi também relatado por Clarke *et al.* (2009) e Batory *et al.* (2013); e (iii) falta de fóruns especializados para discutir problemas relacionados ao tema, o que ajudaria na resoluções de dúvidas e compartilhamento de informações.

Para contornar essas situações desfavoráveis, foram adotadas as seguintes medidas:

- *Leitura e discussão de artigos introdutórios*: dado que a maioria dos alunos inicia a disciplina sem experiência em MDE, a parte teórica inicia-se com a leitura de artigos introdutórios sobre abstração e modelagem de software, para que então possa-se começar a discutir artigos mais relacionados à MDE propriamente dita. Dessa forma, a sugestão S1 da Figura 21 foi atendida.
- *Disponibilização dos exercícios praticados nas aulas*: com essa medida, os alunos puderam se beneficiar com os exemplos praticados nas turmas anteriores. Conse-

quentemente, a qualidade dos trabalhos realizados cresceu, permitindo explorar modelos mais elaborados na turma seguinte. Por exemplo, na primeira turma, o processo de MDE foi exercitado com o Diagrama de Atividades da UML como modelo de origem, o qual é mais simples que o Diagrama de Sequência utilizado na turma seguinte e explicado neste artigo.

- *Compartilhamento de conhecimento*: alunos das turmas anteriores eram selecionados para explicar como resolveram alguns dos seus exercícios práticos, relatando inclusive os problemas encontrados na configuração e utilização das ferramentas, e como eles foram solucionados. Isso diminuiu o tempo de realização dos trabalhos e permitiu uma troca de experiências entre os alunos. Essa medida atendeu à sugestão S2 de melhoria da disciplina mostrada na Figura 21.
- *Criação de uma lista de discussão sobre MDE*: para que o conhecimento gerado e trocado não ficasse perdido, uma lista de discussão e uma página de FAQ foram criadas. Com isso, as dúvidas mais frequentes puderam ser registradas e respondidas, bem como as novos questionamentos puderam ser discutidos.
- *Elaboração de um relatório técnico*: esse documento foi criado para ajudar os alunos a entenderem melhor o assunto, descrevendo as ferramentas utilizadas e como foi implementada cada etapa do processo. Esse documento está disponível para *download*[14]. Com isso, foi atendida a sugestão S3 contida na Figura 21.

6. Trabalhos Relacionados

Existem vários trabalhos que relatam a experiência de ensinar MDE em cursos de graduação e pós-graduação. Em [Clarke et al. 2009], os autores mostram como MDE foi integrada à disciplina de Projeto de Software em um curso de pós-graduação da Florida International University. Essa integração se deu a parir do uso da DSL *Communication Modeling Language* (CML), usada para modelar aplicações de comunicação unificada (e.g., vídeo conferências e mensagens instantâneas) e da plataforma *Communication Virtual Machine* (CVM), usada para modelar e realizar a comunicação dos modelos CML. Usando as técnicas de MDE, os alunos puderam gerar modelos executáveis a partir dos modelos criados em CVM. Para determinar o impacto do uso da MDE, foi realizada uma pesquisa com os alunos após a disciplina. Como resultado, a grande maioria dos alunos afirmou que MDE ajudou no entendimento de projeto de software e no uso de abstração para a criação de modelos de software complexos. Diferentemente da abordagem aqui apresentada, os autores usaram modelos bem específicos e pouco conhecidos, enquanto são utilizados os modelos DS e LTS no curso ministrado na UECE, os quais são mais difundidos e utilizados.

Blay-Fornarino (2008) retrata sua experiência ao ensinar MDE para alunos de pós-graduação utilizando uma abordagem baseada em projetos, que devido ao bom desempenho e interesse dos alunos, foi incorporado no curso de graduação. Nessa abordagem os alunos são divididos em grupos, cada grupo sendo supervisionado por um professor, que faz o papel de cliente. Os grupos trabalham em projetos complementares e, ao final, apresentam sua solução e compartilham suas experiências com os outros. Apesar da abordagem também usar um processo semelhante ao proposto neste artigo, não fica explícito que ferramentas e linguagens são usadas.

[14] http://www.gesad.uece.br/publicacoes/relatorio-tecnico/

Uma forma alternativa para ensinar e demonstrar os princípios da MDE para alunos de graduação é apresentada em [Batory et al. 2013], na qual os autores usaram bancos de dados relacionais para expressar modelos e metamodelos do sistema, Prolog para expressar transformações e restrições M2M, ferramentas Java (MDELite[15]) para implementar transformações M2T, e linguagens de *script shell* para compor as transformações. A ideia de usar essas tecnologias surgiu a partir de uma experiência frustrada no uso de ferramentas MDE baseadas em Eclipse, as quais vários alunos tiveram dificuldades em instalar e configurar. Segundo os autores, após o uso dessa abordagem, os alunos passaram a ter bem menos problemas e melhoraram o aprendizado dos conceitos de MDE. Essa experiência é interessante, especialmente pelo fato de partir de modelos de banco de dados e, a partir dela, pretende-se adotar a ferramenta MDELite na disciplina ministrada na UECE.

O trabalho de Burden *et al.* (2012) apresenta uma ideia inovadora no ensino de MDE, que consiste em aulas ministradas por dois professores (*pair lecturing)* com o objetivo de encorajar os estudantes a participar mais ativamente em desenvolver modelos durante as aulas. Os autores avaliaram o impacto dessa mudança na forma de ensino por meio de um questionário, no qual os alunos declararam que se sentiram mais ativos durante as aulas e lembraram mais os conceitos do que em relação a aulas tradicionais.

Em [Schmidt et al. 2014], os autores expõem a experiência de ensinar geração automática de código a partir de modelos. A metodologia de ensino se baseia em um conjunto de exercícios sequenciais nos quais os alunos aprendem metamodelagem, transformação de modelos UML (representados em *eXtensible Markup Language* (XML)), e geração de código Java. Apesar de seguir um processo MDE bem semelhante ao apresentado neste artigo, os autores usam apenas a linguagem PHP e tecnologias XML (XMI[16], XQuery[17] e XSLT[18]) para implementar todas as etapas do processo, diferentemente das linguagens e ferramentas utilizadas na abordagem adotada neste trabalho.

Poruban *et al.* (2014) elaboraram um curso de desenvolvimento orientado por modelos que abrange diversas técnicas e princípios, podendo estas ser usadas em diferentes combinações e projetos de diferentes escalas, em vez de centralizar em uma única ferramenta. O propósito era apresentar a MDE a partir da perspectiva de um programador como um instrumento pragmático para resolver problemas concretos no processo de desenvolvimento.

Uma discussão a respeito da experiência de ministrar uma disciplina de Engenharia de Software Dirigida por Modelos para alunos de graduação é retratada em [Hamou-Lhadj et al. 2009]. Os autores dividem o trabalho em duas linhas de frente. A primeira tem seu foco na apresentação de conceitos de MDE, juntamente com ferramentas que dão suporte à abordagem. Já na segunda, os autores descrevem a experiência de ensino, apresentando detalhes da estrutura do curso oferecido aos alunos, elencando assuntos cobertos e biografia utilizada. Ainda nessa linha, os autores fazem um relato sobre os desafios e dificuldades enfrentados pelos estudantes, como a enorme quantidade de novos conceitos e termos, inabilidade de aplicar conhecimentos anteriores ao trabalhar com

[15] MDELite: https://www.cs.utexas.edu/users/schwartz/MDELite/index.html
[16] XML Metadata Interchange
[17] XML Query
[18] eXtensible Stylesheet Language for Transformation

um paradigma totalmente diferente do convencional, além da escassez de bons livros que abordem o assunto de maneira completa e ferramentas de suporte eficientes e com vasta documentação.

Finalmente, Akayama *et al.* (2013) apresentam um estudo desenvolvido com base na experiência de ensino de um curso de modelagem UML. O objetivo desse estudo era avaliar os benefícios da utilização de ferramentas de geração automática de código a partir de modelos UML, as quais os autores chamam de ferramentas MDD, no processo de ensino. Nesse experimento, os alunos do curso foram divididos em dois grupos, sendo que apenas o primeiro utilizou ferramentas MDD. Ambos os grupos foram submetidos ao mesmo exercício e, ao final, percebeu-se que o segundo grupo teve como resultado a criação de modelos com melhores representações e nomes mais apropriados. Por outro lado, o primeiro grupo teve como resultado modelos mais precisos e que cobriam melhor o problema do que os do segundo grupo.

7. Conclusão

Devido à crescente complexidade dos sistemas de informação atuais, uma alternativa que vem ganhando força é o uso da Engenharia Dirigida por Modelos como forma de abstrair detalhes técnicos do sistema e focar em soluções de mais alto nível, o que permite raciocinar sobre melhores formas de modelar o sistema em desenvolvimento. Apesar dessa abordagem já ser aplicada em diversos domínios, ela ainda não se tornou um padrão de fato na construção de sistemas possivelmente devido à forma como é ensinada nas universidades [Whittle et al. 2014][Cowling 2003]. No Brasil, em especial, seu ensino ainda é insipiente, o que faz com que alunos não conheçam suas práticas (muitos sequer ouviram falar sobre o assunto) ou não utilizem todo o potencial que a MDE possui. Portanto, expandir e melhorar o ensino de MDE pode ser um fator relevante para a formação de profissionais mais capacitados e a produção de sistemas de maior qualidade.

Este artigo apresentou um relato da experiência no ensino de Engenharia Dirigida por Modelos no curso de Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará. Foram apresentados os principais conceitos teóricos sobre o assunto, bem como um exemplo prático da aplicação desses conceitos e utilização das principais ferramentas. Também foi apresentada uma avaliação da disciplina realizada utilizando-se um questionário respondido pelos alunos. As dificuldades encontradas no decorrer da disciplina foram discutidas e algumas soluções para superá-las foram apresentadas.

Pode-se citar três contribuições principais deste trabalho: (i) a disponibilização de uma metodologia de ensino, que foi avaliada muito positivamente pelos alunos e pode ser replicada em cursos de MDE por outras universidades brasileiras; (ii) apresentação de possíveis soluções para lidar com os problemas comumente encontrados no ensino de MDE, conforme mostrado na seção 5; e (iii) a disponibilização de um relatório técnico sobre a aplicação do processo de MDE e detalhes de utilização das ferramentas, bem como códigos-fonte dos exemplos praticados da disciplina.

Com isso, espera-se disseminar o ensino de Engenharia Dirigida por Modelos em cursos de graduação e pós-graduação na área Sistemas de Informação e afins, além da melhoria no ensino de modelagem de software de maneira geral. Também espera-se aumentar o uso dessa abordagem, uma vez que exemplos práticos de uso de ferramentas e

linguagens utilizadas em diferentes etapas do processo de MDE também foram disponibilizados, atacando assim um dos grandes problemas nessa área, que é a dificuldade de encontrar material especializado.

Como trabalhos futuros, pretende-se introduzir um curso de MDE também na graduação, para que os alunos já possam chegar no mestrado com uma base mais sólida sobre o assunto. Também pretende-se utilizar mais de uma ferramenta em cada etapa do processo de MDE, com o intuito de facilitar a comparação entre tecnologias. Outro trabalho importante é manter os dados dessa pesquisa sempre atualizados, o que pode ser feito pela aplicação do questionário sempre que uma nova turma concluir a disciplina. Por fim, também planeja-se abordar, de maneira prática, outros tópicos de MDE que atualmente só conseguem ser discutidos do ponto de vista teórico, como a criação de perfis UML.

Referências

[Akayama et al. 2013] Akayama, S., Kuboaki, S., Hisazumi, K., Futagami, T., and Kitasuka, T. (2013). Development of a modeling education program for novices using model-driven development. In *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, WESE '12, pages 4:1–4:8. ACM.

[Arisholm et al. 2006] Arisholm, E., Briand, L. C., Hove, S. E., and Labiche, Y. (2006). The impact of uml documentation on software maintenance: An experimental evaluation. *Software Engineering, IEEE Transactions on*, 32(6):365–381.

[Batory et al. 2013] Batory, D., Latimer, E., and Azanza, M. (2013). Teaching model driven engineering from a relational database perspective. In *Model-Driven Engineering Languages and Systems*, pages 121–137. Springer.

[Bézivin 2005] Bézivin, J. (2005). On the unification power of models. *Software & Systems Modeling*, 4(2):171–188.

[Blay-Fornarino 2008] Blay-Fornarino, M. (2008). Project-based teaching for model-driven engineering. In *Proceedings of the 4th Educators Symposium at MoDELS*, pages 1–15.

[Booch 2004] Booch, G. (2004). *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley, Redwood City, CA, USA.

[Brambilla et al. 2012] Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers.

[Burden et al. 2012] Burden, H., Heldal, R., and Adawi, T. (2012). Pair lecturing to model modelling and encourage active learning. pages 1–9.

[Cartaxo et al. 2007] Cartaxo, E. G., Neto, F. G. O., and Machado, P. D. L. (2007). Test case generation by means of uml sequence diagrams and labeled transition systems. In *IEEE International Conference on Systems, Man and Cybernetics (ISIC 2007)*, pages 1292–1297.

[Clarke et al. 2009] Clarke, P. J., Wu, Y., Allen, A. A., and King, T. M. (2009). Experiences of teaching model-driven engineering in a software design course. In *Proceedings of the 5th Educators' Symposium at MODELS 2009*, pages 6–14.

[Cowling 2003] Cowling, A. (2003). Modelling: a neglected feature in the software engineering curriculum. In *Proceedings. 16th Conference on Software Engineering Education and Training, 2003 (CSEE T 2003).*, pages 206–215.

[da Silva 2010] da Silva, W. C. (2010). Gerência de interfaces para sistemas de informação: uma abordagem baseada em modelos. Master's thesis, Universidade Federal de Goiás. Instituto de Informática.

[El Kouhen et al. 2012] El Kouhen, A., Dumoulin, C., Gerard, S., and Boulet, P. (2012). Evaluation of Modeling Tools Adaptation. Technical report.

[Fondement 2007] Fondement, F. (2007). *Concrete syntax definition for modeling languages*. PhD thesis, École Polytechnique Fédérale de Lausanne, France.

[France and Rumpe 2007] France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society.

[Hamou-Lhadj et al. 2009] Hamou-Lhadj, A., Gherbi, A., and Nandigam, J. (2009). The impact of the model-driven approach to software engineering on software engineering education. In *Sixth IEEE International Conference on Information Technology: New Generations, Las Vegas, Nevada, USA*, pages 719–724.

[Herrmannsdoerfer 2011] Herrmannsdoerfer, M. (2011). GMF: A model migration case for the transformation tool contest. In *Proceedings Fifth Transformation Tool Contest, TTC 2011, Zürich, Switzerland*, pages 1–5.

[Hofstader 2006] Hofstader, J. (2006). Model-driven development. Disponível em https://msdn.microsoft.com/en-us/library/aa964145.aspx. Acessado em 15/05/2016.

[Hutchinson et al. 2011] Hutchinson, J., Rouncefield, M., and Whittle, J. (2011). Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 633–642, New York, NY, USA. ACM.

[Keller 1976] Keller, R. M. (1976). Formal verification of parallel programs. *Commun. ACM*, 19:371–384.

[Kitchenham and Pfleeger 2002] Kitchenham, B. A. and Pfleeger, S. L. (2002). Principles of survey research part 2: Designing a survey. *SIGSOFT Softw. Eng. Notes*, 27(1):18–20.

[Kleppe et al. 2003] Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, Boston, MA, USA.

[Kolovos et al. 2010] Kolovos, D. S., Rose, L. M., Abid, S. B., Paige, R. F., Polack, F. A., and Botterweck, G. (2010). Taming emf and gmf using model transformation. In *Model Driven Engineering Languages and Systems*, pages 211–225. Springer.

[Krogstie 2012] Krogstie, J. (2012). *Model-based development and evolution of information systems: A Quality Approach*. Springer Science & Business Media.

[Li et al. 2014] Li, C., Dou, L., and Yang, Z. (2014). A metamodeling level transformation from UML sequence diagrams to Coq. In *Proceedings of the 15th Italian Conference on Theoretical Computer Science*, pages 147–157.

[Ludewig 2003] Ludewig, J. (2003). Models in software engineering–an introduction. *Software and Systems Modeling*, 2(1):5–14.

[Miller and Mukerji 2003] Miller, J. and Mukerji, J. (2003). Mda guide version 1.0.1. Disponível em http://www.omg.org/cgi-bin/doc?omg/03-06-01. Acessado em 01/05/2016.

[Muller et al. 2005] Muller, P.-A., Fleurey, F., Vojtisek, D., Drey, Z., Pollet, D., Fondement, F., Studer, P., and Jézéquel, J.-M. (2005). On Executable Meta-Languages applied to Model Transformations. In *Model Transformations In Practice Workshop*, pages 1–49, Montego Bay, Jamaica.

[Mussbacher et al. 2014] Mussbacher, G., Amyot, D., Breu, R., Bruel, J.-M., Cheng, B., Collet, P., Combemale, B., France, R., Heldal, R., Hill, J., Kienzle, J., Sch"ottle, M., Steimann, F., Stikkolorum, D., and Whittle, J. (2014). The Relevance of Model-Driven Engineering Thirty Years from Now. In *17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014)*, volume 8767 of *Model-Driven Engineering Languages and Systems*, pages 183–200, Valencia, Spain. Springer.

[Neto and de Oliveira 2013] Neto, V. V. G. and de Oliveira, J. L. (2013). Evolução de uma arquitetura de framework de aplicação para sistemas de informação com desenvolvimento dirigido por modelos. In *Anais do IX Simpósio Brasileiro de Sistemas de Informação (SBSI 2013)*, volume 1, pages 320–331.

[OMG 2016] OMG (2016). Mda - the architecture of choice for a changing world. Disponível em http://www.omg.org/mda/. Acessado em 01/05/2016.

[Poruban et al. 2014] Poruban, Bacikova, Chodarev, and Nosal (2014). Pragmatic model-driven software development from the viewpoint of a programmer: Teaching experience. In *Conference on Computer Science and Information Systems*, page 1647–1656. IEEE.

[Rumbaugh et al. 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. E., et al. (1991). *Object-oriented modeling and design*, volume 199. Prentice-hall Englewood Cliffs.

[Schmidt et al. 2014] Schmidt, A., Kimmig, D., Bittner, K., and Dickerhof, M. (2014). Teaching model-driven software development: Revealing the "great miracle"of code generation to students. In *Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148*, ACE '14, pages 97–104. Australian Computer Society, Inc.

[Schmidt 2006] Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):0025–31.

[Selic 2003] Selic, B. (2003). The pragmatics of model-driven development. *IEEE software*, (5):19–25.

[Shen et al. 2008] Shen, H., Virani, A., and Niu, J. (2008). Formalize uml 2 sequence diagrams. In *11th IEEE High Assurance Systems Engineering Symposium (HASE 2008)*, pages 437–440.

[Van Deursen and Klint 2002] Van Deursen, A. and Klint, P. (2002). Domain-specific language design requires feature descriptions. *CIT. Journal of computing and information technology*, 10(1):1–17.

[Whittle et al. 2014] Whittle, J., Hutchinson, J., and Rouncefield, M. (2014). The state of practice in model-driven engineering. *Software, IEEE*, 31(3):79–85.

[Whittle et al. 2013] Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., and Heldal, R. (2013). Industrial adoption of model-driven engineering: Are the tools really the problem? In *Model-Driven Engineering Languages and Systems*, pages 1–17. Springer.

[Whittle et al. 2015] Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., and Heldal, R. (2015). A taxonomy of tool-related issues affecting the adoption of model-driven engineering. *Software & Systems Modeling*, pages 1–19.

[Ziadi et al. 2011] Ziadi, T., Da Silva, M., Hillah, L., and Ziane, M. (2011). A fully dynamic approach to the reverse engineering of uml sequence diagrams. In *Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on*, pages 107–116.

A. Anexo - Questionário de Avaliação da Disciplina

Parte 1 - Questões Gerais

Q1. Você já tinha experiência com modelagem antes da disciplina? *
( ) Sim ( ) Não

Q2. Você trabalha atualmente com desenvolvimento de software? *
( ) Sim ( ) Não

Q3. A disciplina de MDE melhorou sua compreensão em relação à importância da modelagem de software? *
( ) Sim ( ) Não

Q4. A disciplina de MDE melhorou sua forma de usar ou fazer modelagem? *
( ) Sim ( ) Não

Q5. Você usa ou pretende usar conceitos, ferramentas ou linguagens MDE na sua pesquisa ou no seu trabalho? *
( ) Sim ( ) Não

Q6. Você acha que a disciplina impactou ou influenciou na sua formação profissional/acadêmica? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Parte 2 – Metodologia

Q7. Você aprova a metodologia utilizada na disciplina (primeira parte teórica e segunda prática)? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q8. A metodologia da disciplina ajudou a compreender os conceitos de MDE? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q9. Você acha que os artigos lidos e discutidos na parte teórica da disciplina ajudaram a entender os conceitos e a aplicação de MDE? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q10. Você acha que as ferramentas utilizadas na parte prática da disciplina foram adequadas para o aprendizado do assunto, mesmo enfrentando dificuldades em utilizá-las? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q11. Os trabalhos teóricos (resenhas, resumos, e apresentações) foram motivantes para você? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q12. Os trabalhos práticos (implementações) foram motivantes para você? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Parte 3 – Ferramentas

Q13. Eclipse Modeling Framework (EMF) foi fácil de usar? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q14. Eugenia foi fácil de usar? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q15. O Xtext foi fácil de usar? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q16. O ATL foi fácil de usar? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q17. O Acceleo foi fácil de usar? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Q18. A documentação existente sobre as ferramentas (oficial e não oficial) ajudou a utilizá-las? *
( ) Discordo plenamente ( ) Discordo ( ) Neutro
( ) Concordo ( ) Concordo plenamente

Parte 4 – Sugestões e críticas

Q19. Você tem alguma sugestão ou crítica a fazer para a melhora da disciplina?