CoMDD: uma abordagem colaborativa para auxiliar o desenvolvimento orientado a modelos

David Fernandes Neto

CoMDD: uma abordagem colaborativa para auxiliar o desenvolvimento orientado a modelos

David Fernandes Neto

Orientadora: ***Profa. Dra. Renata Pontin de Mattos Fortes***

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, para o exame de Qualificação, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Julho/2011

Resumo

O desenvolvimento orientado a modelos (MDD) é uma abordagem que tem ganhado cada vez mais espaço na indústria e na academia, trazendo grandes benefícios, como o aumento de produtividade, por exemplo. Contudo, ferramentas que auxiliem o desenvolvimento orientado a modelos de maneira colaborativa são praticamente inexistentes. Isso faz com que o MDD seja limitado ao desenvolvimento isolado e *stand alone*, dificultando que sistemas complexos e abrangentes sejam desenvolvidos com esse paradigma. Assim, este trabalho apresenta uma abordagem colaborativa orientada a modelos (CoMDD), a qual pretende mostrar que a colaboração pode trazer benefícios ao MDD e permitir que desenvolvedores de MDD possam trabalhar colaborativamente, aumentando ainda mais a produtividade, possibilitando a troca de experiências e conhecimentos, e permitindo o desenvolvimento de sistemas complexos. Serão realizados dois estudos de caso: um no domínio de tecnologia de informação (TI) e outro no domínio de sistemas embarcados (SE).

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	*Application Programming Interface*
CDE	*Collaborative Development Environment*
CIM	*Computation Independent Model*
CMS	*Content Management System*
CoMDD	*Colaborative Model-Driven Development*
CSCW	*Computer Supported Cooperative Work*
DDS	Desenvolvimento Distribuído de Software
DR	*Design rationale*
DSL	*Domain Specific Language*
DTE	Diagrama de Transição de Estados
EMF	*Eclipse Modeling Framework*
ER	Entidade Relacionamento
GMF	*Graphical Modeling Framework*
IDE	*Integrated Development Environmen*
I/O	*Input/Output*
JET	*Java Emitter Templates*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*
MDE	*Model-Driven Engineering*
MDSD	*Model-Driven Software Development*
MOF	*Meta-Object Facility*
OMG	*Object Management Group*
PIM	*Plataform Independent Model*
PSM	*Plataform Specif Model*
RMA	Robô Móvel Autônomo

SE	*Sistemas Embarcados*
tMDD	*traditional Model-Driven Development*
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*

CAPÍTULO

1

Introdução

O Desenvolvimento Orientado a Modelos (MDD - *Model-Driven Development*) é uma abordagem que se concentra na concepção e transformações de modelos. Os modelos são o foco central, que por meio de transformações podem gerar código-fonte (**?**). Seu objetivo é reduzir a distância semântica que existe entre o domínio do problema e o domínio da implementação/solução, utilizando modelos mais abstratos que protegem os desenvolvedores de software das complexidades inerentes à plataforma de implementação (**?**).

Um dos principais benefícios desta abordagem é a reutilização. Modelos condensam conhecimento produzido durante atividades de análise, projeto e implementação, de uma forma (idealmente) independente da plataforma de implementação. Assim, reutilizando-se um modelo, este conhecimento pode ser mais facilmente reaproveitado em outros contextos (**?**).

A reutilização (ou reúso) de software é uma atividade altamente colaborativa em sua essência, pois normalmente o reutilizador de um artefato não é a mesma pessoa (e possivelmente não precisa pertencer à mesma equipe) que o produtor do artefato (**?**).

Modelos são bons artefatos para comunicação e interação entre os stakeholders e para troca de experiências. Isso porque modelos geralmente são mais fáceis de entender do que código-fonte, de modo que não só os engenheiros de software participam da modelagem, mas também os especialistas de domínio e até os stakeholders podem participar. Assim, um modelo pode ser construído por diferentes pessoas ao mesmo tempo e distribuídas.

O desenvolvimento de software é um processo intensamente colaborativo onde o sucesso depende da habilidade de criar, compartilhar e integrar informação (**?**). Assim, acredita-se que uma abordagem que junte os benefícios da colaboração ao MDD, possa trazer essencialmente o aumento de produtividade.

Embora o MDD se mostre uma abordagem promissora, emergindo ao longo das últimas décadas (**?**) e apresentado ganhos de produtividade no desenvolvimento de sistemas (**?**) as soluções existentes ainda são incipientes em termos de ferramentas e processos necessários para auxiliar os desenvolvedores em seus trabalhos (**?**). Isso se torna mais evidente quando observamos o as ferramentas e processos que auxiliam o MDD e às que auxiliam o desenvolvimento tradicional (sem ser MDD).

No desenvolvimento tradicional, ferramentas permitem à equipe de desenvolvedores trabalhar e coordenar atividades, combinar de código e dividir de tarefas, entre outros. Contudo, pela falta das mesmas ferramentas de apoio no contexto de MDD e pela falta do uso de modelos para criar conhecimentos compartilhados, as equipes de desenvolvimento trabalham de maneira mais restrita (**?**), (**?**). Assim, o MDD que é uma abordagem que promove alta troca de experiências não é usada em colaboração e nem de maneira distribuída, ao contrário do desenvolvimento tradicional. Parte deste trabalho está na observação do desenvolvimento tradicional e como pode-se agregar suas vantagens ao MDD.

Outra tendência promissora de utilização de MDD é em sistemas embarcados (**?**). Gerenciar o rápido crescimento da complexidade do desenvolvimento de sistemas embarcados é uma das tarefas mais desafiadoras para aumentar a qualidade do produto, diminuir o tempo até a chegada do produto ao mercado e reduzir os custos de desenvolvimento. O MDD se apresenta como uma das abordagens que têm emergido nas últimas décadas e encontrada grande aceitação na comunidade de engenharia de software.

Um software embarcado raramente é desenvolvido como um produto singular, mas como um elemento de um sistema embarcado, que é composto de outros elementos (circuitos elétricos, eletrônicos, peças mecânicas, etc). Assim, o desenvolvimento de sistemas embarcados torna-se mais colaborativo que o desenvolvimento de sistemas de informação, em geral, pois para o desenvolvimento de um sistema embarcado é necessário não somente os desenvolvedores do software, como também a intervenção dos demais profissionais envolvidos, como: os engenheiros elétricos, eletrônicos, mecânicos, da computação, dentre outros(**?**).

Ainda no contexto de suporte a trabalhos colaborativos, percebe-se que atualmente, há um movimento de migração de sistemas de desktop para aplicações web. Observando o histórico desses sistemas, acredita-se que num futuro não muito distante, ter-se-á o desenvolvimento de softwares colaborativos via web com muito mais vigor. Por isso o CoMDD é apresentado como um serviço.

A colaboração é uma atividade que pode trazer benefícios para o MDD e conseqüentemente para a equipe e projeto, como se deseja evidencias neste trabalho. Ela pode ser imprescindível em sistemas extensos e complexos.

As vantagens do MDD, o seu potencial para o desenvolvimento distribuído e a carência de ferramentas/processo para o MDD nos levou a identificarmos características do desenvolvimento tradicional com o intuito de contribuir ao MDD e permitir que ele seja aplicado de maneira distribuída.

Assim, apresentamos uma abordagem colaborativa de desenvolvimento orientada a modelos (Colaborative Model Driven Development - CoMDD) que possibilita o desenvolvimento distribuído de software a fim de que desenvolvedores tradicionais possam adotar o MDD e receber seus benefícios e de que os atuais desenvolvedores que usam MDD ganhem com os benefícios da colaboração adotando o CoMDD.

O CoMDD será definido como um conjunto de características e regras (práticas) de desenvolvimento necessários para que haja um trabalho colaborativo orientado a modelos.

Estrutura da monografia

CAPÍTULO

2

MDD

Re, veja se este capítulo está bom ou se precisamos colocar mais coisas ou até mesmo tirar, já que não estamos usando o Eclipse. Será que não seria melhor se falássemos apenas do conceito de DSLs? Ou compensa falar de outras abordagens de MDD como a MDA e o Eclipse?

O Desenvolvimento Orientado a Modelos é uma abordagem que tem amadurecido e apresentada ser uma boa opção para elevar o nível de produtividade, interoperabilidade, portabilidade, reúso, comunicação, entre outros, de um projeto de software. Neste capítulo será apresentada a definição de *Model-Driven Development*, seus principais benefícios e elementos, o conceito de metamodelagem e as duas das mais importantes abordagens usadas na indústria e academia: a abordagem do *Object Management Group* (OMG) e a abordagem da IBM.

2.1 Definição e Vantagens do MDD

O Desenvolvimento Orientado a Modelos (*Model-Driven Development* - MDD) é uma abordagem da Engenharia de Software que consiste na aplicação de modelos para elevar o nível de abstração, na qual os desenvolvedores criam e evoluem o software. Sua intenção é tanto simplificar (tornar mais fácil) quanto formalizar (padronizando, de forma que a automação seja possível) as várias atividades e tarefas que formam o ciclo de vida do software (**?**); ou numa definição mais singela: o MDD é a simples noção de construir um modelo de um sistema e depois transformá-lo em algo real (**?**).

Uma outra definição semelhante é a de que o MDD é uma abordagem de desenvolvimento, integração e interoperabilidade de sistemas de tecnologia da informação. Ela se refere ao uso

sistemático de modelos e de transformações de modelos como artefatos primários, durante todo o ciclo de vida do software. O MDD eleva o nível de abstração em que os desenvolvedores produzem softwares, simplificando e formalizando as diversas atividades e tarefas que compõem o processo de desenvolvimento de software (**?**), expressando o que um computador deve fazer por meio de modelos, que escondem detalhes de implementação (**?**).

O que caracteriza o MDD são os modelos como foco primário do desenvolvimento de software, ao invés das linguagens de programação. Os modelos são usados para descrever vários aspectos do software e para automatizar a geração de código. A principal vantagem disto é poder expressar modelos usando conceitos menos vinculados a detalhes de implementação, além do fato de modelos serem mais próximos do domínio do problema. Isto torna os modelos mais fáceis de se especificar, entender e manter, do que abordagens que não usam modelos. E em alguns casos, ainda é possível os especialistas do domínio produzirem os sistemas ao invés dos especialistas da tecnologia de implementação (**??**)

Portanto, o modelo no MDD é um artefato primordial para o desenvolvimento e dessa forma se contrapõe ao desenvolvimento convencional de software, no qual os modelos são utilizados para representar o problema e servem *apenas* para auxiliar o desenvolvimento de código pelo programador. No MDD, a idéia é que esses modelos possam gerar o código (e outros artefatos) e não fiquem limitados apenas como um auxílio ao desenvolvedor. A Figura 2.1 ilustra os modelos como auxílio e a Figura 2.2 ilustra os modelos como responsáveis por gerar o código.

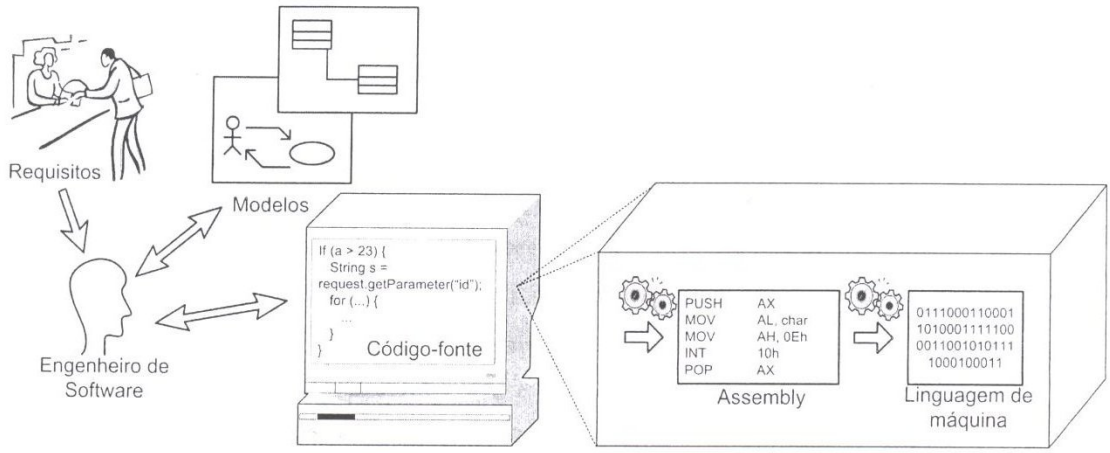

Figura 2.1: Processo convencional de desenvolvimento de software (**?**)

.

Segundo (**??**) o MDD apresenta grandes benefícios como:

- Produtividade: fatores como redução de atividades repetitivas e manuais e o aumento da possibilidade de reúso, podem contribuir para o aumento da produtividade no processo de desenvolvimento;
- Portabilidade e Interoperabilidade: como o modelo é independente de plataforma, um mesmo modelo pode ser transformado em código para diferentes plataformas;

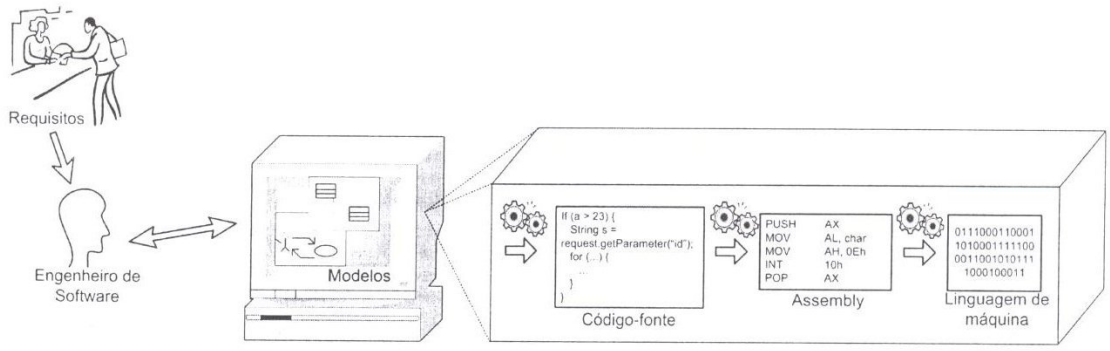

Figura 2.2: Processo de desenvolvimento de software orientado a modelos (**?**)

.

- Corretude: o MDD evita que os desenvolvedores exerçam atividades repetitivas e manuais para gerar código; dessa maneira, evita-se também alguns erros como: geradores de código não introduzem erros acidentais, como o de digitação, por exemplo;
- Manutenção: alterações no código relativas à manutenção podem requerer o mesmo esforço produzido durante o desenvolvimento;
- Documentação: modelos são o artefato principal do processo de desenvolvimento e por isso não se desatualizam, pois o código é gerado a partir deles e com isso a documentação se mantém também sempre atualizada;
- Comunicação: modelos são mais fáceis de entender do que código-fonte, assim isso facilita a comunicação entre os desenvolvedores, os *stakeholders* e demais envolvidos.

Falar das desvantagens do MDD.

Analisando tais definições de MDD e suas vantagens apresentadas, pode-se concluir que MDD é um conceito bem definido, pois diferentes autores apresentam as mesmas idéias de que MDD é uma elevação no nível de abstração do desenvolvimento de software, no qual o modelo passa de um artefato auxiliar para uma peça fundamental no processo de desenvolvimento. Em virtude dessa elevação na abstração, muitos problemas relativos a portabilidade, interoperabilidade, corretude, manutenção e documentação, são reduzidos nesta abordagem. Contudo, é fato que mesmo o MDD com tantos benefícios, inclusive o aumento de produtividade, ele ainda está longe de ser a solução para todos os problemas relativos ao desenvolvimento de software.

2.2 Fundamentos do MDD

É importante conhecer alguns conceitos para entender o processo de desenvolvimento orientado a modelos, uns específicos do MDD - como os principais elementos que compõem o MDD e outros abrangentes, como o conceito de DSL e de metamodelagem.

Principais elementos do MDD

Lucrédio (**?**) define os principais elementos do MDD como sendo:

- *Ferramenta de modelagem*: através dela o engenheiro de software produz modelos que descrevem conceitos do domínio, segundo uma linguagem específica de domínio (*Domain Specific Language* - DSL[1]). Para que os modelos sejam capazes de gerarem código de maneira automática, eles devem ser semanticamente completos e corretos;
- *Ferramenta para definir as transformações*: os modelos serão transformados em outros modelos ou em código fonte; portanto é necessário uma ferramenta para definir transformações na qual o engenheiro de software constrói regras de mapeamento de modelo para modelo ou de modelo para código;
- *Mecanismo que aplique as transformações*: com o modelo e as transformações definidas, resta um mecanismo que aplique as transformações nos modelos. É importante que além de aplicar as transformações, esse mecanismo mantenha informações de rastreabilidade, possibilitando saber a origem de cada elemento gerado.

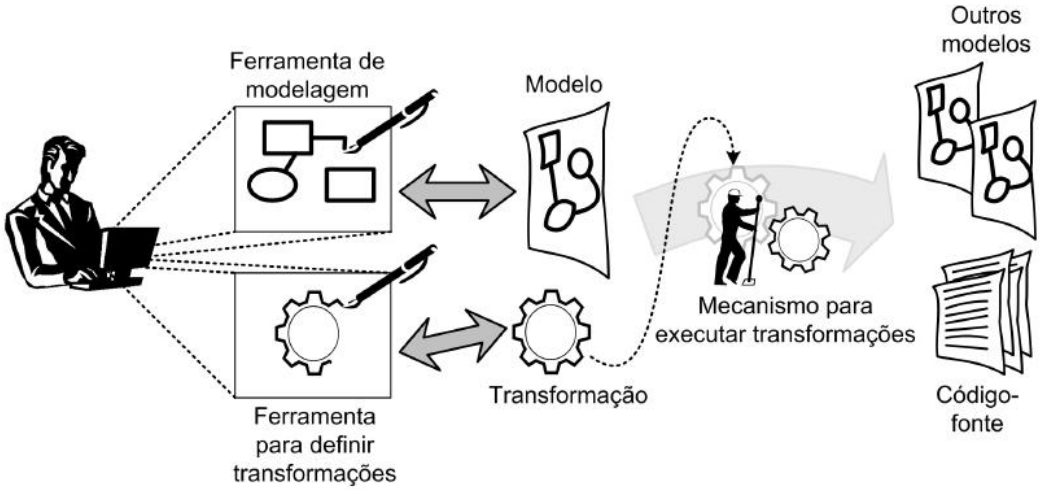

Figura 2.3: Principais elementos do MDD (**?**).

A Figura 2.3 ilustra um engenheiro de software[2] desenvolvendo um modelo e uma transformação, a partir de uma ferramenta de modelagem e de uma ferramenta para definir transformações, respectivamente. Esses dois artefatos, por sua vez, compõem a entrada de um mecanismo de transformação, o qual gera como saída outros modelos ou códigos-fonte.

[1] As DLSs serão definidas ainda nesta Seção.

[2] Embora na Figura só tenha uma pessoa fazendo a modelagem e definindo transformações, pode-se ter uma pessoa para cada atividade

Metamodelagem

A metamodelagem é um dos principais aspectos do MDD. É necessário criar um metamodelo do conhecimento para o MDD lidar com alguns desafios, como: a construção de uma DSL; validar modelos; realizar transformações de modelo; gerar código e integrar ferramentas de modelagem a um domínio (**?**).

Um metamodelo descreve a estrutura de um modelo. De maneira abstrata, o metamodelo define os construtores de uma linguagem de modelagem e seus relacionamentos, bem como as constantes e regras de modelagem; contudo ele não descreve a sintaxe concreta da linguagem. Metamodelos e modelos têm um relacionamento de classe e instância, ou seja, cada modelo é uma instância de um metamodelo. A relação *meta* deve ser vista em relação a um modelo. Embora a definição absoluta de metamodelagem possa não fazer sentido na teoria, na prática ela é bastante útil (**?**).

Mais detalhes sobre metamodelagem serão explicados na Seção 2.3.1.

DSL e DSML

Uma DSL (*Domain-Specific Language*) é uma linguagem de programação ou uma linguagem de especificação executável, que oferece, através de notações e abstrações, poder expressivo focado em um problema de um domínio particular, e geralmente restrito a este domínio. Embora seja uma linguagem específica e não forneça uma solução geral para muitas áreas, ela provê uma solução muito melhor para um domínio particular (**?**) quando comparada com uma linguagem de propósito geral, como a UML, por exemplo.

O termo DSML (*Domain-Specific Modeling Language*) pode ser considerado como um sinônimo de DSL e para Heering (2007/8) DSML é uma DSL no contexto de MDD; com uma sintaxe gráfica, geralmente.

Aplicando uma DSML no lugar de uma linguagem de modelagem genérica, como a UML, permite trazer o especialista para mais perto do problema. DSMLs podem tornar o desenvolvimento de software, em domínios específicos, mais eficiente; uma vez que as DSMLs reduzem o *gap* entre os especialistas e a implementação do software (**?**).

As DSMLs permitem aos desenvolvedores usarem conceitos específicos do domínio, por isso fornecem uma linguagem de modelagem ainda mais intuitiva e integram ainda mais o *know-how* dos desenvolvedores de software e as otimizações específicas de aplicações dentro dos geradores de código.

2.3 Principais Abordagens de MDD

Na literatura existem vários sinônimos para MDD, como *Model-Driven Engineering* - MDE ou *Model-Driven Software Development* - MDSD; entretanto é comum confundir MDD com *Model-Driven Architecture* - MDA. A MDA é uma abordagem de MDD assim como existem outras. Neste

trabalho serão explicados os conceitos de duas das abordagens mais utilizadas de MDD: a MDA e a abordagem Eclipse.

2.3.1 MDA

O *Object Management Group*[3] (OMG) foi formado com o intuito de ser uma organização de padrões para ajudar a reduzir complexidade e custos de desenvolvimento, além de acelerar a introdução de novas aplicações de software. A intenção do OMG é alcançar este objetivo através da introdução do *framework* arquitetural *Model Driven Architecture*, com apoio de especificações detalhadas (UML, MOF, XMI). É graças à essas especificações que a indústria caminha em direção à interoperabilidade, reusabilidade e portabilidade de componentes de software e de modelos, baseados em padrões (**?**).

Os três modelos: CIM, PIM e PSM

A MDA define três modelos que são produzidos ao longo do ciclo de vida do desenvolvimento: o *Computation Independent Model*(CIM), o *Plataform Independent Model*(PIM) e o *Plataform Specif Model* (PSM) e são conceituados como (**?**):

- CIM: é também conhecido como modelo de domínio, onde um vocabulário familiar dos profissionais do domínio é usado para fazer as especificações. Dessa maneira, assume-se que o primeiro usuário do CIM, o profissional do domínio, não é conhecedor dos modelos ou artefatos usados para realizar as funcionalidades para as quais os requisitos são articulados no CIM. O CIM tem o importante papel de fazer a ligação entre os *experts* do domínio (com seus requisitos) e os *experts* de *design*, que constróem os artefatos. Não faz parte de seu escopo mostrar detalhes da estrutura do sistema.
- PIM: é o modelo que apresenta um determinado grau de independência de plataforma, de modo a ser adequado para o uso por diferentes plataformas de tipos semelhantes. Um PIM foca no funcionamento do sistema, pois esconde os detalhes necessários da plataforma específica, mostrando a parte da especificação que não muda de plataforma para outra.
- PSM: é o modelo que combina as especificações do PIM com os detalhes específicos da plataforma.

A Figura 2.4 apresenta o ciclo de vida de software adotando-se a MDA, onde a partir dos requisitos levantados, um CIM é gerado. Esse modelo será processado na fase de Análise e gerado um PSM, que por sua vez será processado na fase de projeto e transformado em um PSM. Depois o PSM será transformado em código-fonte, neste caso, na fase de codificação. Nesta Figura, também é importante observar que o processo MDA, depois da fase de implantação, volta para a fase inicial de requisitos, enquanto que o processo tradicional volta apenas na fase de codificação.

[3] http://www.omg.org/

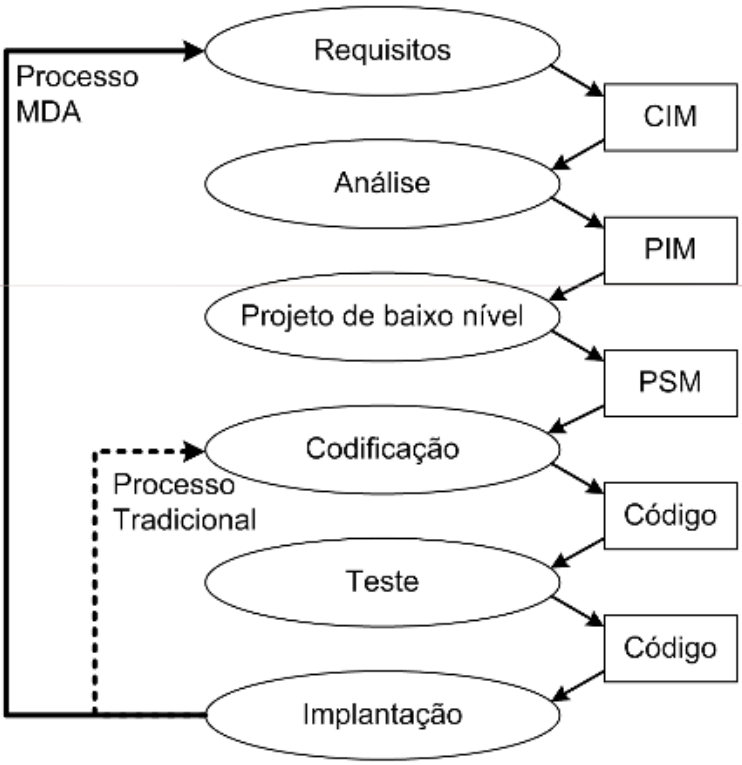

Figura 2.4: Ciclo de vida da MDA

As transformações entre modelos são utilizadas para converter um modelo em outro, sucessivamente, até o código-fonte. A transformação entre CIM e PIM é menos passível de automação, pois envolve mais decisões e maiores possibilidades de interpretação dos requisitos. Já transformação entre PIM e PSM é realizada a partir da combinação entre o PIM e outra informação (Figura 2.5). Quanto menos informação for preciso, mais automatizado é o processo. Já a transformação entre PSM e código-fonte é mais passível de automação, uma vez que o PSM está intimamente ligado com a plataforma de implementação (**?**).

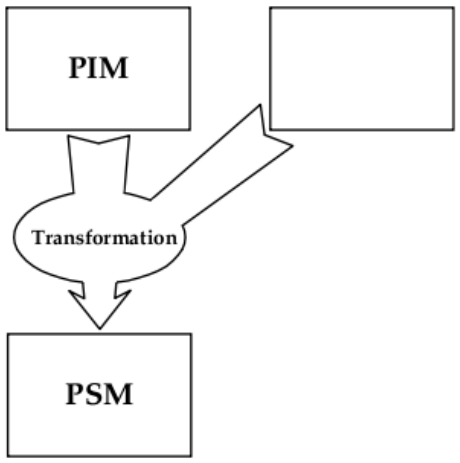

Figura 2.5: Transformação de Modelos (**?**)

Essa estrutura de modelos é formada para que um mesmo PIM possa gerar vários PSMs (Figura 2.6), promovendo maior portabilidade, uma vez que pode-se ter PSMs para várias plataformas a partir de um mesmo PIM.

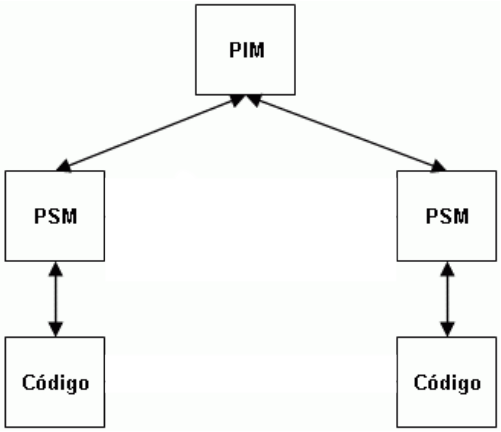

Figura 2.6: Múltiplos PSMs a partir de um único PIM

Especificações usadas na MDA

Na MDA, um modelo é uma representação de parte de uma função, uma estrutura e/ou um comportamento de um sistema (de maneira geral e não somente sistemas de software). Uma especificação é dita formal quando é baseada em uma linguagem que tenha uma estrutura (sintaxe) e significados (semântica) bem definidos. A sintaxe pode ser gráfica ou textual. A semântica pode ser definida formalmente em termos de coisas observadas no mundo ou por traduções de construtores de linguagens de alto nível em construtores que tenham um significado bem definido (**?**).

Na MDA, uma especificação que não é formal não é um modelo. Por isso, um diagrama com caixas, linhas e setas que não tenha por trás uma definição do significado de uma caixa, de uma linha e de uma seta; não é um modelo. É apenas um diagrama informal. Um modelo MDA deve estar combinado de maneira inequívoca com a definição da sintaxe e da semântica de uma linguagem de modelagem, como é fornecido pelo MOF (**?**). O MOF é a linguagem de meta-metamodelagem da MDA.

A arquitetura de modelagem da MDA é dividida em quatro camadas ou níveis (Figura 2.7). O nível M0 são os dados, as instâncias. O nível M1 é composto pelos modelos das instâncias, ou seja, pelos modelos (ou metadados) da camada M0. A camada M2 é composta pelos metamodelos, ou seja, são modelos dos modelos da camada M1. A UML é um exemplo de metamodelo ou de linguagem de modelagem, a qual é usada na MDA. O nível M3 é usado para definir os metamodelos do nível M2, ou seja, um meta-metamodelo que define linguagens de modelagem. Um exemplo de meta-metamodelo é o MOF (*Meta-Object Facility*, padrão da MDA usado para modelar). Normalmente o meta-metamodelo é uma instância de si mesmo e por isso não existe um quinto nível (**?**).

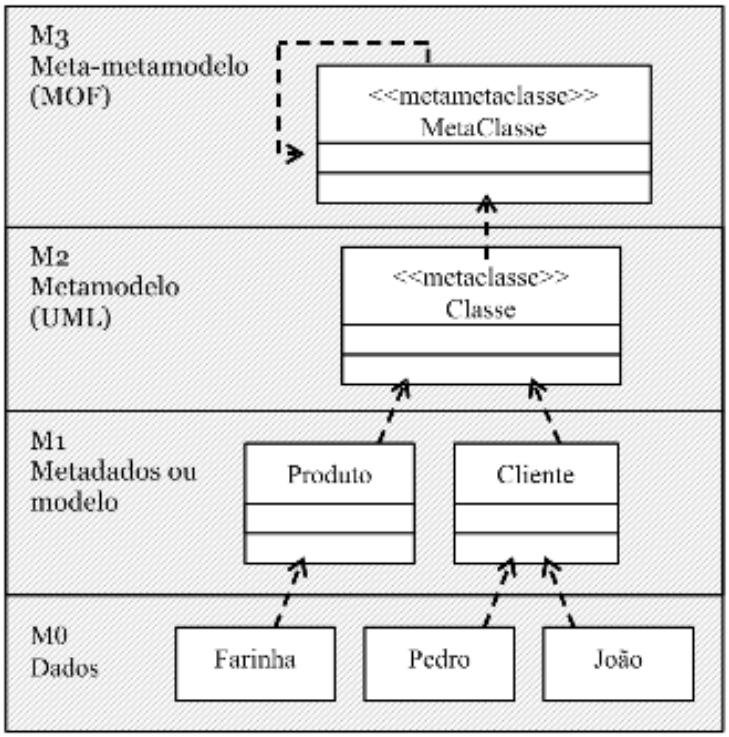

Figura 2.7: Arquitetura da Metamodelagem

O XMI (*XML Metadata Interchange*) é uma especificação do OMG[4] como sendo um modelo orientado a XML para fazer o intercâmbio ("interchange") de UML e outros modelos baseados em MOF, através da padronização de formatos XML, DTDs e schemas. Ao fazer isso, o XMI define um mapeamento de UML/MOF para XML. Padrões baseados em XMI são usados para integrar ferramentas, repositórios e aplicações. O XMI provê regras pelas quais um schema pode ser gerado por qualquer metamodelo baseado em MOF, transmissível por XMI (**??**).

2.3.2 Abordagem Eclipse

O projeto Eclipse foi originalmente criado em novembro de 2001 pela IBM (*International Business Machines*) (**?**) e comporta diversos subprojetos. Um deles é o *Eclipse Modeling Project*, cujo foco é a evolução e promoção de tecnologias baseadas em modelos, fornecendo um conjunto unificado de *frameworks* de modelagem, ferramentas e implementações de normas (**?**).

Eclipse Modeling Framework

O Eclipse Modeling Framework (EMF) é um framework Java/XML de modelagem para geração de código, ferramentas e outras aplicações baseadas em modelos. O EMF ajuda o desenvolvedor a transformar modelos rapidamente em eficientes, corretos e facilmente customizáveis códigos Java. O EMF destina-se a proporcionar os benefícios da modelagem formal, mas com um

[4]O MOF e a UML também são especificações do OMG, assim como existem outras especificações

custo de entrada baixo. Os modelos podem ser criados usando anotações Java, documentos XML ou através de ferramentas de modelagem como o *Rational Rose*[5], e então importados para o EMF (**??**).

Assim, a partir de uma especificação de um modelo em XMI, o EMF fornece ferramentas e suporte em tempo de execução, para produzir um conjunto de classes Java para o modelo. Uma vez especificado um modelo EMF, o gerador EMF pode criar um conjunto de classes Java correspondentes ao modelo e caso haja mudanças no código Java, elas podem ser usadas para atualizar o modelo. Além da geração de código o EMF fornece a habilidade de salvar objetos em documentos XML para o intercâmbio com outras ferramentas e aplicações (**??**).

Fazendo um paralelo entre as duas abordagens, na abordagem Eclipse, a linguagem de modelagem é o EMF, enquanto que na MDA é a UML. A linguagem de meta-metamodelagem na abordagem Eclipse é o *Ecore* e na MDA é o MOF. A abordagem Eclipse também usa a especificação XMI, da OMG, para fazer o intercâmbio de modelos.

JET

Java Emitter Templates - JET é tipicamente usado na implementação de um gerador de código. Um gerador de código é um importante componente do MDD.

O objetivo do MDD é descrever um sistema de software usando modelos abstratos (como modelos em EMF ou UML) e então refinar e transformar esses modelos em código. O verdadeiro poder do MDD vem da automatização desse processo. Tais transformações aceleram o processo do MDD e resultam em um código de melhor qualidade (**?**).

Uma das vantagens das transformações é que elas podem capturar "boas práticas" de especialistas e assim assegurar que o projeto empregue essas práticas. Contudo, transformações não são sempre perfeitas. Boas práticas dependem do contexto - o que é ótimo em um contexto pode não ser em outro. As transformações podem resolver esse problema incluindo algum mecanismo para o usuário final modificar o código gerado. Isto é freqüentemente feito usando "templates" para criar artefatos e permitir que usuários substituam suas próprias implementações por esses *templates* se for necessário. Este é papel do JET (**?**).

O JET também permite que qualquer comando Java possa ser utilizado, além de marcações (*tags*) que implementam comandos condicionais de laços, por exemplo. O JET pode ser inserido em arquivos XML ou em modelos EMF, sendo possível de utilizá-lo como um gerador de código para uma DSL (**?**).

GMF

O *Graphical Modeling Framework* (GMF) é mais um projeto do Eclipse que provê um componente gerador e uma infraestrutura em tempo de execução para o desenvolvimento de editores

[5]http://www.developers.net/ibmshowcase/view/1423

gráficos baseados em EMF (**?**). Ele possibilita a definição de um ambiente de modelagem para uma linguagem visual de um determinado domínio.

2.4 Considerações Finais

Neste capítulo foram apresentados os conceitos relativos a MDD e duas de suas principais abordagens, ressaltando que existem outras, como a abordagem da Sun Microsystems[6] [7], a da *Vanderbilt University*[8] e a da Microsoft[9], entre outras.

Neste projeto de mestrado as duas abordagens apresentadas: a Eclipse e a MDA serão os referenciais para os estudos necessários para que o ambiente CoMDD seja desenvolvido.

[6] http://java.sun.com/products/jmi/
[7] http://mdr.netbeans.org/
[8] http://www.isis.vanderbilt.edu/projects/gme/
[9] http://msdn.microsoft.com/en-us/library/aa480032.aspx

CAPÍTULO

3

Colaboração

Neste trabalho pretende-se utilizar os benefícios da colaboração no desenvolvimento orientado a modelos, a fim de alcançar melhores resultados quanto a produtividade, troca de experiências, entre outros. Para isso é importante entender a complexidade dos sistemas colaborativos e como usá-los para atingir os objetivos do trabalho. Assim, este capítulo apresenta definições relacionadas ao tema de colaboração em sistemas computacionais, como: CSCW, *groupware* e CDEs. São também apresentados os requisitos que um ambiente colaborativo deve ter. Por fim, é realizado um levantamento dos tipos de sistemas gerenciadores de conteúdo (*Content Management System* - CMS) que este trabalho deverá usar para projetar e desenvolver a proposta da abordagem do CoMDD.

3.1 CSCW, Groupware, CDEs

Colaboração significa duas ou mais pessoas trabalhando juntas para compartilhar e trocar dados, informações, conhecimento, entre outros (**?**). A colaboração é uma atividade que tem sido evidenciada cada vez mais com o aumento da complexidade do desenvolvimento de sistemas, pois para se desenvolver um sistema robusto e complexo, uma só pessoa dificilmente é o suficiente. Assim, o desenvolvimento e a manutenção de softwares complexos requerem a colaboração de diversos desenvolvedores (**?**).

O termo colaboração é muito amplo e é estudado por especialistas de diversas áreas distintas, como: comunicação, psicologia, computação e outras. Surge então o termo *Computer Supported Cooperative Work*. CSCW estuda como as pessoas usam a tecnologia, em relação ao hardware e ao software, em tempo e espaço compartilhados (**?** *apud* **?**). Segundo *Garrido*, CSCW estuda e

analisa os mecanismos para a comunicação e colaboração efetivas entre as pessoas e os sistemas que os apóiam.

Ao longo dos anos, vários sistemas foram concebidos para ajudar as pessoas a trabalharem de forma colaborativa e conjunta em um mesmo projeto. Estes sistemas são conhecidos como *groupware*, ou softwares colaborativos, e podem ser definidos como softwares que auxiliam grupos de pessoas envolvidas em uma tarefa (ou objetivo) comum e que fornecem uma interface de um ambiente compartilhado (**?**). São exemplos de *groupware* o e-mail, programas de *chat*, as Wikis, blogs, entre outros.

Portanto, o termo CSCW é usado para definir a área da pesquisa que foca no estudo de ferramentas e técnicas para *grouware*, bem como os seus impactos sociais, organizacionais e psicológicos. *Groupware* é considerada a tecnologia de apoio, seja software, serviços e/ou técnicas, as quais possibilitam as pessoas trabalharem em grupos, simultaneamente no mesmo projeto ou não (**?**).

Considerando que um sistema de usuário único ("single user system") foca no trabalho individual, o *groupware* foca no grupo. Mas quais as vantagens que um *groupware* propicia em comparação com um *"single user system"* e que motivariam seu uso? Quando se trabalha em um projeto no qual a comunicação é essencial entre os colaboradores, o *groupware* facilita a comunicação mais rápida e clara, permitindo a comunicação onde ela não seria possível. Ele visa possibilitar múltiplas perspectivas, especialidades e assistências na solução de problemas pelo grupo. Sua intenção é economizar tempo e custo coordenando o trabalho do grupo. Por outro lado, o *groupware* é mais complexo de se desenvolver e manter do que um sistema de usuário único (**?**).

Um outro conjunto de ferramentas que permitem o trabalho em grupo são os chamados *Collaborative Development Environment* (CDE).

Um CDE é um espaço visual onde todos os *stakeholders* do projeto, mesmo distribuídos em tempo e/ou espaço, podem negociar, fazer *brainstorm*, discutir, compartilhar conhecimento e trabalharem em conjunto para realizar alguma tarefa. Comparando os CDEs com as IDEs (*Integrated Development Environmen*), as IDEs são essenciamente centradas no desenvolvedor, enquanto que as CDEs são essencialmente centrados na equipe e nas necessidades dela (**?**).

O CDE fornece um espaço de trabalho associado a um conjunto de ferramentas padronizadas para serem usadas pelo time de desenvolvimento. Interoperabilidade e interface são fortes motivações para integrar soluções específicas e *groupware* genéricos dentro dos CDEs (**?**).

Os CDEs foram desenvolvidos em projetos *open source*, porque estes projetos eram desenvolvidos por indivíduos distribuídos geograficamente. Hoje, um número de CDEs estão disponíveis como produtos comerciais, iniciativas *open source* e protótipos de pesquisa que permitem o desenvolvimento distribuído (**?**).

Embora seja difícil definir CSCW e *groupware*, e não existir uma definição unânime, neste trabalho refere-se CSCW a área de pesquisa que estuda softwares colaborativos, ou seja, *groupware*. *Groupware* é então a ferramenta que permite o trabalho colaborativo. Os CDEs, pelas definições

apresentadas, são um subconjunto dos *groupware* e fazem parte do estudo do Desenvolvimento Distribuído de Software (DDS) (*Global Software Development* - GSD).

3.2 Requisitos de CDEs

Na Seção anterior, foi mencionado que embora o desenvolvimento colaborativo traga benefícios, ele é mais complexo do que o desenvolvimento *stand alone* e consequentemente uma ferramenta que auxilie o trabalho em equipe (CDE) é mais complexa de se desenvolver e manter do que uma IDE. Assim, (**?**) indentificam uma série de problemas, provenientes da complexidade que CDEs possuem, e apresentam requisitos que um ambiente colaborativo deve atender para eliminar ou reduzir esses problemas. A seguir são apresentados esses requisitos:

- Submeter o projeto a um controle de versão: os artefatos importantes para o sistema são identificados e é estabelecida uma estrutura para organizá-los em um repositório;
- Permitir o trabalho isolado: é importante em alguns casos certificar-se de que apenas uma pessoa está desenvolvendo (fazendo alterações);
- Integrar o trabalho: após alguma alteração local, seja realizada de maneira isolada, seja em paralelo; ela precisa ser integrada com a versão atual do repositório, o que é chamado de *merge* (intercalar, mesclar), ou seja, as mudanças realizadas devem ser atualizadas. Na maioria dos casos uma ferramenta pode realizar o *merge*, mas em caso de conflito, é necessária intervenção humana;
- Verificar e validar o resultado do *merge*: as ferramentas de *merge* automático não são completamente confiáveis, uma vez que elas não levam em consideração nem a sintaxe e a semântica da linguagem de programação, nem a lógica do sistema;
- Investigar o histórico: é uma grande ajuda poder visualizar como o código foi desenvolvido ao longo do tempo para entendê-lo. Normalmente o histórico mostra a diferença entre duas versões do mesmo componente;
- Manter o desenvolvedor informado: pode ser interessante ao desenvolvedor saber quem está trabalhando em que, quais alterações foram realizadas e quais as consequencias de uma integração na versão corrente do sistema. Para isso, os desenvolvedores precisam ter uma separação individual no repositório. Seria necessário mostrar as diferenças não somente entre o *workspace* e o repositório, mas entre dois *workspaces*, por exemplo.
- Manter a versão anterior

(**?**) também apresentam requisitos de ambientes colaborativos mais específicos de MDD, como:

- Desenvolvimento do modelo da arquitetura (apenas gráfico, sem a necessidade da ação do código): a partir dos requisitos do sistema, um modelo de arquitetura deve ser desenvolvido. O modelo da arquitetura fornece o contexto do sistema para os participantes do projeto, além de descrever as interfaces do sistema;

- Desenvolvimento do modelo de *design*: fornece o detalhamento do sistema que pode ser usado para gerar o código completo;

- Atualização de modelo sem *merge*: ao se criar uma versão nova, a ferramenta de controle de versão verifica pelo *timestamp* do arquivo que não existe outra versão anterior no repositório, não havendo *merge* entre os modelos apenas a inserção desse modelo no repositorio;

- Atualização de modelo com *merge*: para criar uma nova versão de alguns itens do modelo, a ferramenta de controle de versão analisa se há atualizações dos itens. A ferramenta de *merge* compara os itens alterados com os itens do modelo do repositório e faz então o *merge* entre eles, gerando um modelo mais atual, desde que todos os conflitos tenham sido resolvidos;

- Atualização do modelo com *merge* complexo: semelhante ao caso anterior; contudo, a ferramenta de *merge* compara os itens, mas não consegue fazer o *merge* automático entre eles. Dessa forma o usuário é questionado sobre como combinar os resultados de dois modelos conflitantes.

Alguns desses requisitos serão mais detalhados na proposta deste trabalho.

3.3 Os Quadrantes de Colaboração

A colaboração tem duas dimensões: tempo e espaço (**?**). O tempo pode ser síncrono ou assíncrono. O espaço pode ser distribuído ou compartilhado. A Figura 3.1 ilustra os quadrantes que a colaboração pode assumir.

Uma reunião é exemplo de colaboração do *primeiro quadrante*, pois as pessoas estão compartilhando o mesmo tempo e espaço. Duas pessoas que trabalham no mesmo código-fonte e no mesmo computador (logo, na mesma sala), só que em turnos diferentes, é um tipo de colaboração que se enquadra no *segundo quadrante*, pois elas estão compartilhando o mesmo espaço (computador, sala, código-fonte) em horários diferentes. Uma ligação de celular ou vídeo conferência são exemplos de colaboração de *terceiro quadrante*, pois estão compartilhando o mesmo tempo em locais distintos. Um e-mail, blog ou fax são exemplos de colaboração de *quarto quadrante*, uma vez que estão em tempo e locais diferentes.

É importante salientar as quatro facetas da colaboração para que a proposta deste trabalho seja melhor compreendida.

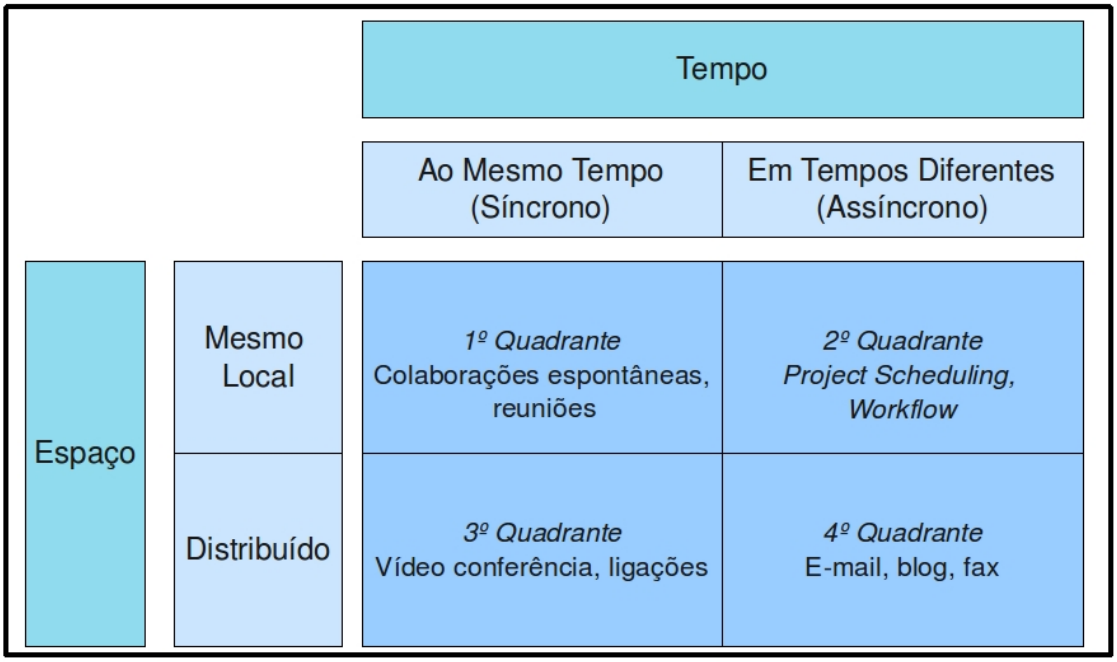

Figura 3.1: Quadrantes da Colaboração (**?**)

3.4 Wikis

As Wikis são consideradas sistemas de computador que tornam fácil a edição de páginas web por qualquer pessoa. Elas também são consideradas uma filosofia em relação à forma como os usuários devem editar páginas (**?**). Em sua essência, a Wiki é uma página web, ou um conjunto de páginas web, editável. Ela permite que qualquer pessoa possa, facilmente, adicionar ou revisar o conteúdo por meio de, praticamente, qualquer navegador web (**?**).

As Wikis têm ganhado muita aceitação pelos usuários (**?**) e sido muito utilizadas no meio acadêmico. Atualmente, têm se mostrado uma excelente opção no mundo das ferramentas colaborativas (**?**) e têm ganhado espaço no setor privado, se tornando uma tecnologia reveladora para apoiar a colaboração dentro e entre empresas (**?**).

A colaboração que a Wiki oferece tem permitido muitos trabalhos científicos em um leque amplo de áreas, como pode-se notar em trabalhos que usam a Wiki para gerenciar conhecimento (**?**) e requisitos (**?**), auxiliar o ensino e aprendizado (**??**), compartilhar informações [1], entre outros.

O suporte à edição colaborativa do conteúdo é, portanto, uma das principais funcionalidades da Wiki (**?**). As Wikis são colaborativas por natureza e a qualidade do conteúdo melhora à medida que as pessoas contribuem (**?**). Todos os usuários habilitados podem editar uma Wiki e revisar seu conteúdo (**?**), de maneira irrestrita e democrática (**?**). A Wiki enfatiza velocidade e flexibilidade ao invés de controle restrito (**?**) e o fato da Wiki ser fácil de implementar e de entender é, provavelmente, a chave para sua popularidade (**?**).

[1] http://en.Wikipedia.org/Wiki/Main_Page

Além da colaboração, simplicidade e facilidade, as Wikis são as plataformas eleitas pela engenharia de requisitos, uma vez que permitem que diversos *stakeholders* distribuídos, compartilhem a definição do sistema; além do que as Wikis têm controle de versão de páginas (*pages versions*), gerenciamento de alterações de conteúdo, comunicação entre usuários por meio de fóruns e *chats* (**?**). Por permitirem a edição em tempo real do conteúdo, o conteúdo de uma Wiki está sempre atualizado (**?**). **O Masieiro disse que nao é verdade, mas eu acho que é. O que a sra acha?**

O que as torna tão especiais, não é apenas a facilidade de contribuição, mas também seus principais recursos (**?**):

- Simples sintaxe de marcação;
- Criação de links simples ou automáticos, mesmo quando a página de destino ainda é inexistente;
- Controle de acesso total de usuários e páginas;
- Histórico completo das revisões, com possibilidade de reverter a qualquer momento;
- Muitas Wikis não precisam de banco de dados, o que as deixa portáveis;
- Categorizar páginas para melhor organização;
- Criação automática de uma tabela de conteúdo (índice);
- Amplos recursos de pesquisa;
- Customizáveis.

Atualmente, existem centenas de Wikis desenvolvidas, mas exemplos que se destacam são as seguintes Wikis: MediaWiki[2], DokuWiki[3], Xwiki[4], entre outras. Em especial, uma boa opção de Wiki para este trabalho é a Xwiki, por ser implementada em Java. A MediaWiki e a DokuWiki também são boas opções, por já serem utilizadas em outros trabalhos desenvolvidos pelo grupo.

As Wikis focam na velocidade, flexibilidade, colaboração e compartilhamento de conteúdo; ao invés do controle restrito de acesso e edição (**?**).

[2]http://www.mediawiki.org/wiki/MediaWiki
[3]http://www.dokuwiki.org/dokuwiki
[4]http://www.xwiki.org/xwiki/bin/view/Main/WebHome

CAPÍTULO

4

Trabalhos Relacionados

Apresentar nesta seção:

- Artigos
- Ferramentas de MDD
- Ferramentas de Colaboração
- Talvez editores simultâneos caso consigamos implementar

ARTIGOS: qual desses podemos tirar e que tá menos relacionado com o nosso?

Há uma literatura relativamente vasta sobre o tema de pesquisa deste trabalho, o que ressalta sua importância de estudo. Nesta Seção será apresentado um levantamento de alguns trabalhos relacionados.

O trabalho de Levytskyy et al (2009), está focado em aplicações M&S (*Modeling & Simulation*) desenvolvidas na web. Os autores atentam para o fato de que a modelagem de artefatos requer o envolvimento de várias pessoas e que, por isso, o desenvolvimento tradicional da forma individual, usando ferramentas M&S, deverá ser substituído ou complementado com ferramentas mais poderosas que apoiam a prática colaborativa dos usuários.

Como a complexidade dos sistemas continua a crescer, é esperado que o uso de M&S no desenvolvimento de aplicações e a demanda por ambientes M&S colaborativos e mais poderosos aumente. Além do que, separar modelos de seus simuladores possibilita aos usuários a liberdade de executar seus modelos nos simuladores de sua escolha (**?**). A partir desse cenário e das expectativas futuras, o trabalho propõe um ambiente colaborativo baseado na web, que seja capaz de se adaptar

facilmente às necessidades de modelagem e simulação de um número de domínios de aplicações M&S arbitrário. Para isto, os autores sugerem um *framework collaborative Modeling and online Simulation* (cMoS) como suporte para aplicações M&S desenvolvidas em plataforma web.

Para a construção do ambiente, os autores apresentam uma abordagem orientada a modelos. O intuito de usar MDD para construção do ambiente é reduzir a complexidade e aumentar a eficiência da implementação de modelos cMoS.

O cMoS proporciona uma base geral para a família de aplicações M&S desenvolvidas em plataforma web, tendo como algumas funcionalidades: modelagem, compartilhamento colaborativo de modelos conceituais, simulação online e gerenciamento dos recursos de simulação. A colaboração, portanto, fornece o suporte necessário ao *framework*.

Outro trabalho relativo a colaboração em modelagem, é o trabalho de Abeti et al (2009). Eles desenvolvem seu trabalho sob uma plataforma Wiki no contexto de gerenciamento de requisitos para *Business Process Reengineering* (BPR). Eles propõem um *framework* BPR que formalize o conhecimento empresarial por meio de um conjunto de modelos conectados com os requisitos do software e que também possibilite uma automação parcial da implementação do sistema. Esse *framework* provê uma Wiki, denominada de *Wiki for Requirements (WikiReq)*, baseada na plataforma *Semantic MediaWiki*.

Para representação do conhecimento, os autores combinam os benefícios de três notações: Si*, diagramas de Casos de Uso em UML e *Business Process Management Notation* (BPMN). Através da notação Si*, os *stakeholders* podem definir seus conhecimentos sobre *Business Process* (BP) e sobre os requisitos do negócio dentro da WikiReq; os diagramas de casos de uso são intuitivos e são os diagramas mais usados por *stakeholders* e a notação BPMN é uma notação de BP fácil de compreender por todas as empresas interessadas (**?**).

Para a automação da implementação parcial do sistema, a WikiReq exporta para IDE Eclipse um conjunto de modelos que são conectados a BMPN e modelos de caso de uso em UML, por meio de um conjunto de plugins (**?**).

A partir do início do levantamento de requisitos e dos BPs da empresa adquiridos pelos *stakeholders*, combinar as funcionalidades da Wiki com a modelagem MDA é uma tarefa fundamental para manter a rastreabilidade entre artefatos de sistemas e requisitos. As funcionalidades do MDA automatizam parcialmente o gerenciamento do conhecimento por meio das transformações de modelos (**?**).

Um trabalho que contribuiu para o CoMDD foi o trabalho de Bendix e Emanuelsson (2009), que identifica um problema quanto às limitações dos desenvolvedores em MDD, principalmente relacionadas a carência de ambientes que auxiliem o desenvolvimento orientado a modelos de maneira colaborativa. Assim, por meio de casos de uso, os autores realizam um levantamento dos requisitos que um ambiente colaborativo, em particular orientado a modelos, precisa satisfazer, de tal forma que possibilite a um time de desenvolvedores trabalhar eficientemente neste ambiente.

Existem na literatura trabalhos que propõem técnicas de *merge* e de comparação entre modelos (**???**), entre outros. Tais trabalhos são importantes para o CoMDD, uma vez que foi identificada a importância das técnias de *merge* e de comparação entre modelos para o desenvolvimento colaborativo orientado a modelos. Existem também *workshops* sobre comparação e versionamento de modelos (**?**) mostrando a importância da área.

Existem muitas ferramentas que possibilitam o MDD como o Eclipse, Matlab/Simulink, WebRatio, Labview, AndroMDA, mas nenhuma delas se preocupa com colaboracao, como edição simultânea por exemplo. Já ferramentas como ThoughtSlinger, Notapipe, Zoho, Moonedit e o Google Docs, permitem edição simultânea, mas não desenvolvimento de sistemas, visto que são ferramentas de escritório. E ferramentas como sharepoint, creatly, gforge auxiliam o trabalho (comunicação e troca de conhecimento entre os envolvidos) colaborativo mas não permitem uma modelagem colaborativa, ou seja, no CoMDD o usuário desenvolve o sistema a partir da modelagem (qual o nosso diferencial para estas ferramentas).

No CoMDD, a idéia é que todo o processo possa ser realizado pela web, sem a necessidade de programas instalados no computador e que permita o desenvolvimento distribuído de software.

Editores simultâneos:

- http://www.thoughtslinger.com/
- http://notapipe.net/
- https://writer.zoho.com/home?serviceurl=
- http://moonedit.com/tutor.htm

Ferramentas de MDD:

- http://www.dsmforum.org/events/MDD-TIF07/
- http://www.infopar.com/br/produtos/mdd-mdsd-ferramenta-desenvolvimento-software-gerador-codigo-programas.aspx
- Matlab/Simulink
- Labview

Ferramentas que auxiliam o trabalho colaborativo e compará-las com o comdd mostrando o que temos de diferencial:

- Sharepoint
- Creately

CAPÍTULO

5

CoMDD

Nesta será ...

O CoMDD consiste de uma ferramenta de modelagem e de um mecanismo que aplica as transformações, que roda sob uma wiki e que permite a colaboração de várias pessoas. Assim, esta seção define os três pontos principais do CoMDD e suas principais vantagens.

5.0.1 Edição simultânea, Merge e Divisão de Tarefas

No desenvolvimento tradicional e distribuído, geralmente, os desenvolvedores fazem uma cópia do servidor para sua máquina local, alteram essa cópia e depois submetem uma versão modificada para o servidor, havendo o merge de documentos neste momento. Quando duas pessoas baixam o mesmo arquivo do servidor, o alteram e depois submetem novamente, haverá conflito de dados. Uma forma de solucionar esse problema é através da edição simultânea.

Assim, o CoMDD usa a edição simultânea e define um processo para que desenvolvedores trabalhem no mesmo artefato e façam merge só quando necessário.

A abordagem analisa 4 pontos:

1. **Artefatos diferentes e funcionalidades diferentes** É uma tarefa auto-contida e independente, ou seja, neste caso o desenvolvedor pode criar, editar e compilar um artefato, de modo que outro desenvolvedor esteja trabalhando em outra tarefa auto-contida e independente. Assim as tarefas o desenvolvimento das tarefas não interferem uma na outra.

2. **Mesmo artefato e mesma funcionalidade** Quando dois desenvolvedores estão trabalhando juntos no mesmo artefato eles têm suporte à edição simultânea dele e tem suporte de chat e conferência de voz.

3. **Mesmo artefato e funcionalidades diferentes** Aqui tem um problema, porque caso o desenvolvedor A queira compilar enquanto o desenvolvedor B estiver editando o artefato haverá erros de compilação, por isso, para contornar o problema, o servidor cria uma cópia do artefato que está sendo editado para cada desenvolvedor, como se fosse uma cópia local, e depois os desenvolvedores fazem o submit e o servidor faz o merge dos artefatos. É apenas neste caso em que haverá o merge.

4. **Artefatos diferentes e mesma funcionalidade** Essa atividade é evitada pelo CoMDD porque gera conflitos e não promove discussões. Assim, ou um artefato é editado simultaneamente por várias pessoas ou apenas uma pessoa pode editá-lo.

Com a edição simultânea promovemos mais discussão, pois todos envolvidos participam da edição do artefato. Para a edição simultânea não gerar conflitos, define-se duas regras:

- Não pode duas ou mais pessoas editarem ao mesmo tempo a mesma linha de código, para isso elas devem editar partes separadas do código (para todos os casos).

- Caso alguém queira compilar, todas devem estar de acordo porque se não dificilmente vai compilar, dado que uma pode estar digitando enquanto a outra quer compilar (para o caso 2).

Uma das vantagens da edição simultânea na web é a possibilidade de que pessoas geograficamente distantes possam trabalhar colaborativamente em um mesmo modelo.

Uma outra possibilidade para o uso da abordagem, além do desenvolvimento de sistemas, é em treinamentos de programação, por exemplo; de modo que professor e aluno possam programar juntos.

5.0.2 Transformações e Validações

Um modelo, objeto editado colaborativamente ou individualmente, pode ser transformado em código-fonte ou em outros modelos, mas para isso acontecer transformações devem ser definidas. O CoMDD precisa de uma linguagem específica de domínio com transformações definidas para poder gerar código-fonte e um metamodelo definido de forma que o modelo seja validado, ou seja, um modelo só tem significado e pode gerar código-fonte caso ele siga um metamodelo.

A atual implementação do CoMDD encontra-se nesta fase e no processo de edição simultânea.

5.0.3 Comunicação

Para que o CoMDD permita o desenvolvimento distribuído de software é necessário um suporte a comunicação. Assim, o CoMDD define o uso de uma interface de comunicação textual ou por voz e que possibilite a contribuição de todos para termos modelos mais completos.

5.0.4 Benefícios

Os benefícios dessa abordagem são:

- Aumento da discussão e participação entre os envolvidos
- Possibilidade de Pair Programming por pessoas distribuídas geograficamente
- Registro das decisões tomadas durante o desenvolvimento (Design Rational) a partir da interface de comunicação
- Independência de tecnologia e de programas instalados localmente
- Aumento do reuso e produtividade, uma vez que a abordagem usa MDD

5.1 Conclusão

A partir da abordagem definida espera-se que o desenvolvimento orientado a modelos seja usado por mais desenvolvedores e que os desenvolvedores que usam MDD possam se beneficiar das vantagens de trabalhar-se colaborativamente. Atualmente o CoMDD está sendo desenvolvido pelo ICMC-USP e está na fase

CAPÍTULO

6

Proposta

Neste capítulo serão apresentados: o problema de pesquisa do trabalho, a importância dele para a Engenharia de Software, o objetivo e a hipótese da pesquisa, o escopo e as contribuições que o trabalho pretende alcançar.

6.1 Problema de Pesquisa

O MDD é uma abordagem que se concentra na concepção e transformação de modelos. Os modelos são o foco central dos artefatos, que por meio de transformações podem gerar código-fonte (**?**). O MDD tem se mostrado uma abordagem promissora, emergindo ao longo das últimas décadas (**?**) e apresentado ganhos de produtividade no desenvolvimento de sistemas (**?**).

Embora o MDD tenha se tornado uma abordagem madura, existem soluções muito incipientes em termos de ferramentas e processos necessários para auxiliar os desenvolvedores em seus trabalhos (**?**).

No desenvolvimento tradicional, ferramentas permitem à equipe de desenvolvedores trabalhar e coordenar as atividades usando ferramentas sofisticadas de combinação de código e de divisão de tarefas. Contudo, pela falta das mesmas ferramentas de apoio no contexto de MDD e pela falta do uso de modelos para criar conhecimentos compartilhados, as equipes de desenvolvimento trabalham de maneira mais restrita (**?**), (**?** *apud* **?**).

Neste trabalho define-se o termo *traditional Model-Driven Development* (tMDD) como sendo o desenvolvimento orientado a modelos tradicional, aquele já conhecido e típico da indústria ((**??**)) e academia ((**??**)), o qual não permite que seja realizado de maneira simultânea e distribuída (3º

quadrante da Figura 3.1). Assim, no tMDD, o trabalho é realizado de forma isolada; ou para ser colaborativo necessita que os desenvolvedores estejam juntos (localmente próximos), trabalhando sobre o mesmo artefato e com aplicações *stand alone* (1º quadrante da Figura 3.1).

Mas por que criar uma abordagem que integre MDD e Colaboração?

O desenvolvimento de software está mudando da programação manual para o MDD, como uma maneira de reduzir a crescente complexidade dos sistemas (**??**) e o desenvolvimento de software é um processo intensamente colaborativo onde o sucesso depende da habilidade de criar, compartilhar e integrar informação (**?** *apud* **?**). Assim, acredita-se que uma abordagem que junte os benefícios da colaboração ao MDD, possa trazer essencialmente o aumento de produtividade.

Embora existam outros tipos de colaboração no MDD (seja por repositórios (**?**), seja pela simples troca de conhecimento por e-mails), a modelagem simultânea e distribuída ainda é motivo de pesquisa e um tema bastante novo na literatura, de maneira que ainda não existe, ou pelo menos não se tem conhecimento, nenhuma ferramenta ou ambiente de MDD colaborativo de maneira simultânea e distribuída; sendo a maioria dos trabalhos encontrados nessa área (**???**) apenas propostas idealizadas de um ambiente assim.

A importância dessa linha de pesquisa reside no fato de que a colaboração pode trazer diversos benefícios ao MDD tradicional, assim como ela já fornece para o desenvolvimento tradicional orientado a código [1].

6.2 Objetivo e Hipótese

Diante do problema, descrito na Seção anterior, de que há uma carência de ferramentas que apoiem o densenvolvimento colaborativo orientado a modelos e observando as incipientes propostas na literatura, neste trabalho objetiva-se confirmar a seguinte hipótese:

> **Hipótese:** A colaboração pode contribuir de maneira positiva, trazendo benefícios para o MDD tradicional e conseqüentemente para a equipe e para o projeto.

Portanto, acredita-se que com o uso de colaboração[2] no MDD haja um aumento na produtividade do desenvolvimento de software. Assim, é também um dos **objetivos específicos** deste trabalho apresentar o porquê a produtividade seria aumentada com o uso da colaboração no desenvolvimento orientado a modelos.

[1] Desenvolvimento tradicional orientado a código ou simplesmente desenvolvimento tradicional ou desenvolvimento orientado em nível de código; são conceitos sinônimos referindo-se ao desenvolvimento de sistemas com qualquer paradigma de implementação orientado a código (procedural, orientado a objetos, lógico e etc); popularizado e empregado na indústria e ensino.

[2] A colaboração que é referida neste Capítulo é aquela em que duas ou mais pessoas possam trabalhar juntas em um mesmo artefato, mas distribuídas geograficamente - terceiro quadrante da Figura 3.1

Embora a colaboração possa ser um excelente suporte ao desenvolvimento de software, seja orientado a código, seja orientado a modelos - como se quer provar, ao inserir a colaboração no desenvolvimento de um sistema, diversos problemas que antes não existiam ou eram menos evidentes, passam a ficar mais acentuados, conforme mencionado na Seção 3.1.

A partir dos problemas expostos na Seção 3.2, este trabalho foca mais nas questões de controle de versão, em manter os desenvolvedores cientes da evolução do sistema e em permitir o desenvolvimento distribuído de software por meio de um CDE.

Um gerenciador de versões que suporte modelos, que calcule a diferença entre dois modelos, que faça o *merge* e os apresente ao desenvolvedor, fornecendo um modelo consistente, é importante para auxiliar o trabalho cooperativo de uma equipe (**?**)[3]. A comparação entre modelos e o *merge* se torna uma importante característica necessária para manter a consistência dos modelos ao longo do desenvolvimento; uma vez que a construção de um modelo pode ser feita por diversos desenvolvedores, e para isso todos precisam estar cientes e terem acesso ao estado corrente do modelo em construção.

Existem diversas técnicas que propõem abordagens para a comparação entre modelos (**????**) e para *merge* de modelos (**??**). Contudo, mesmo ressaltando-se a elevada importância desses dois tópicos (comparação e *merge*) para a área, ainda não foi definida nenhuma abordagem para tal, neste trabalho. Portanto, ainda será feito um levantamento mais detalhado das atuais abordagens de comparação entre modelos e de *merge*, como sendo outro **objetivo específico** do trabalho a ser detalhado na Seção 6.4.

Além dos problemas apresentados por (**?**), ainda foi identificado pelo grupo mais dois problemas que ocorrem com o desenvolvimento colaborativo. Um seria em relação ao acesso aos modelos, ou seja, nem todos os desenvolvedores podem ter acesso a todos os modelos. Dessa forma, é importante que haja um controle de permissão para acesso aos modelos.

Outro problema identificado é quanto às razões para tomada de decisões. Muitas vezes, as razões que levaram a uma determinada escolha em detrimento de outra são ignoradas. Assim, documentar o desenvolvimento de um projeto é de fundamental importância para evitar erros cometidos, reutilizar soluções, ajudar na manutenção do sistema, resolver problemas, melhorar a comunicação da equipe, verificar o *design* e gerar documentação (**?**).

Deste modo, este trabalho pretende resolver os principais problemas do desenvolvimento distribuído de software, segundo (**?**): ambientes e ferramentas de desenvolvimento e *Design Knowledge*, entre outros apresentados por (**?**). Para isso, o trabalho apresenta como estudo de caso um ambiente chamado de CoMDD (Colaborative Model-Driven Development), o qual permitirá realizar o desenvolvimento distribuído de software com modelagens e transformações; o controle de verão e de permissão; o suporte a *design rationale*; a interoperabilidade entre modelos e a independência de plataforma; sendo projetados como prova de conceito, os seguintes produtos, que serão melhor explicados na Seção 7.1:

[3]Colocar outra referencia a mais que fale da importância do merge

- Um editor web colaborativo de modelagem, que irá prover um mecanismo para que diversos desenvolvedores possam criar e alterar modelos de software;
- Um mecanismo de meta-modelagem, provendo a infra-estrutura necessária para a criação de linguagens específicas de domínio, e a manipulação dos modelos criados, possibilitando, por exemplo, a geração de outros modelos/código;
- Um mecanismo de transformação, responsável pelos refinamentos automatizados dos modelos até o código;
- Uma estrutura que permita o controle de permissão;
- Um mecanismo que comporte o controle de versão;
- Um mecanismo que permita a inserção e o registro do conhecimento das razões sobre as decisões do desenvolvimento.

A partir dos resultados desses produtos, objetiva-se ainda definir um processo visando auxiliar o desenvolvimento de software orientado a modelos adotando-se o CoMDD.

Assim, como resultado deste projeto, será disponibilizado um ambiente web colaborativo para o desenvolvimento orientado a modelos, no qual diversos desenvolvedores poderão criar modelos e automaticamente gerar outros modelos/ código; juntamente com um processo que auxilie o desenvolvimento. Este ambiente irá apoiar o controle de versão e de permissão, além de possibilitar o conhecimento das decisões ocorridas durante o desenvolvimento.

Por ser uma área nova e precisar integrar diversas linhas de pesquisa (versionamento e *merge*, DR, DDS, MDD e outras), tem-se poucos trabalhos já desenvolvidos e a grande maioria deles são protótipos ou apenas propostas sem nenhuma implementação.

6.3 Estudo de Caso: CoMDD para Sistemas Embarcados

Sistema Embarcado é um termo genérico para uma ampla gama de sistemas, como por exemplo: celulares, sistemas de transporte ferroviário, aparelhos auditivos, marca passos, sistema de monitoramento dos mísseis, entre outros. No entanto, todos os sistemas embarcados têm uma características em comum: interagir com o mundo real controlando algum hardware específico (**?**). A Figura 6.1 apresenta um esquema genérico da arquitetura de um sistema embarcado.

O software é armazenado (embarcado) em um tipo de memória não volátil (NVM). Geralmente é uma memória ROM, mas o software também pode ser armazenado em memória *flash*, em HD (*hard disk*) ou *downloaded* via uma rede ou satélite. O software embarcado é compilado para um processador específico, com uma unidade de processamento (*processing unit*) que opera com uma quantidade específica de memória RAM (**?**).

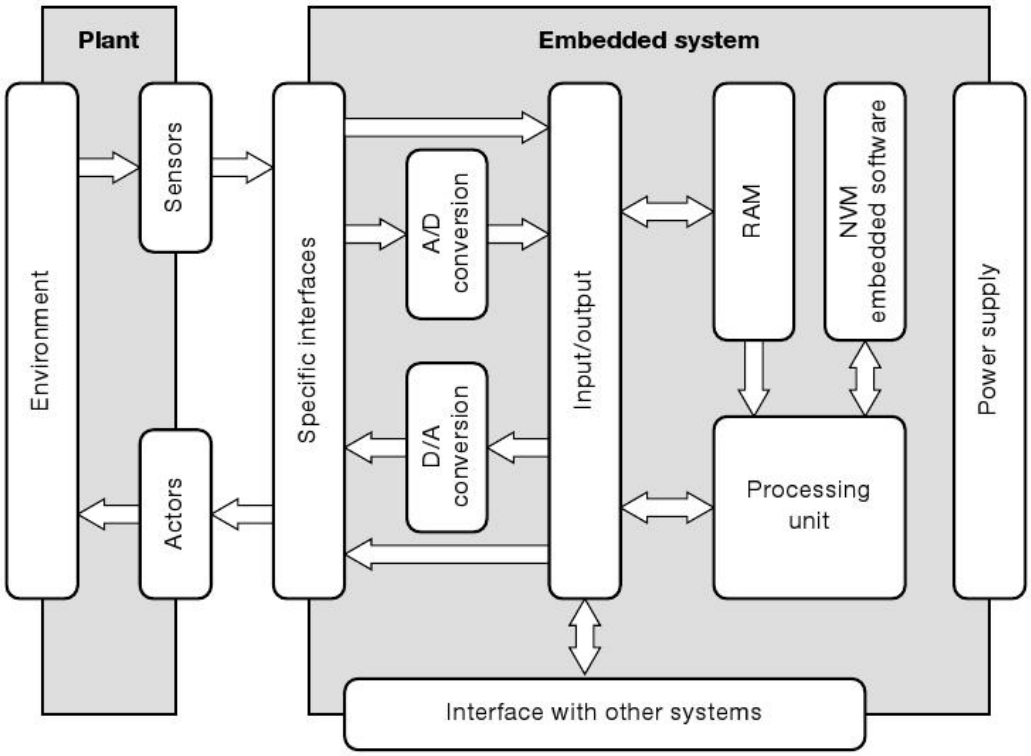

Figura 6.1: Esquema Genérico da Arquitetura de um Sistema Embarcado (**?**)

Como a unidade de processamento trabalha apenas com sinais digitais (desconsiderando os computadores analógicos) e o ambiente lida, provavelmente, apenas com sinais analógicos, os conversores analógicos-digitais (A/D) e digitais analógicos (D/A) intermediam a comunicação entre o ambiente e o sistema. A unidade de processamento manipula toda entrada e saída de sinais através de uma camada dedicada para isso (*input/output - I/O layer*). O sistema interage com a planta e com outros sistemas através de interfaces específicas (*Specific interfaces*) (**?**).

Especificidades dos Sistemas Embarcados

Entre 1990 e 2009, o impacto do software nas funcionalidades de sistemas embarcados, bem como no potencial de inovação de novos produtos, cresceu rapidamente. Isto levou a um enorme crescimento na complexidade de softwares, períodos de inovação mais curtos e à procura sempre crescente por requisitos não funcionais, como segurança, confiabilidade e rapidez, a custos acessíveis.(**?**):

Os sistemas de TI (Tecnologia da Informação) possuem plataformas de desenvolvimento maduras que fornecem a infra-estrutura básica para arquiteturas orientada a serviço, sistemas distribuídos, recuperação de erros e segurança, além de uma série de outros serviços que estes ambientes de desenvolvimento oferecem. Entretanto, tal abordagem não ocorre no domínio de sistemas embarcados. Requisitos oriundos de custo, energia, restrições de calor, humidade, radiação, tamanho

e/ou peso, demandam pelo uso eficiente de recursos de hardware disponíveis. A diversidade de sistemas embarcados também impede a criação de uma plataforma única especializada (**?**):.

A necessidade de criar um hardware e software especializados é um ponto chave que diferencia os sistemas de TI dos sistemas embarcados. Requisitos não funcionais contribuem para a difícil tarefa de criar uma plataforma única de desenvolvimento. Tais requisitos são mais importantes em sistemas embarcados que em sistemas de TI. Além disso, os requisitos extrafuncionais costumam entrar em conflito com os recursos de custo e outras limitações como peso, energia, resposta em tempo tempo real, entre outras. Por fim, outro fator que dificulta o desenvolvimento destes tipos de sistemas é que desenvolvedores precisam de um extenso conhecimento do domínio (**?**).

Assim, com o aumento da complexidade dos sistemas embarcados e a exigência cada vez maior por qualidade, o papel da Engenharia de Software tem se tornado cada vez mais importante nesta área (**?**).

Abordagem MDD em Sistemas Embarcados

O MDD em sistemas embarcados começou antes da UML e da MDA serem padronizadas. Um exemplo disso são as linguagens de modelagem Matlab/Simulink[4] e Labview[5]. Com essas ferramentas, desenvolvedores podem especificar sistemas embarcados completos usando modelos de alto níveis[6] (**?**).

Para lidar com a ineficiência dos métodos atuais de desenvolvimento , muitos esforços de pesquisa na academia e na indústria estão concentrados no desenvolvimento de novas tecnologias, as quais devem ser capazes de lidar com o crescimento da complexidade do processo de desenvolvimento de sistemas embarcados. Abordagens baseadas em MDD têm sido propostas como uma metodologia eficiente para o *design* de sistemas embarcados (**?**). (**?**) afirmam que sistemas de grande escala, desenvolvidos por vários engenheiros, agora estão usando abordagens orientada a modelos, pois o MDD é essencial para reduzir custos, manter e evoluir sistemas complexos de software. O principal inconveniente de usar uma metodologia centrada no código é a dificuldade de manter e estender o sistema. Por isso, uma metodologia alternativa é o MDD. Os autores também constróem com sucesso um protótipo em menor escala.

Ainda é incerto se a abordagem MDD (e suas implementações atuais) poderiam ser suficientemente rigorosas para a alta integridade do desenvolvimento de softwares embarcados. Contudo, vale a pena examinar essa possibilidade por muitas razões. Primeiramente, uma vez que a habilidade de lidar com mudanças e reúso é fundamental para o MDD, então existe a possibilidade da abordagem MDD poder ser usada para certificações incrementais. Em segundo, o MDD permite o desenvolvimento modular, usando transformações para produzir modelos de componentes fundidos (mesclados). Isto é potencialmente compatível com uma abordagem de certificação modular.

[4]http://www.mathworks.com/products/simulink/
[5]http://www.ni.com/labview/
[6]http://www.ni.com/robotics/

Também é importante notar que a abordagem MDD tem sido usada para implementar missões crí ticas de software (**?**).

Colaboração em MDD para Sistemas Embarcados

A colaboração é essencial em todos os domínios (**?**). Um software embarcado raramente é desenvolvido como um produto singular, mas como um elemento de um sistema embarcado, que é composto de outros elementos (circuitos elétricos, eletrônicos, peças mecânicas, etc). Assim, o desenvolvimento de sistemas embarcados torna-se mais colaborativo do que o desenvolvimento de sistemas de informação, em geral, pois para o desenvolvimento de um sistema embarcado é necessário não somente os desenvolvedores do software, como também a intervenção dos demais profissionais envolvidos, como: os engenheiros elétricos, eletrônicos, mecânicos, da computação, dentre outros(**?**).

Assim, acredita-se que a colaboração em MDD para sistemas embarcados possa trazer benefícios (iguais ou não aos benefícios de TI), assim como traria para os sistemas de TI (como se quer mostrar). O MDD já é empregado com sucesso em sistemas embarcados, os quais são desenvolvidos por diversos profissionais, mas que ainda não se beneficiam de um auxílio ao trabalho colaborativo. Logo, cabe ainda neste trabalho, como mais um dos objetivos específicos, identificar se a colaboração pode contribuir para o desenvolvimento destes sistemas e se sim, quais seriam os benefícios e a que circunstâncias se aplicariam sem perda de desempenho.

6.4 Objetivos Específicos

Os objetivos específicos já foram identificados ao longo do trabalho e nesta Seção serão apresentados formalmente. Portanto, são objetivos específicos que este trabalho pretende alcançar:

- Identificar quais benefícios a colaboração associada ao MDD traz ao desenvolvimento de sistemas de TI;
- Identificar quais benefícios a colaboração associada ao MDD traz ao desenvolvimento de sistemas embarcados;
- Explicar o porquê a produtividade aumenta com o uso de colaboração em MDD;
- Levantar mais detalhadamente as atuais técnicas de comparação entre modelos e técnicas de *merge* entre modelos;

6.5 Considerações Finais

Neste capítulo foram apresentados a proposta do trabalho, bem como os objetivos que ele pretende alcançar. No capítulo seguinte será apresentado o método que este trabalho pretende seguir para alcançar seus objetivos almejados.

CAPÍTULO

7

Método

Neste capítulo será apresentado o método para a realização do trabalho e o cronograma de atividades previsto.

7.1 Método

São previstas cinco atividades que serão responsáveis por estender um ambiente colaborativo web (CMS[1]) de tal forma que possibilite o desenvolvimento colaborativo orientado a modelos. As atividades deverão compor a prova de conceito a ser desenvolvida.

Para testar o ambiente serão realizados dois estudos de caso: um no domínio de sistemas de TI e outro no domínio de sistemas embarcados. A intenção de realizar dois estudos de caso é levantar características (vantagens, desvantagens e singulariedades) que a colaboração pode trazer para cada um desses domínios e depois compará-los entre si. Deste modo, espera-se que com a execução do método e com a realização dos estudos de caso, seja validada a hipótese de pesquisa deste trabalho.

7.1.1 Inclusão de Suporte à Metamodelagem

Um modelo, no contexto deste projeto, é uma especificação abstrata de um conceito, e pode surgir a partir das atividades de análise, projeto ou implementação. Neste sentido, um modelo pode

[1]Neste capítulo, será simplificado o termo ambiente colaborativo web, podendo ser um Wiki, um CMS ou um intraCMS - ver Seção **??** - para CMS, por ser um termo mais abrangente; contudo, é importante ressaltar que ainda não está definido o tipo de CMS a ser usado no trabalho.

ser visual, como por exemplo um diagrama de classes ou de estados, ou mesmo textual, como por exemplo uma especificação formal. Para que um modelo possa servir de entrada para um transformador ou um gerador, é necessária uma estrutura que defina seus elementos, relacionamentos, e as informações contidas no modelo.

Esta estrutura é também conhecida como metamodelo (**?**). Um gerador ou transformador precisa conhecer o metamodelo para poder executar sua função. Enquanto muitas ferramentas suportam apenas um número fixo de metamodelos (tais como as ferramentas UML, que somente suportam o metamodelo da UML), no desenvolvimento orientado a modelos é desejável a existência de diferentes metamodelos, para dar suporte às DSLs (**?**), tornando a atividade de modelagem mais flexível e natural para os desenvolvedores e especialistas do domínio.

Esta atividade tem como objetivo oferecer o suporte à metamodelagem no ambiente colaborativo web. Marcações especiais simplificadas permitem que o usuário formate o texto de forma mais intuitiva e eficiente, sem a necessidade de conhecer comandos específicos de uma linguagem de formatação, como HTML, por exemplo. Um exemplo é a quebra de linha: em HTML, as quebras de linha inseridas no código são ignoradas, sendo necessário incluir o comando `<br>`, enquanto uma Wiki, por exemplo, mantém automaticamente as quebras inseridas.

Para isso, três principais funções serão incluídas, conforme ilustra a Figura 7.1.

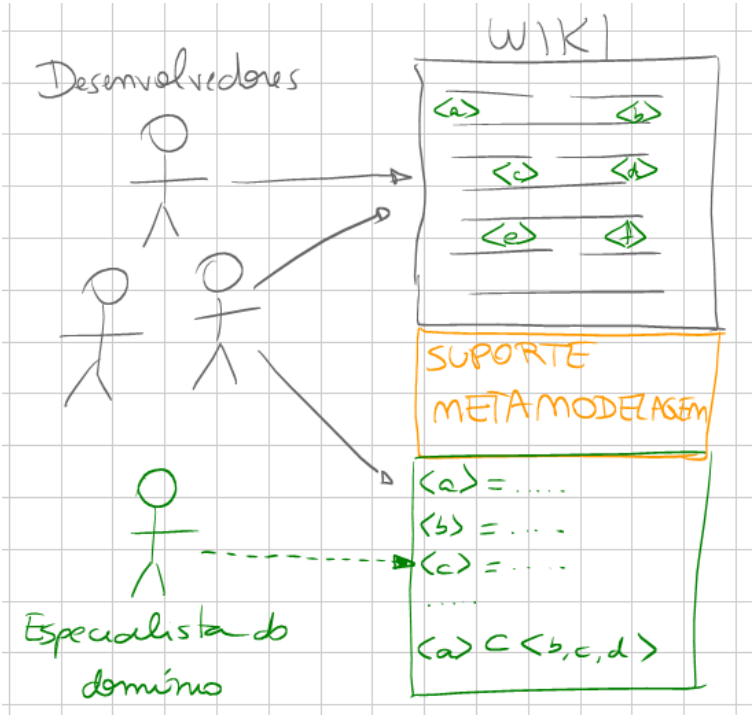

Figura 7.1: Suporte à metamodelagem em uma Wiki

Na *primeira função*, o CMS estendido irá permitir que especialistas do domínio criem descrições do metamodelo em forma de marcações (*tags*) a serem inseridas no texto. Estas descrições especificam quais são as *tags* existentes para um domínio, e como elas se relacionam entre si (ex: uma *tag* deve estar contida em outra, uma *tag* A deve vir após outra *tag* B, etc).

A *segunda função* será a inserção das *tags* definidas pelo especialista em um texto qualquer. Um desenvolvedor utiliza as *tags* disponíveis em determinadas partes do texto, enriquecendo-o com informações específicas daquele domínio. A *terceira função* deste CMS estendido é interpretar este texto, enriquecido com base nas *tags* do domínio (metamodelo).

O resultado destas funções é que o texto marcado pelo desenvolvedor passa a ser um modelo específico de domínio, passível de interpretação automática, o que possibilita, por exemplo, a geração de código, geração de outros modelos, análises automáticas, validação, otimizações, etc. Além disso, este modelo específico de domínio está situado em um ambiente colaborativo, permitindo que estas operações sejam executadas diretamente na Wiki.

A Figura 7.1 mostra o especialista do domínio definindo o metamodelo do domínio. O metamodelo descreve o significado das *tags*, e como elas se relacionam. Deste modo, o mecanismo de suporte à metamodelagem irá prover meios para inserção das *tags* no texto, assim como uma forma para ler/interpretar o seu conteúdo, numa preparação para a subseqüente geração de código e/ou transformações. Isso possibilita que os desenvolvedores utilizam uma Wiki, por exemplo, para criar os modelos de forma colaborativa.

7.1.2 Suporte à Geração de Código

Uma vez que modelos com as marcações (*tags*) estejam disponíveis, é possível ler/ interpretar os modelos e realizar diferentes operações automáticas tais como transformações, otimizações ou validação. Porém, uma das principais operações a serem realizadas é a transformação modelo-para-texto, ou geração de código. Este é o objetivo desta atividade.

A transformação de modelo para código é desenvolvida por um especialista (normalmente com conhecimento do metamodelo e da plataforma de implementação), e é responsável por interpretar os dados do modelo, de acordo com o metamodelo, e gerar texto de saída.

A construção das transformações é facilitada pelo suporte à metamodelagem desenvolvido na atividade anterior. Através dele, o desenvolvedor da transformação pode definir regras de mapeamento entre os elementos do metamodelo e o código final desejado.

A Figura 7.2 ilustra um desenvolvedor de transformações definindo uma transformação com base no metamodelo, para possibilitar a geração de código.

7.1.3 Extensão da Interface

Nesta atividade, a interface será estendida para incluir as funções de:

- Criação/edição de metamodelos;
- Inserção de *tags* do metamodelo em documentos; e
- Execução/chamada das transformações.

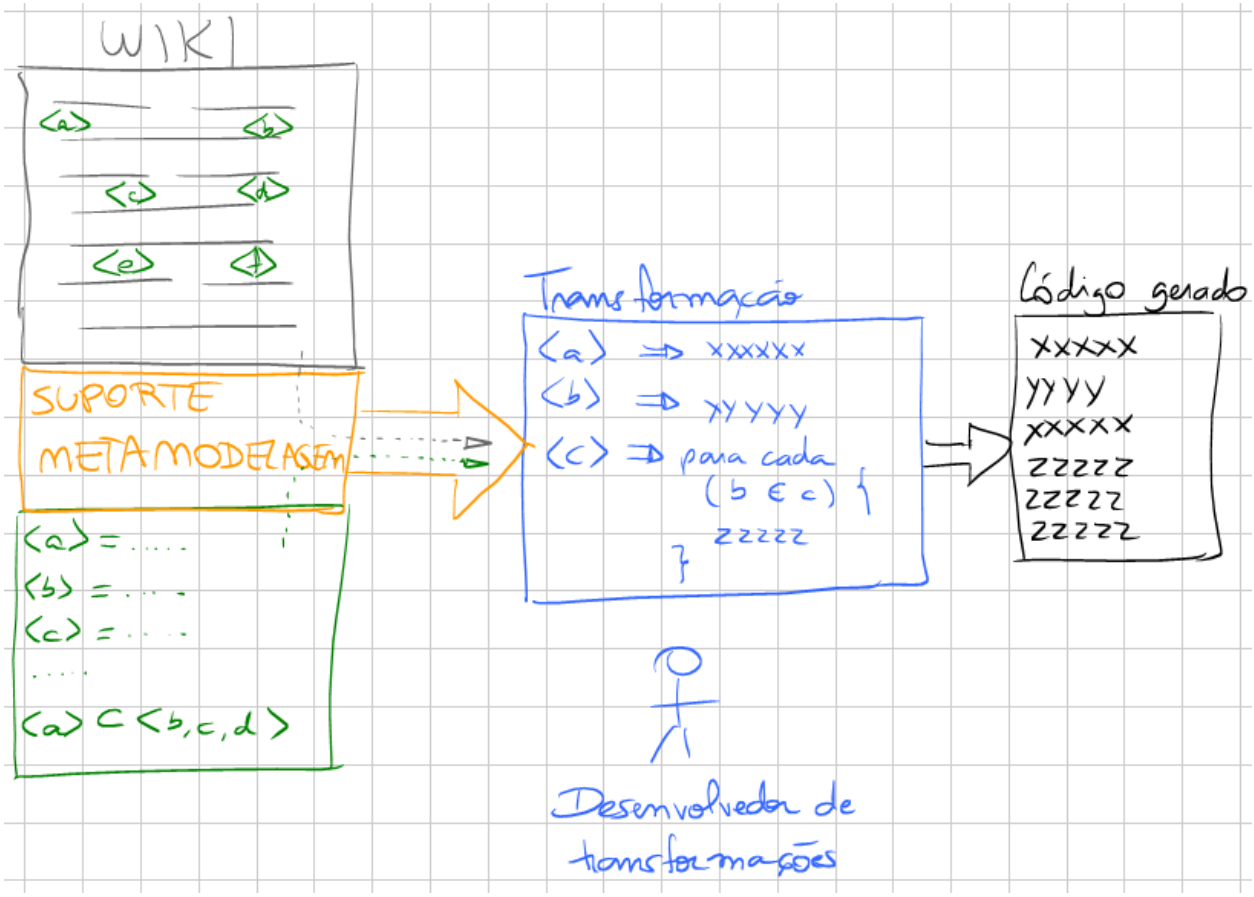

Figura 7.2: Suporte à geração de código

A idéia é que, no mesmo ambiente, os desenvolvedores possam criar/editar/visualizar os modelos e metamodelos, e gerar o código correspondente.

A princípio, a criação das transformações será feita fora do ambiente, sendo possivelmente integrada em um trabalho futuro.

7.1.4 Controle de Versão

Um sistema de controle de versão ou controle de revisões é parte do conhecido gerenciamento de configuração de software (SCM - *Software Configuration Management*). É uma ferramenta que ajuda no gerenciamento de múltiplas revisões do código. O SCM automatiza o armazenamento, recuperação, registro, identificação e *merge* das revisões e fornece controle de acesso. É útil para textos que são revisados com freqüência, como por exemplo softwares e documentação (**?**).

Como mencionado na Seção 3.2, um repositório de controle de versão é importante para se manter um histórico de desenvolvimento, facilitando a manutenção e evolução do sistema com base no seu passado. É importante também para compartilhar o que está sendo desenvolvido pela equipe de maneira organizada e disciplinada. É relevante destacar que o controle de versão do CoMDD irá apoiar a captura de *Design Rationale* - DR.

Para realização desta atividade, será usado a ferramenta de controle de versão Phoca. A Phoca foi desenvolvida pelo grupo e está sob a licença de código aberto Apache versão 2[2]. O diferencial dela consiste em oferecer uma API para o controle de versões de artefatos de software na forma de documentos estruturados (XML, por exemplo). Portanto, esta atividade consiste na implementação da Phoca no CoMDD.

7.1.5 Controle de Permissão de Acesso

O desenvolvimento de um sistema é comumente dividido em diversas etapas de tal forma que pessoas (ou grupos de pessoas) diferentes são responsáveis por realizarem tarefas diferentes. Por exemplo, supondo um ambiente de desenvolvimento de um sistema qualquer, que será realizado por dois grupos: o grupo A (GA) e o grupo B (GB). O GA terá que implementar a funcionalidade X e o GB a funcionalidade Y. Pode ser um requisito do projeto o GA não poder alterar[3] a implementação do GB e vice-versa. Portanto neste contexto, o controle de permissão de acesso no ambiente de desenvolvimento se torna uma funcionalidade fundamental.

Um outro ponto importante quanto à permissão de alteração de modelos é que um dos grupos não pode alterar as versões anteriores de seus próprios modelos, ou seja, se o GA criou um modelo (versão 1.0) e no dia seguinte o modificou, essa alteração passa então a ser uma outra versão do modelo (versão 1.1) de forma que a versão 1.0 ficou "inalterada". Trata-se de um controle também essencial para se manter um histórico dos modelos e assim apoiar o registro de decisões de projeto (DR). Nesse contexto, é um requisito do CoMDD fornecer permissões de acesso diferentes para diferentes pessoas.

7.1.6 Suporte ao Conhecimento das Decisões

(**?**) define *Desgin Rationale* (DR) como sendo explicações. Elas explicam as relações entre a estrutura, comportamento e as funções dos artefatos, tais como a maneira que a estrutura implementa uma função, ou como o comportamento emerge da estrutura. Eles também explicam a tomada de decisão do processo. Este segundo tipo de *rationale* é baseado no histórico das decisões de *design*, descrevendo como as alternativas foram consideradas e avaliadas e como as decisões foram tomadas ao longo do tempo.

Para (**?**), a informação armazenada no DR é muito útil em diversas maneiras como: colaboração, manutenção, aprendizagem, etc; além de trazer grandes benefícios para o reúso; na comunicação entre membros da equipe, principalmente para os mais recentes; para encontrar erros no desenvolvimento e na solução de problemas de uma maneira geral; entre outros (**?**).

[2]http://www.apache.org/licenses/LICENSE-2.0.html

[3]Alterar pode ter o significado tanto de alterar algum modelo quanto de alterar um documento em branco, ou seja, criar uma primeira versão de um modelo.

Para permitir que o projeto suporte razões de decisão, será implementado no CoMDD um trabalho já desenvolvido pelo grupo de pesquisa: a DocRat (**?**). A DocRat é uma ferramenta CASE com o intuito de registrar decisões de projeto de maneira menos intrusiva ao usuário.

O trabalho desta atividade é importar o que foi desenvolvido na DocRat, fazendo seu relacionamento com o CoMDD de forma que haja um sincronismo entre a modelagem e as anotações das razões.

7.1.7 Execução de Estudo de Caso I - Domínio de TI

Uma vez que os componentes do ambiente estejam integrados, será realizado um estudo de caso para validar o mesmo, identificar pontos fortes/fracos, e possibilidades de melhorias.

O estudo de caso I envolverá o domínio de TI, em especial os diagramas ER (Entidade-Relacionamento), conforme ilustra a Figura 7.3:

Inicialmente, será definido um metamodelo para representar tabelas, colunas e relacionamentos, como um esquema de banco de dados. As *tags* deste metamodelo serão inseridas em um texto descrevendo dados (modelo), identificando o tipo de cada elemento.

Com base nestas informações, e com o auxílio do suporte de metamodelagem, será definida uma transformação para gerar *scripts SQL* correspondentes aos modelos.

Como resultado, será possível definir um esquema de banco de dados em um modelo, e obter automaticamente seus *scripts*. Adicionalmente, os modelos de entrada ficarão disponíveis no CMS, podendo ser editados por diferentes desenvolvedores. A execução das transformações também será executada no CMS, não exigindo a instalação de nenhuma ferramenta nas estações de trabalho dos desenvolvedores.

É importante ressaltar que as maiores contribuições desse trabalho em termos de desenvolvimento são os itens 7.1.1, 7.1.2, 7.1.3, 7.1.5 e 7.1.7, pois os itens 7.1.6 e 7.1.4, como mencionado, são adaptações de trabalhos anteriores.

7.1.8 Execução de Estudo de Caso II - Domínio de SE

No caso do domínio de sistemas embarcados (SE) será definido um metamodelo para representar diagramas de transição de estados (DTE) para robô móvel autônomo (RMA). O DTE para o robô especifica que ele terá que seguir uma linha preta no chão e parar quando encontrar um obstáculo, seu estado final. Como estados, tem-se: estado final ou em movimento. Como transições, tem-se *ir para frente*, *ir para esquerda*, *ir para direita* e *parar*.

Com base nestas informações, e com o auxílio do suporte de metamodelagem, será definida uma transformação para gerar comandos para o robô, correspondentes aos modelos.

Como resultado, será possível definir um percurso a ser seguido pelo robô e obter automaticamente seus comandos para serem transmitidos ao robô. Adicionalmente, os modelos de entrada

ficarão disponíveis no CMS, podendo ser editados por diferentes desenvolvedores. A execução das transformações também será executada no CMS, não exigindo a instalação de nenhuma ferramenta nas estações de trabalho dos desenvolvedores.

A Figura 7.4 ilustra um possíve modelo a ser implementado a partir do metamodelo. O RMA começa inicialmente desligado. Ao ser ligado ele vai para o estado Parado. Caso ele detecte em todos seus sensores a presença da pista, ele vai para Frente. No estado Frente, ele pode ir para Direita ou para Esquerda, caso saia da pista pela esquerda ou pela direita, respectivamente. A partir de todos os estados (com exceção do Parado) ele pode ir para o estado Parado, caso saia da pista, ou para o Estado Final, caso encontre seu destino.

7.2 Cronograma

O cronograma de execução das atividades é apresentado na Tabela 7.1. O projeto tem duração prevista de 24 meses e a tabela segue a redefinição do número das atividades conforme abaixo:

1. Cumprimento dos Créditos Exigidos;
2. Exame de Inglês;
3. Revisão Bibliográfica;
4. Exame de Qualificação;
5. Inclusão de Suporte à Metamodelagem;
6. Suporte à Geração de Código
7. Extensão da Interface;
8. Controle de Versão e de Permissão;
9. Suporte ao Conhecimento das Razões;
10. Execução de Estudo de Caso I e II ;
11. Redação da Dissertação;
12. Redação de Artigos;
13. Defesa.

Ativ.	2009					2010						2011	
	Mar/ Abr	Mai/ Jun	Jul/ Ago	Set/ Out	Nov/ Dez	Jan/ Fev	Mar/ Abr	Mai/ Jun	Jul/ Ago	Set/ Out	Nov/ Dez	Jan/ Fev	Mar/ Abr
1	●	●	●	●	●								
2					●								
3	●	●	●	●	●	●	●	●	●	●	●		
4						●	●						
5							●	●	●	●			
6							●	●	●	●			
7							●	●	●	●			
8							●	●	●	●			
9							●	●	●	●			
10											●	●	
11											●	●	
12							●				●	●	●
13													●

Tabela 7.1: Cronograma de Atividades

7.3 Considerações Finais

Este trabalho prevê o desenvolvimento da abordagem CoMDD e, para tanto, a realização de cinco atividades: inclusão de suporte à metamodelagem; suporte à geração de código; extensão da interface; controle de versão e de permissão aos diferentes tipos de colaboradores, e suporte ao conhecimento das decisões de projetos dos modelos.

Essas atividades são planejadas para serem concluídas de acordo com o cronograma da tabela 7.1.

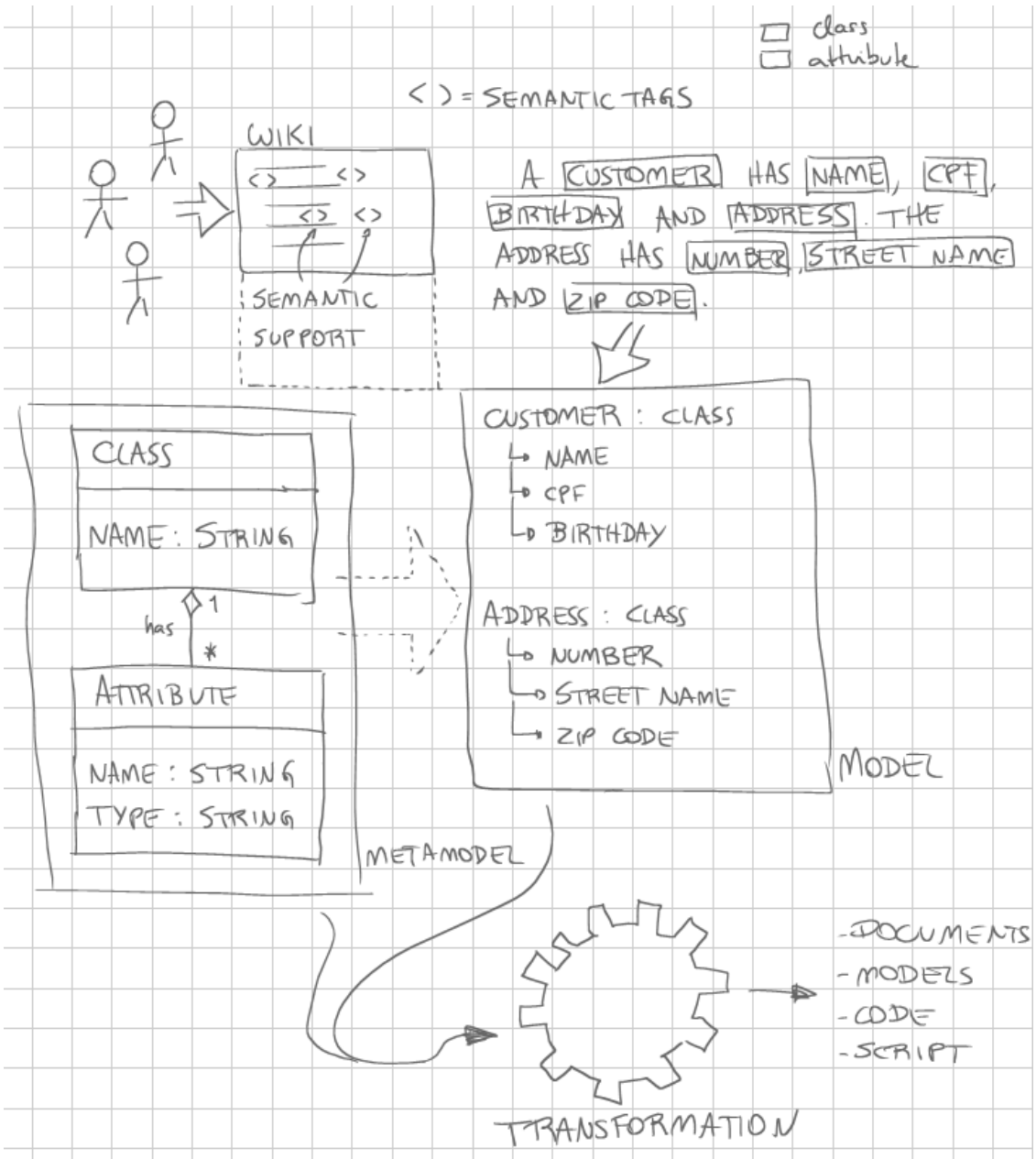

Figura 7.3: Estudo de caso do domínio de TI

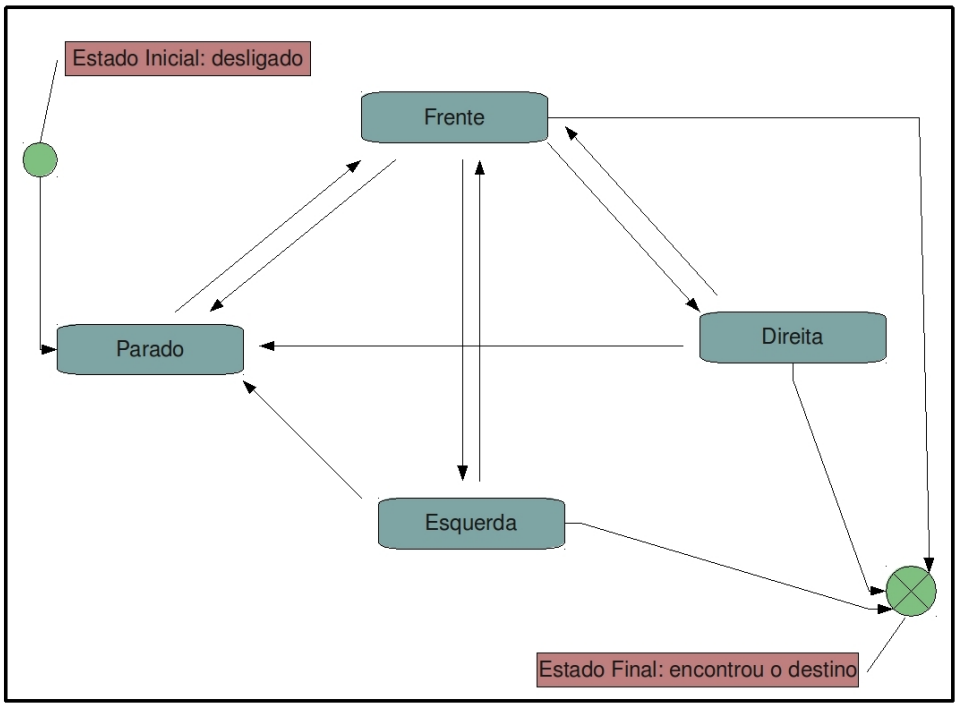

Figura 7.4: Exemplo do DTE para o Estudo de caso do domínio de Sistemas Embarcados