Uma Abordagem Flexível para Comparação de Modelos UML

Kleinner Oliveira[1], **Marcos Silva**[2], **Toacy Oliveira**[1], **Paulo Alencar**[1]

[1]School of Computer Science – University of Waterloo
Waterloo, ON – Canada, N2L 3G1

kleinner@gmail.com,{toliveira,palencar}@csg.uwaterloo.ca

[2]Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre, RS – Brasil

marcos.tadeu@pucrs.br

Abstract. *With the Model Driven Architecture (MDA), the role of model composition has become very important. One challenge in model composition is specifically to merge models expressed in the Unified Model Language (UML) and its extensions. However, to put model composition in practice it is necessary to perform an essential task: model comparison. In this paper, we present a model comparison technique that aims to insert flexibility into the equivalence definition process between input models of a model composition mechanism. Such flexibility is achieved by match strategy. Hence input models are merged if they are considered equivalent according to a specific match strategy. Such strategy is implemented by a match operator that makes use of match rules, synonym dictionary and typographic similarity. Moreover, some challenges are specified and a guidance defines the activities that should be performed throughout the model comparison process.*

Resumo. *Com o surgimento da MDA (Model Driven Architecture) o papel da composição de modelos tornou-se mais importante. Um desafio enfrentado é compor modelos representados em UML (Unified Model Language) e em suas extensões. Porém, para colocar a composição em prática é necessário realizar uma atividade essencial: a comparação de modelos. Este artigo apresenta uma técnica de comparação de modelos que visa dar flexibilidade ao processo de definição de equivalência entre os modelos de entrada de um mecanismo de composição. Esta flexibilidade é alcançada através da definição de estratégias de comparação. Conseqüentemente, modelos de entrada passam a ser compostos se considerados equivalentes de acordo com uma estratégia específica de comparação. Estas estratégias são implementadas por um operador de comparação que faz uso de regras de comparação, dicionário de sinônimo e similaridade tipográfica. Além disso, são especificados alguns desafios e proposto um guia para especificar as atividades que devem ser realizadas ao longo do processo de comparação.*

1. Introdução

Um fator crítico relacionado à dificuldade de desenvolvimento de sistemas de software complexo é a existência de um *gap* conceitual entre o domínio do problema e o domínio

da solução [France et al. 2006, France and Rumpe 2007]. As abordagens dirigidas por modelos mudam o foco do processo de desenvolvimento do código, geralmente representados em linguagens de programação de terceira geração como Java, para os modelos, especialmente modelos representados em UML (*Unified Modeling Language*) e *profiles* da UML [OMG 2007, Sendall and Kozaczynski 2003]. O objetivo é gerenciar e manipular software em nível de seus conceitos em uma tentativa de solucionar, ou minimizar, o *gap*. Com isso é possível chegar mais rapidamente ao código e tornar o processo de desenvolvimento menos difícil e custoso. Uma referência para estas abordagens é a MDA (*Model Driven Architecture*) [Object Management Group 2003], uma abordagem MDD (*Model Driven Development*) criada e mantida pela OMG (*Object Management Group*).

Dentro do contexto de um processo de desenvolvimento de software dirigido por modelos, um conjunto de modelos UML é utilizado para representar os aspectos estruturais e comportamentais do sistema de software que está sendo desenvolvido através de diferentes perspectivas. Para isto são utilizados diferentes (e ao mesmo tempo complementares) tipos de diagramas definidos na especificação da UML. Estes modelos freqüentemente representam diferentes plataformas (por exemplo, J2EE ou .NET) e domínios (por exemplo, aplicações de tempo real e financeiras) nos quais o sistema está inserido. Uma vez que estes modelos tenham sido definidos, algumas transformações são executadas com o objetivo de chegar ao código da aplicação. Tais modelos podem tanto ser utilizados horizontalmente, para descrever diferentes aspectos do sistema, como também verticalmente, sendo utilizados, desta maneira, com o objetivo de serem refinados de um nível mais alto de abstração (modelo) para um nível mais baixo de abstração (código). As abordagens MDD fazem uso de técnicas de transformação e composição de modelos para manipular e gerenciar os modelos UML em um mesmo nível de abstração ou em diferentes níveis de abstração.

A composição de modelos pode ser vista como uma operação (uma tipo especial de transformação de modelos) onde um conjunto de atividades devem ser realizadas com o objetivo de compor dois modelos de entradas M_A (*receiving*) e M_B (*merged*) produzindo M_{AB} como modelo de saída [OMG 2007, Bézivin et al. 2006]. A composição pode ser representada através da equação: M_A+$M_B \rightarrow M_{AB}$. Porém, uma importante atividade que deve ser executada antes de realizar a composição: a comparação de modelos. A comparação tem como um dos seus objetivos verificar a existência de sobreposição sintática e semântica nos modelos de entrada. A necessidade de evitar tais sobreposições surge do fato que o modelo do sistema final deve representar cada conceito unicamente a fim de evitar conflitos, ambigüidade e transformações de modelos deficientes. Por exemplo, de acordo com a especificação do metamodelo da UML [OMG 2007] não deve existir dois (ou mais) modelos com nomes iguais (por exemplo, duas classes) em uma mesma *namespace*, caso contrário isto representa que a *namespace* possui conflitos. Com isso, um mecanismo de composição de modelos deve levar em conta tais conflitos para produzir modelos de saída M_{AB} corretamente, caso contrário M_{AB} pode ter elementos com nome e valor semântico conflitantes. Em resumo, se a comparação de modelos é realizada de maneira ineficiente, pode comprometer, conseqüentemente, a atividade de composição de modelos a qual se trata de uma atividade importante em MDA.

Neste artigo é discutido o papel e a importância da comparação de modelos UML em um processo de composição. São descritos os desafios que devem ser enfrentados para

colocar em prática a comparação de modelos e é proposto um operador de composição, o qual é responsável por colocar em prática uma abordagem flexível de comparação. Esta flexibilidade é alcançada através da definição de estratégias e regras de comparação. Além disso, foi elaborado um guia para a comparação de modelos a fim de definir as atividades necessárias para realizar tal comparação e o fluxo de execução entre elas.

1.1. Motivação

Este trabalho foi motivado pela necessidade de desenvolver um mecanismo de composição de UML *profiles*. Portanto, a abordagem será discutida dentro do contexto de comparação de *profiles*. A razão de escolher UML *profiles* é devido ao papel central que os mesmos desempenham em MDA. Sendo assim, será apresentado um exemplo de motivação que consiste na composição de dois *profiles Tree* (*receiving model*) e *Topology* (*merged model*) [Fernández and Moreno 2004] os quais são mostrados na Figura 1, onde cada *profile* representa uma linguagem de modelagem específica de domínio (DSML). *Tree* representa uma estrutura de dados hierárquica usada por aplicações em ciência da computação. Enquanto que *Topology* representa as conexões entre os elementos de dois sistemas de informação com uma topologia de rede estrela.

Em *Topology* tem-se nodos (representados pelo estereótipo *Node*) conectados por *links* que podem ser locais (*LocalEdge*), se eles conectam um nodo com o elemento central da estrela, ou remoto (*Edge*), se eles conectam nodos centrais entre si [Fernández and Moreno 2004]. Cada nodo é identificado por sua posição (*location*) e cada nodo central tem um tipo de estado (*state*) que define sua disponibilidade (seus valores são definidos pela enumeração *StateKind*). Um nodo final (*EndNode*) é também identificado por sua posição (*position*). O *Tree* tem nodos (representados pelo estereótipo *Node*) conectados por *links* (*Edge*). Possui um nodo que representa o nodo final (*Leaf*) da hierarquia e apresenta também um nodo raiz (*Root*) que possui um tipo de estado (*state*) para definir a sua disponibilidade (seus valores definidos pela enumeração *StateKind*). Cada *Node* é determinado pelo par: nome (*name*) e valor (*value*). Além disso, é possível realizar operações de busca na hierarquia (*Search*).

Antes de compor *Tree* e *Topology* é necessário comparar os elementos destes *profiles* a fim de determinar suas equivalências. Para fazer isto é necessário identificar as correspondências existentes entre os elementos dos *profiles* de uma maneira coerente. Por exemplo, apesar de os estereótipos *Tree.Leaf* e *Topology.EndNode* terem diferentes nomes, seria possível os mesmos representarem conceitos de domínio com mesmo valor semântico?

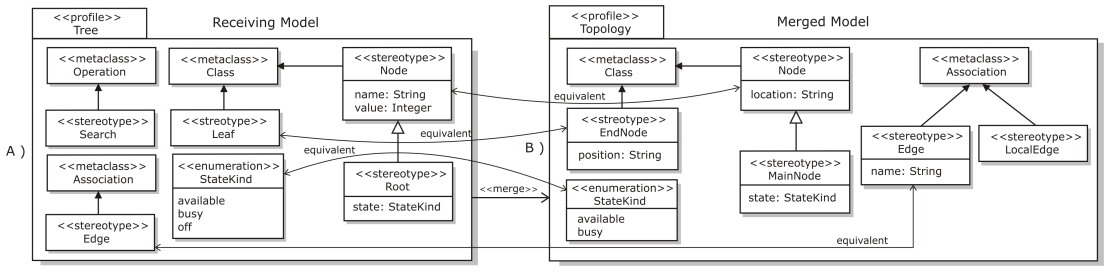

Figure 1. Exemplo de comparação de UML *profiles*

1.2. Contribuições do Artigo

A contribuição geral deste artigo consiste na definição de uma abordagem flexível de comparação modelos UML baseada em estratégias de comparação. Porém, para colocar a comparação de modelos em prática implica em responder algumas questões que são intrínsecas a este tipo de operação. Como discutido em [Nejati et al. 2007], quais critérios são necessários para identificar correspondência entre modelos diferentes? Como estes critérios podem ser quantificados? Um vez definidos dois modelos de entrada, o mecanismo de comparação deve ser capaz de produzir "apenas" um único possível resultado que represente a correspondência entre os elementos dos modelos de entrada? Como questões sintáticas e semânticas podem ser utilizadas? Quais propriedades dos modelos de entrada devem ser consideradas na comparação? O que deve ser utilizado e quais atividades devem ser executadas para colocar a comparação de modelos em prática? O conjunto de respostas para estes questões representa as contribuições específicas deste artigo.

Além disso, a definição de um operador de composição, o guia de comparação de modelos e a mudança da visão de comparação de modelos de uma forma não-flexível para uma forma flexível representam também contribuições deste artigo. Para implementar as estratégias, o operador de comparação faz uso de um conjunto de heurísticas relacionadas à definição de similaridade tipográfica e à especificação da equivalência entre os valores semânticos dos elementos dos modelos. Para isto, o operador usa o conceito de assinatura de modelos definido na especificação metamodelo da UML. Nossa abordagem foi implementada como parte de um mecanismo de composição de UML *profiles* [Oliveira 2008, Oliveira and Oliveira 2007b, Oliveira and Oliveira 2007a] e tem mostrado ser uma forma eficiente e flexível para especificar correspondência entre *profiles*. Além disso, a abordagem foi especificada usando a linguagem de especificação formal *Alloy* [Jackson 2002, Jackson 2006] que é baseada em lógica de primeira ordem e teoria dos conjuntos. Além disso, a ferramenta de suporte da linguagem *Alloy*, o *Alloy Analyzer* [Jackson 2006] foi utilizado com o objetivo de realizar uma verificação formal e fazer uma análise automática da abordagem proposta.

O artigo é estruturado como segue. Na Seção 2 é apresentada a fundamentação teórica e os principais desafios que pesquisadores enfrentam quando tentam realizar comparação de modelos. Na Seção 3, as estratégias de comparação e o operador de comparação são descritos. Na Seção 4, o guia de comparação é discutido. Na Seção 5 são discutidos os trabalhos relacionados. Finalmente, na Seção 6 são mostradas algumas conclusões e a descrição dos trabalhos futuros.

2. Fundamentação Teórica e Desafios

Comparação de modelos surge como uma atividade essencial para colocar a composição em prática e pode ser vista como uma operação genérica que varia de aplicação para aplicação, na qual os elementos de M_A e M_B são comparados de diferentes formas dependendo do tipo da aplicação e dos modelos que são considerados. Por exemplo, a comparação de duas *state chart* [Nejati et al. 2007] e a comparação de duas versões de diagramas de classe UML representam dois tipos de comparação de modelos diferentes devido às particularidades de cada modelo. Com isso, diferentes propriedades devem ser consideradas no momento da comparação.

Na especificação da UML [OMG 2007] são apresentadas algumas formas de estender o seu metamodelo com o objetivo de adaptá-la a determinados domínios ou plataformas. Os *profiles* representam uma destas formas, os quais consistem de um mecanismo de extensão conservativa do metamodelo da UML. Com os *profiles* é possível, por exemplo, adicionar semântica a um elemento do metamodelo da UML que não tem sido definida, atribuir uma terminologia mais adequada a um elemento presente em uma determinada plataforma ou domínio, ou adicionar informações que podem ser usadas quando um modelo participa de um processo de transformação.

Porém, o mecanismo de composição definido na especificação da UML não é capaz de compor ou comparar modelos de entrada (M_A e M_B) corretamente [Oliveira 2008, Rumbaugh et al. 2005, Zito 2006]. Conseqüentemente, algumas questões de pesquisa surgem: como comparar dois *profiles*? Quais atividades devem ser realizadas para comparar M_A e M_B? Uma vez que o metamodelo da UML é estendido e é atribuído semântica a alguns do seus elementos, como é possível comparar estes modelos de uma forma flexível? Para o nosso conhecimento, a necessidade de comparar modelos de uma forma flexível não tem sido trabalhada nem proposta pelas abordagens atuais de comparação de modelos, as quais geralmente estão associadas a um mecanismo de composição de modelos. Desse modo, tal fato mostra o caráter pioneiro deste trabalho. Baseado em trabalhos anteriores [Oliveira 2008, Oliveira and Oliveira 2007b, Oliveira and Oliveira 2007a] e em abordagens relacionadas ao tema (descritas na Seção 5), foi possível concluir que os principais desafios que pesquisadores e modeladores enfrentam para colocar em prática a comparação de modelos no contexto de MDD podem ser agrupados nas seguintes categorias:

- ***Desafio de comparar modelos específicos de domínio***: alguns desafios surgem a partir do interesse em construir e manipular modelos específicos de domínio gerados a partir de uma linguagem de modelagem específica de domínio usada em um processo MDD. Por exemplo, é possível estender a UML usando *profiles* para definir um variante da UML associando uma particular semântica para os "pontos de variação semântica" definidos no seu metamodelo [OMG 2007, France and Rumpe 2007]. Conseqüentemente, um desafio seria como desenvolver um suporte para adaptar as técnicas de comparação de modelos frente às semânticas adicionadas aos pontos de variação semântica e frente as especializações do metamodelo da UML realizadas através de *profiles*.
- ***Desafio de Nível de Abstração***: dentro de um processo MDD, os modelos são representados e manipulados (transformação e composição) em diferentes níveis de abstração. Por exemplo, os três níveis conceituais definidos em MDA: CIM (*Computer Independent Model*), PIM (*Platform Independent Model*) e PSM (*Platform Specific Model*). Sendo assim, como comparar modelos representados em diferentes níveis de abstração e como adaptar/evoluir a técnica de comparação dentro deste contexto? Este desafio indica a concentração de esforços na investigação sobre o entendimento e evolução das técnicas de comparação de modelos em relação à manipulação de modelos representados em diferentes níveis de abstração e em diferentes linguagens de modelagem, onde cada linguagem tem suas particularidades.
- ***Desafio semântico e de Propriedades***: todo modelo tem associado a ele algum valor semântico, considerando dois modelos (por exemplo, duas classes) com no-

mes iguais dentro de uma mesma *namespace* (por exemplo, um pacote) poderiam, desse modo, formar uma equivalência. Porém, o que deve ser realizado se os modelos possuem diferentes valores semânticos ou propriedades diferentes? Por exemplo, duas classes Professor (*isAbstract* = true) e Professor (*isAbstract* = false), embora tenham nomes iguais (podendo ser consideradas equivalentes pela definição apresentada no mecanismo de composição da UML) apresentam uma discordância entre os valores de suas propriedades.

3. Comparação de Modelos Baseada em Estratégias de Comparação

Uma vez mostrado o exemplo de motivação e definido os desafios em comparação de modelos, será apresentada a abordagem flexível de comparação de modelos baseada em estratégias (*match strategy*). Foram especificadas três estratégias: *default, partial* e *complete match strategy* podendo novas estratégias serem definidas e inseridas em nossa abordagem. Um operador de comparação foi definido o qual é o responsável, de fato, por colocar em prática as estratégias. A partir dos modelos de entrada (M_A e M_B) e da definição do tipo de estratégia de comparação o operador define um grau de equivalência entre os modelos de entrada e, de acordo com um limiar, é determinado os modelos que são equivalentes.

3.1. Operador de Comparação

O operador consiste de uma heurística e seu objetivo é definir a equivalência entre os modelos de entradas de acordo com uma estratégia de comparação. Para isto, ele faz uso de dicionário de sinônimo, similaridade tipográfica e assinatura de modelos a fim de definir um grau de equivalência ($\mathcal{S}$), onde $0 \leq S \leq 1$. Com o dicionário de sinônimo é possível fazer um mapeamento entre os conceitos do domínio que possuem valor semântico iguais. O dicionário de sinônimos possibilita ao especialista do domínio aplicar suas *expertise* ao processo de definição de equivalência dos modelos, uma vez que ele define quais termos/conceitos do domínio são considerados sinônimos. Tal associação (especialista $\Leftrightarrow$ dicionário) permite melhorar o resultado da comparação.

O operador compara os elementos de *receiving model* (r) com os de *merged model* (m) (ver Figura 1) e define o valor de $\mathcal{D}(r,m) \rightarrow$[0,1], o qual representa o grau de similaridade entre os elementos. $\mathcal{D}$ é igual a 1 (zero) se *r* e *m* são sinônimos, caso contrário ele retorna 0. $\mathcal{D}$ é calculado para todo par de elementos de *r* e *m*. No estágio inicial da comparação, o operador assume que todo par (*r, m*) não são sinônimos, então $\mathcal{D}(r,m)$ = 0 para todo par (*r, m*). Por exemplo, de acordo com o dicionário de sinônimo mostrado na Tabela 1, os estereótipo *Tree.Leaf* e *Topology.EndNode*, mostrados na Figura 1(a), representam conceitos equivalentes, portanto $\mathcal{D}$(*Leaf,EndNode*) = 1.

O objetivo da similaridade tipográfica é determinar $\mathcal{T}(r,m) \rightarrow$[0..1] para todo pares possíveis de *r* e *m*. Primeiro é definido todos os pares de elementos através do produto cartesiano em (R$\times$M), onde R e M são o conjunto de elementos do *r* e *m*, respectivamente. A definição do grau de similaridade entre os elementos dos *profiles Tree* e *Topology* é mostrada na Tabela 2. Para fazer isto, é utilizado o algoritmo *N-Gram* [Manning and Shütze 1999] para atribuir um valor em [0..1] que representa grau de similaridade do par de elementos. O algoritmo *N-Gram* especifica um grau de similaridade para um dado par de *strings* baseado na contagem do número de *substring* de tamanho N que são idênticas (foi utilizado N = 2).

O operador usa o conceito de assinatura que é definido na especificação do metamodelo da UML. Tal assinatura é determinada em termos das propriedades sintáticas dos elementos dos modelos, onde as propriedades sintáticas são utilizadas para definir a estrutura destes modelos. A assinatura é uma coleção de valores para um conjunto (ou subconjunto) de propriedades sintáticas definidas em uma metaclasse dentro do metamodelo da UML. Por exemplo, *isAbstract* é uma propriedade sintática definida em uma metaclasse do metamodelo da UML chamada de *Class*, que é utilizada para representar o conceito de classe definido no paradigma de programação orientada a objetos. Se uma instância de uma classe é uma classe abstrata, então *isAbstract* = *true*. Caso contrário, a instância é um classe concreta, *isAbstract* = *false*.

Table 1. Exemplo de dicionário de sinônimos.

Nome	Sinônimo
Leaf	EndNode, FinalNode
Edge	Border, Limit, Margin
Search	Research, Searching, Query

O conjunto de propriedades sintáticas utilizado para determinar a assinatura de um elemento de modelo é chamada de *signature type*, como definido em [Reddy et al. 2006]. A assinatura que consiste de todas as propriedades sintáticas associadas a um elemento de um modelo é chamada de *complete signature type*. Por outro lado, a baseada em um subconjunto de propriedades sintáticas é chamada de *partial signature type* e a assinatura baseada apenas no nome é chamada de *default signature type*. As assinaturas podem ser estruturadas em níveis de comparação organizados de forma hierárquica. Por exemplo, na Figura 1, uma possível definição de níveis para o estereótipo *Tree.Node* seria: *Tree.Node* (nome) (nível 2), com *Tree.Node.name* e *Tree.Node.value* (como *tagged values*) (nível 1). Desse modo, cada elemento de um modelo tem uma assinatura associada a ele, a qual será utilizada durante o processo de comparação. O grau de similaridade entre *r* e *m* baseado na definição das assinaturas é representado por $\mathcal{M}$(r,m) e é definido através de uma média ponderada das médias aritméticas geradas pelos níveis. $\mathcal{M}$ é calculado seguindo a fórmula ilustrada na Equação 1.

$$\mathcal{M} = \frac{\sum_{i=1}^{n} p_i \cdot \left[\sum_{j=1}^{k} \frac{\varphi_{i,j}}{k} \right]}{\sum_{i=1}^{n} p_i} \rightarrow [0..1] \tag{1}$$

- n é o número de níveis definidos para comparar os elementos, onde $n \geq 1$ e $n \in \mathbb{N}_+^*$.
- p_i representa o peso, sendo $p_i = i$, onde $i \geq 1$ e $i \in \mathbb{N}_+^*$; k expressa o número de elementos em cada nível, onde $k \geq 1$ e $k \in \mathbb{N}_+^*$ (por exemplo, *Tree.Node* tem duas propriedades e como estas propriedades representam um nível, então $k = 2$).

- $\varphi_{i,j}$ (i e j representam o nível e o item do elemento do modelo que está sendo comparado, respectivamente) é usado para denotar se um item de um elemento de *r* (por exemplo, *name:String* em *Tree.Node*) é equivalente a outro item de elemento de *m*. Ele é uma variável booleana e seu valor é definido através de regras de comparação que serão descritas a seguir. Estas regras comparam ítens dos elementos do modelo retornando 1 (um), se a regra é satisfeita, caso contrário retorna 0 (zero). Por exemplo, quando os estereótipos *Tree.Root* e *Topology.MainNode* são comparados $\varphi_{2,1}$ = 0 (aplicando a regra de comparação MR1) e $\varphi_{1,1}$ = 1 (aplicando a regra de comparação MR3).

Para calcular o grau de similaridade dos elementos dos modelos ($\mathcal{S}$) é necessário levar em conta $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{M}$ a fim de combinar os valores gerados por eles. Desse modo, S é definido através da fórmula mostrada na Equação 2. Se $\mathcal{D}$ = 1, então $\mathcal{T}$ também assume valor igual a 1 e vice-versa.

$$\mathcal{S} = \frac{(\mathcal{D} + \mathcal{T} + \mathcal{M})}{\mathcal{D} + 2} \rightarrow [0..1] \tag{2}$$

Com a Equação 2 é calculado o grau de similaridade de todos os elementos do *profile Tree* em relação aos elementos do *profile Topology*. O resultado da comparação é mostrado na Tabela 2. Para produzir uma relação de correspondência entre os dois modelos, é necessário especificar um limiar (t) que é utilizado como ponto de corte. Para o exemplo discutido neste artigo é utilizado t = 0.7. Sendo assim, todos os pares que possuem grau de similaridade acima do limiar serão considerados equivalentes. Se $\mathcal{S}(r,m) > t$, então *r* e *m* são considerados equivalentes. Dentro de um processo de composição de modelos, tais elementos equivalentes seriam compostos com o objetivo de obter um elemento único, M_A+$M_B \rightarrow M_{AB}$. Na Tabela 2 são destacados os elementos que apresentam o grau de similaridade acima do limiar t = 0.7, podendo tais elementos serem agrupados ao pares, como segue: (*Tree.Node, Topology.Node*), (*Tree.Edge, Topology.Edge*), (*Tree.Leaf, Topology.EndNode*) e (*Tree.StateKind, Topology.StateKind*).

		Topology Profile					
		Node	MainNode	Edge	LocalEdge	EndNode	StateKind
Tree Profile	Node	0,83	0,22	0	0,08	0	0
	Root	0	0,05	0	0	0	0
	Edge	0	0	1	0,22	0	0
	Search	0	0	0	0	0	0
	Leaf	0	0	0	0	1	0
	StateKind	0	0	0	0	0	0,96

grau de similaridade acima do limiar (t = 0.7)

Table 2. Similaridade entre os elementos dos *profiles*.

3.2. Regras de Comparação e Estratégias de Comparação

As regras de comparação são utilizadas para determinar se dois modelos (ou elementos de modelos) são equivalentes. O operador de comparação é o responsável por executar tais regras dentro do processo de comparação tendo como objetivo definir o valor de $\varphi_{i,j}$. Para todo elemento de um modelo é necessário uma regra de comparação a fim de que operador possa realizar a comparação entre eles. O operador de comparação aplica

as regras de comparação nos modelos para realizar a comparação baseada na assinatura dos mesmos. Se a regra falhar, então os modelos não são equivalentes ($\varphi_{i,j}$ = 0). Caso contrário, os modelos são equivalentes ($\varphi_{i,j}$ = 1). As regras verificam se os valores das propriedades dos elementos definidas no metamodelo da UML são iguais ou não.

Um requisito importante para realizar a comparação é permitir que os modelos sejam comparados de forma flexível [France and Rumpe 2007, Oliveira 2008], pois este ganho de flexibilidade pode melhorar o processo de composição de modelos. Com isso em mente, foram desenvolvidas *estratégias de comparação* de modelos cuja sua definição é baseada na especificação dos tipos de assinaturas de modelos (discutidas anteriormente) e em regras de comparação. Comparar modelos seguindo uma estratégia de comparação significa que um determinado número de propriedades dos modelos serão consideradas durante o processo de comparação garantindo, desse modo, uma flexibilidade. Foram definidos três tipos de estratégias de comparação os quais são diretamente ligados ao tipo de assinatura de modelos, são elas: *default, partial* e *complete*. Por exemplo, comparar duas classes A e B seguindo a estratégia de comparação *complete*, significa que o operador de comparação fará uso de um conjunto de regras de comparação que foram criadas de acordo com a *complete signature type*, ou seja, todas as propriedades dos elementos dos modelos serão consideradas durante a comparação.

Para implementar os três tipos de estratégias são necessários três tipos de regras de comparação (em conformidade aos tipos de assinatura) que se diferenciam pelo tipo de assinatura utilizado, são elas: (i) *default* trata-se de um conjunto de regra que compara os modelos levando em conta apenas o nome dos modelos (usam a especificação da *default signature type*); (ii) *partial* trata-se de um conjunto de regras que leva em consideração um determinado numero de propriedades sintáticas do modelo (usam a especificação da *partial signature type*); (iii) *complete* trata-se de um conjunto de regras que leva em consideração todas as propriedades sintáticas dos modelos (usam a especificação da *complete signature type*). Por exemplo, operador de comparação faz uso da *default match strategy* para produzir a tabela mostrada na Tabela 2. Algumas regras de comparação *default* são descritas brevemente a seguir:

MR1. Stereotype match rule:
MatchStereotype(Stereotype rcv, Stereotype mrgd) → (rcv.name = mrgd.name) AND
MatchAttribute(rcv, mrgd) AND MatchOperation(rcv, mrgd)

MR2. Association match rule:
MatchAssociation(Association rcv, Association mrgd) →
(rcv.name = mrgd.name) AND (rcv.memberEnds = mrgd.memberEnds)

MR3. Attribute match rule:
MatchAttribute(Stereotype rcv, Stereotype mrgd) →
(rcv.ownedAttribute.name = mrgd.ownedAttribute.name) AND
(rcv.ownedAttribute.TypedElement = mrgd.ownedAttribute.TypedElement)

MR4. Operation match rule:
MatchOperation(Stereotype rcv, Stereotype mrgd) →
(rcv.ownedOperation.name = mrgd.ownedOperation.name) AND
(rcv.ownedOperation.ownedParameter.length =
mrgd.ownedOperation.ownedParameter.length) AND
(∀x(rcv.ownedOperation.ownedParameter[x] =

mrgd.ownedOperation.ownedParameter[x])

MR5. Enumeration match rule:
MatchEnumeration(Enumeration rcv, Enumeration mrgd) →
(rcv.name = mrgd.name) AND
MatchEnumerationLiteral(Enumeration rcv, Enumeration mrgd)

MR6. Enumeration Literal match rule:
MatchEnumerationLiteral(Enumeration rcv, Enumeration mrgd) →
∀x(rcv.ownedLiteral.name[x] = mrgd.ownedOperation.name[x])

4. Um Guia Para Comparação de Modelos

De acordo com [France and Rumpe 2007, Oliveira 2008] não existe um consentimento sobre os requisitos, as atividades e os passos que devem ser seguidos para colocar a comparação de modelos em prática, e nem mesmo boas práticas para evitar erros durante a elaboração de um mecanismo de comparação. Alguns trabalhos (por exemplo, [Kolovos et al. 2006b, Ohst et al. 2003]) foram propostos para atacar os problemas encontrados em comparação de modelos, porém nenhum deles, até agora, foi definido como padrão ou referência. Em [OMG 2007], o mecanismo de comparação de modelos da UML não apresenta um fluxo de atividades para ajudar na especificação de comparação, não apresenta uma boa documentação, e muito menos descreve como a comparação pode ser executada de forma flexível.

Nas seções anteriores foram identificadas e delegadas atividades ao operador de comparação. Esta seção tem o foco em ordenar e fornecer um fluxo de como tais atividades devem ser executadas. Este fluxo pode tanto ser utilizado como um guia para auxiliar na comparação de modelos quanto aplicado como boas práticas. Além disso, ele pretende tornar mais compreensível o papel do operador dentro do processo de comparação a fim de facilitar o entendimento das atividades e da dinâmica entre elas. O guia de comparação de modelos é mostrado na Figura 2. O guia é organizado em três fases: inicial, comparação e composição. A *fase inicial* é iniciada quando o operador recebe os modelos de entrada. Então, o operador analisa os modelos de entrada com o objetivo de conhecer os seus tipos (por exemplo, estereótipo, associação, classe, e etc) separando e agrupando os elementos de acordo com os seus tipos. Por exemplo, os estereótipos (*Tree.Node* e *Topology.Node*) e as associações (*Tree.Node* e *Topology.Node*) são identificadas e agrupadas de acordo com os seus tipos.

O objetivo da fase de comparação é definir quais modelos são equivalentes. Inicialmente são definidas as assinaturas para cada tipo de elemento de modelo. O próximo passo é especificar a estratégia de comparação que determinará como a comparação será realizada. O operador define o grau se similaridade para todos os elementos dos modelos de entrada e, baseado em um limiar, define os elementos que são equivalentes e uma descrição de equivalência entre eles. A fase é finalizada assim que o operador determina os elementos que não são equivalentes e os que são equivalentes, e determina uma descrição de equivalência entre eles. O próximo passo é realizar a fase de composição dos modelos que se concentra em compor os elementos equivalentes, porém esta fase está fora do escopo deste trabalho e foi colocada apenas com o caráter ilustrativa para representar que a abordagem é integrada a um mecanismo de composição.

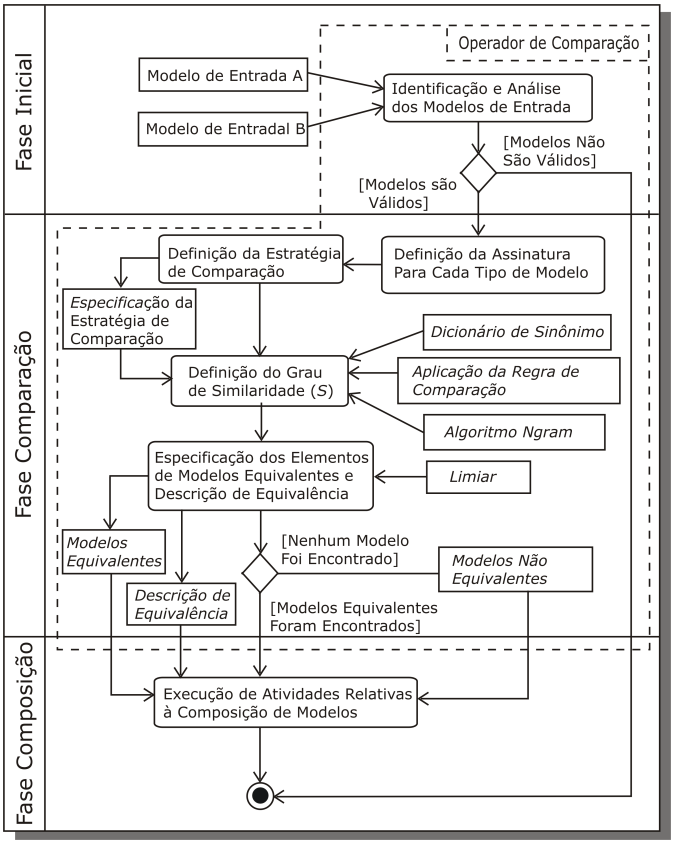

Figure 2. Um guia para comparação de modelos

5. Trabalhos Relacionados

A comparação de modelos é aplicada em diferentes domínios e plataformas, e desempenha um papel central em algumas aplicações, tais como: composição de modelos, integração e evolução de esquema, composição de código fonte, evolução de aplicações, integração de banco de dados, definição de diferença entre documentos XML, e diferença entre versões de diagramas UML. Pesquisas realizadas até o momento têm proposto algumas técnicas para atacar problemas relacionados à comparação e têm alcançado certo grau de automação nas atividades em específicos domínios de aplicação.

Sendo assim, é apresentado um *overview* das principais abordagens que são relacionadas aos objetivos apresentados neste artigo. Algumas abordagens que fazem uso de comparação como elemento chave para um mecanismo de composição também serão discutidas. Para fazer isto, o foco de cada abordagem é descrito brevemente, seguido por uma descrição das similaridades e diferenças em relação à abordagem proposta neste artigo. A Figura 3 mostra esta comparação.

Model Composition Semantics. S. Clarke introduz semântica de composição ao metamodelo da UML [Clarke 2001, Clarke and Walker 2001]. A abordagem define um novo elemento, *composition relationship*, que permite especificar como dois modelos UML podem ser compostos. Com este elemento é possível: (i) identificar e especificar conceitos que se sobrepõem ou não (comparação); (ii) especificar como modelos devem ser compostos, e como conflitos são conciliados (composição). Porém, a abordagem não

apresenta flexibilidade na sua forma de comparação e compara os modelos apenas baseado no nome, o que representa um ponto fraco da abordagem.

Model Composition Directives. Reddy et al. [Reddy et al. 2006] apresenta uma técnica de composição de modelos baseada em assinatura de modelos, na qual elementos de modelos são compostos se suas assinaturas são equivalentes (apenas o nome). Porém, em nossa abordagem, o operador faz uso de dicionário de sinônimo, similaridade tipográfica e estratégias de comparação para definir o grau de similaridade entre elementos dos modelos de entrada.

		Critérios de Avaliação							
		Guia	Estratégia	Tipografia	Name	Estrutural	Semântica	Regras	Operador
Abordagens	Model Composition Semantics				X				
	Model Composition Directives				X	X			
	Package Merge				X			X	
	Epsilon Merging Language					X			
	Difference between Models				X	X			

Legenda:
X – suportada pela abordagem
☐ – não suportado pela abordagem

Figure 3. Comparação entre os trabalhos relacionados

Package Merge. Trata-se do mecanismo de composição da UML [OMG 2007] o qual é baseado em regras de comparação, restrições e transformações (as regras de composição). Sua principal função dentro do metamodelo da UML é permitir a implementação de níveis de complacência. A princípio, suas regras de comparação são similares às usadas pelo nosso operador de comparação, porém suas regras são representadas em linguagem natural e consideram apenas o nome dos elementos dos modelos para comparar. Além disso, de acordo com [Rumbaugh et al. 2005, Zito 2006, Zito et al. 2006] a definição do *Package Merge* é inconsistente, incompleta e ambígua.

Epsilon Merging Language. EML [Kolovos et al. 2006a] é uma linguagem para realizar composição de modelos e possui uma linguagem de comparação e transformação de modelos como subconjunto. Nesta abordagem, a comparação é apenas baseada em critérios sintáticos, ao contrário da nossa abordagem que considera tanto as propriedades sintática e semântica, trabalhando com ambas de uma forma integrada e flexível.

Difference between Models. Esta abordagem apresenta um mecanismo de como detectar e visualizar diferença entre versões de modelos UML. Uma vez que diagramas de entrada são especificados, a abordagem produz um diagrama unificado que contêm as partes que são comuns e particulares de ambos os diagramas de entrada [Ohst et al. 2003]. Enquanto a nossa abordagem lida com um certo número de problemas relacionados à comparação levando em consideração critérios sintáticos e semânticos de uma forma flexível, este abordagem é concentrada na manipulação e comparação de modelos de um mesmo domínio e com mesmo valor semântico sem apresentar flexibilidade.

6. Conclusões e Trabalhos Futuros

Ao longo do artigo foi discutida a importância da comparação de modelos para atividades relacionadas à composição de modelos e foram descritos alguns dos seus problemas e desafios envolvidos com a sua implementação. Tais desafios devem encorajar pesquisadores a realizarem trabalhos com foco em solucionar tais lacunas a fim de que novas gerações de aplicações possam fazer uso de melhores técnicas de comparação.

A principal contribuição do artigo é concentrada na definição de uma abordagem que fornece uma forma flexível de realizar comparação de modelos fazendo uso de estratégia de comparação, a qual é implementada por um operador de comparação. Além disso, foi considerado e discutido que o conjunto de diferentes formas de comparação melhora o processo de comparação. Foi proposto também um guia de comparação com o objetivo de torna a realização da comparação mais compreensível e clara, ajudando no seu entendimento e evolução.

A abordagem proposta apresenta algumas limitações que serão investigadas em trabalhos futuros. Por exemplo, quando dois modelos são definidos é possível associar restrições semânticas aos mesmos. Tais restrições devem ser consideradas e respeitadas durante a composição a fim de que a semântica especificada com as restrições não sejam desrespeitadas. Entretanto, no momento, a abordagem não é capaz de comparar tais restrições o que caracteriza um problema a ser investigado em trabalhos futuros. Pretende-se melhorar e acrescentar novas funcionalidades ao operador, especificar novas estratégias de comparação e melhorar as regras de comparação. Além disso, outro trabalho a ser realizado será usar ontologias para melhorar a comparação do valor semântico dos modelos. Por fim, esforços e melhorias na área de comparação de modelos são necessárias a fim de evoluir a engenharia de modelos permitindo a se torne uma realidade na indústria.

References

Bézivin, J., Bouzitouna, S., Fabro, M., Gervais, M. P., Jouault, F., and D.Kolovos (2006). A Canonical Scheme for Model Composition. In *ECMDA-FA'06*, pages 346–360, Bilbao, Spain.

Clarke, S. (2001). *Composition of Object-Oriented Software Design Models*. PhD thesis, School of Computer Applications, Dublin City University, Dublin, Irland.

Clarke, S. and Walker, R. (2001). Composition Patterns: an Approach to Designing Reusable Aspects. In *Tthe 23rd ICSE'01*, pages 5–14, Toronto, Ontario, Canada.

Fernández, L. and Moreno, A. (2004). An Introduction to UML Profiles. In *The European Journal for the Informatics Professional*, volume 5, pages 6–13.

France, R., Ghosh, S., and Dinh Trong, T. (2006). Model Driven Development Using UML 2.0: Promises and Pitfalls. *IEEE Computer Society*, 39(2):59–66.

France, R. and Rumpe, B. (2007). Model-Driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering (FOSE'07) co-located with ICSE'07*, pages 37–54, Minnesota, EUA.

Jackson, D. (2002). Alloy: a Lightweight Object Modelling Notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290.

Jackson, D. (2006). *Software Abstractions: Logic, Language and Analysis*. The MIT Press, USA.

Kolovos, D., Paige, R., and Polack, F. (2006a). Merging Models with the Epsilon Merging Language (eml). In *ACM/IEEE 9th MODELS'06*, Genova, Italy. ACM Press.

Kolovos, D., Paige, R., and Polack, F. (2006b). Model Comparison: a Foundation for Model Composition and Model Transformation Testing. In *International Workshop on Global Integrated Model Management*, pages 13–20, New York, NY, USA. ACM Press.

Manning, C. and Shütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, USA.

Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., and Zave, P. (2007). Matching and Merging of Statecharts Specifications. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, pages 54–64, Minnesota, EUA.

Object Management Group (2003). *MDA Guide Version 1.0.1*. http://www.omg.org/docs/omg/ 03-06-01.pdf.

Ohst, D., Welle, M., and Kelter, U. (2003). Differences between Versions of UML Diagrams. In *9th European Software Engineering Conference*, pages 227–236. ACM Press.

Oliveira, K. (2008). Composição de UML Profiles. Master's thesis, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil.

Oliveira, K. and Oliveira, T. (2007a). A Guidance for Model Composition. In *International Conference on Software Engineering Advances (ICSEA'07)*, pages 27–32, France. IEEE Computer Society.

Oliveira, K. and Oliveira, T. (2007b). Composição de UML Profiles. In *Workshp de Tese e Dissertações em Engenharia de Software SBES'07*, pages 17–23, João Pessoa, PB.

OMG (2007). *Unified Modeling Language: Infrastructure version 2.1*. Object Management Group. http://www.omg.org/docs/formal/07-11-04.pdf.

Reddy, Y., France, R., Straw, G., J. Bieman, N. M., Song, E., and Georg, G. (2006). Directives for Composing Aspect-Oriented Design Class Models. *Transactions of Aspect-Oriented Software Development*, 1(1):75–105.

Rumbaugh, J., Jacobson, I., and Booch, G. (2005). *The Unified Modeling Language Reference Manual*. Object Technology Series, Addison-Wesley, USA, Second edition.

Sendall, S. and Kozaczynski, W. (2003). Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45.

Zito, A. (2006). UML's Package Extension Mechanism: Taking a Closer Look at Package Merge. Master's thesis, School of Computing, Quenn's University Kingston, Ontario, Canada.

Zito, A., Diskin, Z., and Dingel, J. (2006). Package Merge in UML 2: Practice vs. Theory? In *Proceedings 7th MODELS'04*, pages 185–199. Springer Berlin.