UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GUILHERME ERMEL

UMA FERRAMENTA PARA COMPOSIÇÃO DE DIAGRAMAS DE COMPONENTES DA UML

São Leopoldo
2014

GUILHERME ERMEL

UMA FERRAMENTA PARA COMPOSIÇÃO DE DIAGRAMAS DE COMPONENTES DA UML

Artigo apresentado como requisito parcial para obtenção do título de Tecnólogo em Análise e desenvolvimento de sistemas, pelo curso de Análise e desenvolvimento de Sistemas da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Prof. Kleinner Silva Farias de Oliveira

São Leopoldo

2014

UMA FERRAMENTA PARA COMPOSIÇÃO DE DIAGRAMAS DE COMPONENTES DA UML

Guilherme Ermel *

Resumo: Composição de modelos de software desempenha um papel fundamental em muitas atividades de Engenharia de Software como, por exemplo, evoluindo modelos de arquitetura de software ou reconciliando modelos criados em paralelo. Por este motivo, várias técnicas têm sido propostas nos últimos anos na academia e na indústria, tais como IBM RSA e Epsilon. Apesar disso, vários estudos na literatura apontam que estas técnicas são imprecisas para integrar diagramas de componentes da UML, visto que inconsistências poderão ser inseridas no modelo composto produzido. Este trabalho, portanto, propõe uma ferramenta para composição de diagramas de componentes da UML. A ferramenta foi projetada como uma linha de produto de software e implementada como um *plugin* da plataforma Eclipse. Os resultados da avaliação mostraram que a ferramenta demonstrou ser efetiva para dar suporte a evolução de diagramas de arquitetura de software ao apresentar uma alta taxa de precisão, *recall* e *f-measure*.

Palavras-chave: composição de modelos, integração de modelos, diagrama de componentes.

1 INTRODUÇÃO

O alto nível de complexidade que os sistemas de software têm incorporado atualmente torna essencial a utilização de ferramentas para auxiliar nos processos de desenvolvimento. Essas ferramentas permitem que os desenvolvedores atuem nos pontos críticos com o intuito de reduzir sua dificuldade de implementação e, consequentemente, o esforço e o tempo de desenvolvimento. Dessa forma, a modelagem UML se tornou uma atividade fundamental para grande parte das empresas de software, de modo que a partir de um projeto bem definido e estruturado, diversas equipes de desenvolvimento possam realizar um trabalho em paralelo. Neste contexto, os desenvolvedores precisam constatemente compor modelos com o objetivo de gerar uma versão consolidada dos mesmos.

Composição de modelos pode ser definida como um conjunto de atividades a ser executada sobre dois modelos de entrada, M_A e M_B, com o objetivo de produzir um modelo integrado, M_{AB}, aquele desejado pelos desenvolvedores. Porém, usualmente um modelo composto com inconsistência, M_{CM}, é gerado ao contrário do M_{AB}. Isso acontece porque os modelos M_A e M_B apresentam partes conflitantes, as

* Estudante de Análise e Desenvolvimento de Sistemas na Universidade do Vale do Rio dos Sinos, São Leopoldo, Brasil. E-mail: gui_ermel@hotmail.com.

quais são tipicamente resolvidas incorretamente (FARIAS, 2012), (FARIAS, 2010), (GUIMARÃES, 2014). Consequentemente, os desenvolvedores devem investir algum esforço para detectar e resolver tais inconsistências no modelo M_{CM} com o objetivo de transformá-lo no modelo pretendido, M_{AB}.

Dado que frequentemente o modelo produzido e o modelo pretendido não são iguais ($M_{CM} \approx M_{AB}$), então a academia e a indústria têm proposto ao longo dos anos ferramentas de composição de modelos para dar suporte a integração de modelos. Exemplos destas ferramentas seriam IBM RSA[1] e Epsilon[2]. Ao mesmo tempo que as empresas de software reconhecem a importância da aplicação de técnicas de composição de modelos em suas atividades de desenvolvimento—visando consolidar o conhecimento e as alterações realizadas por diferentes equipes em um único modelo—as técnicas ainda são imprecisas ao ponto de não garantirem a ausência de inconsistências nos modelos composto produzidos (FARIAS, 2014) (FARIAS, 2012). Ou seja, o modelo resultante da composição não corresponde ao modelo esperado, M_{AB}, gerando assim um modelo com inconsistências, M_{CM}.

Este trabalho, portanto, propõe uma ferramenta para composição de diagramas de componentes da UML, a qual foi projetada como uma linha de produto de software e implementada como um *plugin* da plataforma Eclipse. A ferramenta foi utilizada na integração de diagramas de componentes de uma linha de produto de software (chamada de *MobileMedia* (FIGUEIREDO, 2008)) para dar suporte aos cenários de evolução sofridos pela linha de produto. Os resultados da avaliação mostraram que a ferramenta demonstrou ser efetiva para dar suporte a evolução de diagramas de arquitetura de software ao apresentar uma alta taxa de precisão, *recall* e *f-measure*.

O restante do artigo é organizado da seguinte forma. A Seção 2 apresenta o referencial teórico. A Seção 3 descreve o projeto e a implementação da ferramenta proposta. A Seção 4 descreve a avaliação realizada. A Seção 5 apresenta os trabalhos relacionados. A Seção 6 descreve as conclusões e os trabalhos futuros.

[1] http://www.ibm.com/developerworks/rational/products/rsa/
[2] https://www.eclipse.org/epsilon/

2 REFERENCIAL TEÓRICO

Essa seção apresenta os principais conceitos necessários para o entendimento deste trabalho. Para isso, a Seção 2.1 introduz o diagrama de componentes da UML. A Seção 2.2 apresenta uma breve definição de composição de modelos. A Seção 2.3 apresenta a ferramenta SDMetrics. A Seção 2.4 descreve formulas para avaliação dos resultados.

2.1 Diagrama de componentes da UML

Devido ao avanço tecnológico e a maior complexidade que os softwares têm incorporado, é fundamental ter ferramentas para auxiliar nos processos de elaboração de sistemas. Sendo assim, a modelagem UML se tornou uma atividade fundamental para grande parte das empresas de software se tornando um processo importante para definição do projeto e modularização do software. Desta forma, decompondo processos complexos em menores partes, gerando assim tarefas mais detalhadas e específicas, contendo menor complexidade e possibilitando um paralelismo de desenvolvimento, abstraindo sua complexidade e tornando o processo mais ágil.

Assim como descrito em (GUEDES, 2006), deve-se deixar claro, no entanto, que a UML não é uma linguagem de programação e sim uma linguagem de modelagem, cujo objetivo é auxiliar os engenheiros de software a definir as características do software, tais como seus requisitos, seus comportamentos, suas lógicas, e a dinâmica de seus processos a fim de facilitar a compreensão do desenvolvimento e manutenção de seus sistemas.

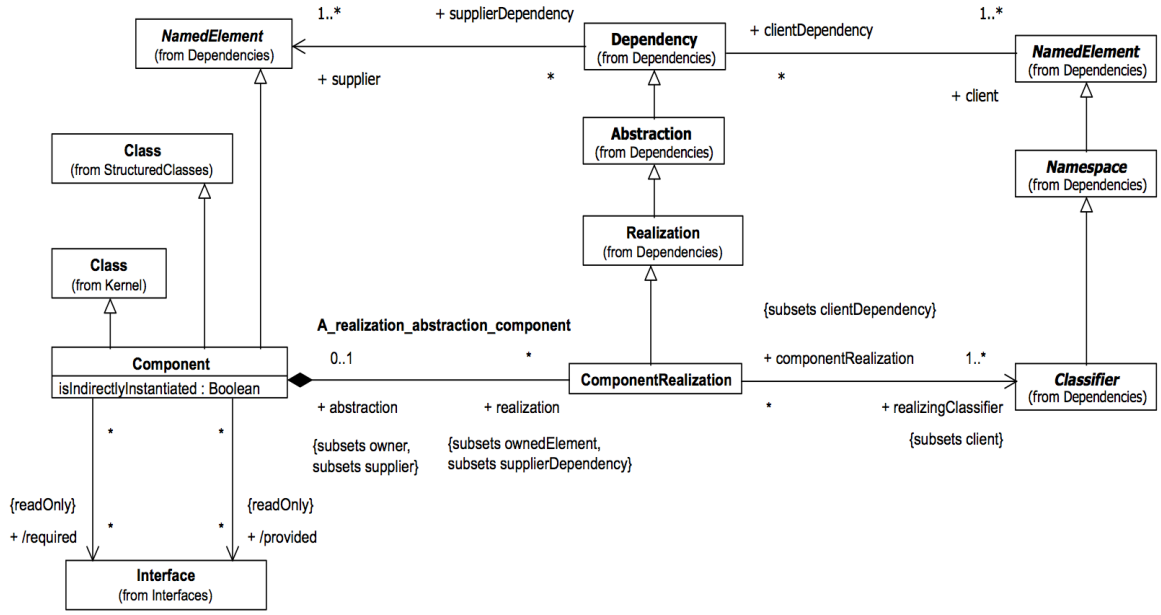
Para o desenvolvimento deste trabalho, o estudo é direcionado para o diagrama de componentes. Este diagrama UML permite uma visão dos módulos de um sistema e seus relacionamentos através de suas interfaces requerida e/ou provida. Logo, esta modelagem permite uma perspectiva de alto nível da estrutura interna do software, exibindo de maneira estática como este sistema será desenvolvido, apresentando assim, quais os componentes que irão incorporar o software. Desta maneira, proporcionando ao usuário a percepção da função de cada módulo, permitindo melhor organizar seus relacionamentos, auxiliando na

identificação de possíveis características e requisitos de software, além de facilitar sua reutilização em outros sistemas.

A Figura 1 apresenta o metamodelo da UML definindo a sintaxe abstrata do diagrama de componentes. Em especial, destaca-se o elemento *Component*, visto que o mesmo é o elemento primário no diagrama de componentes. A seguir, seu atributo e associações são descritas:

1. Atributos:
    - *isIndirectlyInstantiated*: Boolean {*default = true*}, assume valor *false* para indicar que o componente pode ser instanciado e ser referenciado. Por outro lado, quando assume valor *true*, (seu valor padrão) o componente assume características de uma abstração em nível de projeto; consequentemente, o mesmo não pode ser diretamente referenciado como objeto. Desse modo, o acesso aos comportamentos implementados pelo componente é acessível através de suas portas. Exemplo: *«specification»*, *«focus»*, *«subsystem»*.
2. Associações:
    - /*provided*: *Interface* [*], interface que define o que o componente expõe ao ambiente. Responsável por definir os serviços oferecidos pelo componente ao ambiente. Estas interfaces podem ser implementadas pelos seus *realizingClassifiers* (vide Figure 1).
    - /*required*: *Interface* [*], interface que define quais serviços o componente precisa do meio para fornecer os serviços especificados na interface provida.
    - *packagedElement*: *PackageableElement* [*], representa o conjunto de elementos que um componente pode ter como, por exemplo, classes, componentes, pacotes, etc.

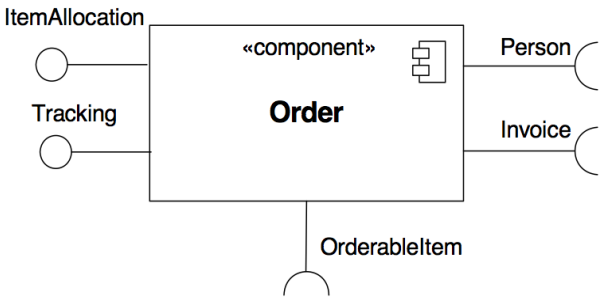
Figura 1 - Metamodelo da UML que define o diagrama de componentes



Fonte: OMG (2011)

A Figura 2 mostra o componente *Order*, o qual possui duas interfaces do tipo provida, *ItemAllocation* e *Tracking*, e três interfaces do tipo requerida, *Person*, *Invoice* e *OrderableItem*.

Figura 2 - Exemplo de componente



Fonte: (RUMBAUGH, 1999)

2.2 Composição de modelos

Pesquisadores e profissionais reconhecem a importância da composição modelo em muitas atividades de engenharia de software, por exemplo, na evolução dos modelos do design para adicionar novas funcionalidades e consolidar modelos desenvolvidos em paralelo por diferentes equipes de desenvolvimento de software

(FARIAS, 2014). É salientado, porém, uma grande necessidade na evolução dessas técnicas de modo a flexibilizar suas estratégias de composição. Isso se origina devido às diversas necessidades presentes entre os modelos, fazendo com que a execução dessa atividade não se mostre totalmente efetiva quanto aos resultados, assim, gerando muitas vezes um modelo que não coincide com o modelo esperado. Isso ocorre devido às técnicas de composição atuais serem muito rígidas e não atenderem as diferentes exigências nos modelos de entrada, assim, seguindo estratégias de composição inadequadas e não garantindo a ausência de inconsistências no modelo resultante.

As técnicas de composição usam dois modelos de entrada M_A e M_B, com o intuito de se produzir o modelo pretendido, M_{AB}, como anteriormente mencionado. Porém, esse modelo muitas vezes não é atingido, resultando então em modelo com inconsistências M_{CM} (FARIAS, 2014)(FARIAS, 2012). De acordo com (FARIAS, 2012), (FARIAS, 2010), (GUIMARÃES, 2014) isso ocorre devido os modelos M_A e M_B comumente conflitarem entre si. Estas inconformidades ocasionam decisões errôneas ou imprecisas nas estratégias de composição, convertendo em inconsistências no M_{CM}.

Na Engenharia de Software, porém, ainda não se tem estipulado um consenso sobre quais atividades e particularidades abrangem o processo de composição de modelos (FARIAS, 2014), isso se deve principalmente por ser uma nova área de pesquisa e que implica em grande complexidade.

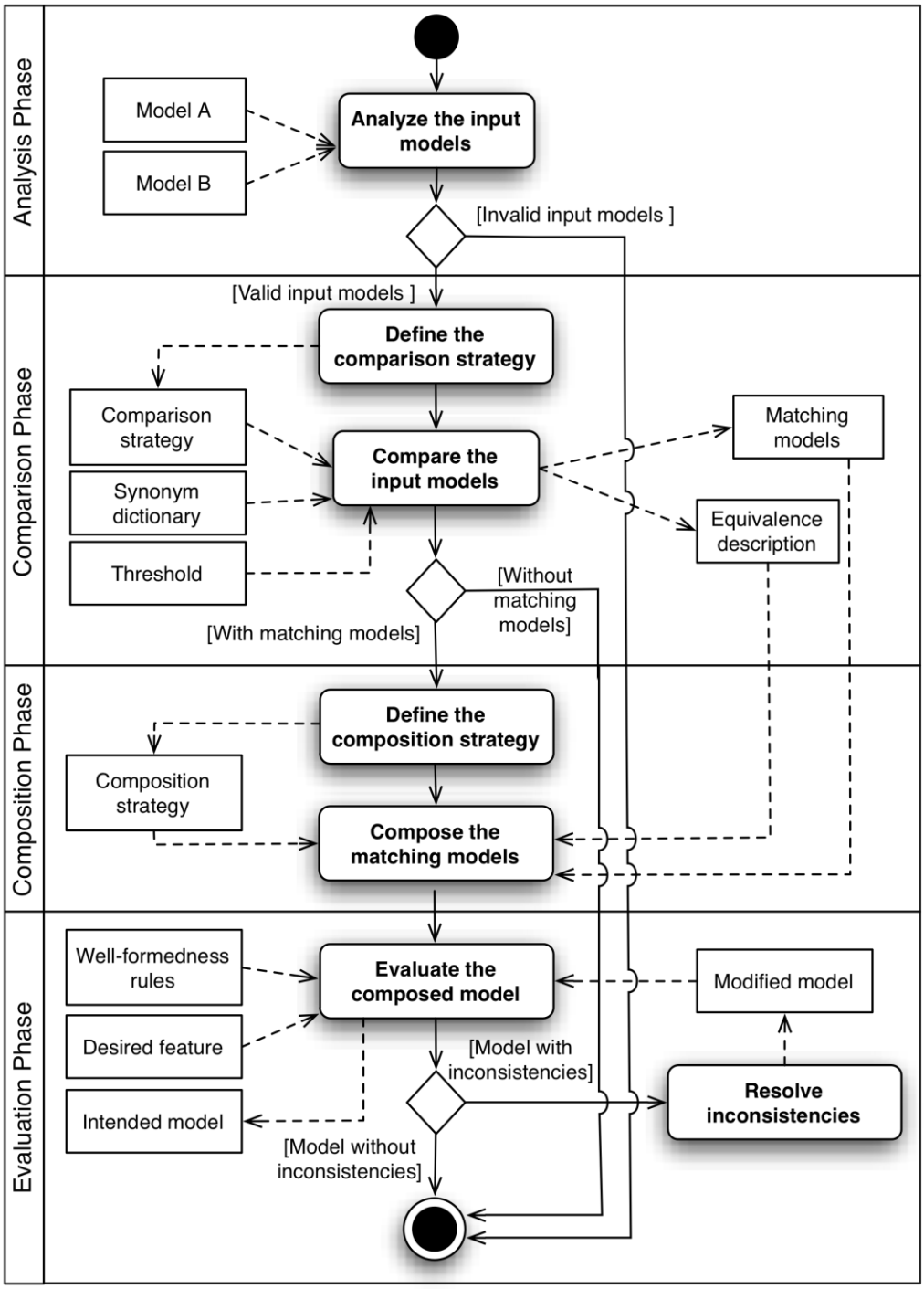
Em (FARIAS, 2014), o autor propões um processo que contempla as principais atividades e seus relacionamentos pertinente a um processo de composição de modelos. Esta pesquisa resultou no desenvolvimento de um diagrama de atividades, o qual é organizado em quatro fases, cada uma delas contendo respectivas atividades (Figura 3):

- *Analysis Phase*. Esta fase se inicia ao receber dois modelos de entrada. O conteúdo destes são analisados e organizados conforme seu tipo.
- *Comparison Phase*. Fundamentado nas informações elaboradas na fase anterior, nesta etapa especifica quais elementos internos são equivalentes. Para isto, é estipulado um grau de similaridade, e após comparar os elementos dos dois diagramas de entrada, observa-se

seu valor de similaridade e estabelece se determinados elementos são equivalentes.

- *Composition Phase*. Após definidos quais componentes internos se correspondem ou não, neste estágio é realizada a integração dos elementos equivalentes através da utilização de uma estratégia de composição (*merge*, *override* ou *union*). Produzindo assim o modelo composto.
- *Evaluation Phase*. Neste momento é verificado se o modelo obtido possui uma má formação. Isto é, se este apresenta conflitos. Quando encontrado conflitos, é executada uma atividade para transformação do modelo com base em regras para solucionas estas falhas. Esse processo é repetido até o momento que não se encontre mais inconsistências.

Figura 3 - Fluxo de Composição de Modelos



Fonte: (FARIAS, 2014)

2.3 SDMetrics

Como forma de mensurar a efetividade da ferramenta proposta e dos modelos compostos, o estudo de avaliação dos resultados é baseado em métricas resultantes da execução do software SDMetrics (SDMETRICS, 2014) sobre os diagramas UML gerados. Com isso, são comparadas as métricas resultantes entre os diagramas compostos M_{CM} com os esperados M_{AB}, analisando a eficácia a partir da aplicação das fórmulas de precisão (*precision*), *recall* e *f-measure* sobre os resultados.

O SDMetrics é uma ferramenta para analisar qualitativamente arquivos UML através da mensuração de métricas pré-estabelecidas. Este conjunto de medidas cobrem as propriedades estruturais de elementos de *design* para todos os tipos de diagramas UML. Desta forma, a partir da leitura do arquivo UML é possível analisar a existência de possíveis problemas pertinentes ao modelo.

Sua flexibilidade para definir métricas personalizadas o torna uma ótima ferramenta para que pesquisadores encontrem novas abordagens de medição de projeto. Sendo o SDMetrics assim, uma forte ferramenta de apoio a educação e pesquisa na área de engenharia de qualidade de produtos de software .

2.4 Medidas de avaliação

Neste trabalho são utilizadas três fórmulas para a análise da efetividade da ferramenta desenvolvida: *precision*, *recall* e *f-measure* (FRAKES, 1992). Estas fórmulas foram escolhidas para a avaliação por serem medidas utilizadas para verificar a efetividade de resultados, além de serem medidas que se complementam.

A fórmula de precisão tem a finalidade de medir o quão correto a integração ocorreu. Informando assim o percentual de elementos relevantes entre os que foram compostos. Isto é, entre os elementos que foram gerados em M_{CM}, quais estão presentes no modelo esperado M_{AB}. Porém, esta não verifica se todos os elementos relevantes foram gerados. Por essa razão utiliza-se a formula de *recall*. Esta por sua vez, analisa entre o total de elementos relevantes qual a porcentagem deles que foram gerados. Porém, esta também não é uma verificação totalmente eficiente, pois, não são levados em conta os elementos compostos que não são relevantes.

Pode-se perceber que estas duas fórmulas anteriores são eficazes em um ponto, porém falho em outro. Com isso, a fórmula *f-measure* tem a finalidade de

verificar a exatidão dos resultados gerados. Ou seja, são levados em conta os resultados obtidos com as fórmulas de *precision* e *recall*, podendo ser interpretada como uma média ponderada entre estas duas.

Como medidas de entrada para aplicação das fórmulas, são utilizadas para cada métrica: sua quantidade total presente no modelo composto (M_{CM}), total de elementos relevantes (elementos no modelo esperado) (M_{AB}) e a quantidade relevante no modelo composto ($M_{CM} \cap M_{AB}$). As fórmulas podem ser observadas abaixo (FRAKES, 1992):

$$precision = \frac{| M_{CM} \cap M_{AB} |}{| M_{CM} |}$$

$$recall = \frac{| M_{CM} \cap M_{AB} |}{| M_{AB} |}$$

$$F\text{-}Measure = \frac{2 . precision . recall}{precision + recall}$$

3. FERRAMENTA DE COMPOSIÇÃO DE MODELOS PROPOSTA

Com base em estudos anteriores (OLIVEIRA, 2008)(FARIAS, 2010)(FARIAS,2012)(FARIAS, 2014), a ferramenta foi projetada para executar o processo de composição de modelos através da realização das atividades identificadas dentro de cada uma das fases do diagrama de atividades apresentado na Figura 3. Desta forma, produzindo artefatos de saída que irão atuar sobre a os modelos de entrada.

Esta ferramenta consiste em fornecer um método flexível por meio de uma solução particular baseada em estratégias (regras de comparação, composição e avaliação) previamente informadas, assim, produzindo uma técnica de composição de modelos específica para a situação proposta. Com isso, através da manipulação dos modelos de entrada seguindo as múltiplas estratégias, as técnicas de composição podem resultar em modelos de saída perto do que é tido como o esperado, isso é $M_{CM} \approx M_{AB.}$

A partir desta proposta, esta seção descreve como a ferramenta é constituída. Exibindo suas características e atividades através da apresentação de seus

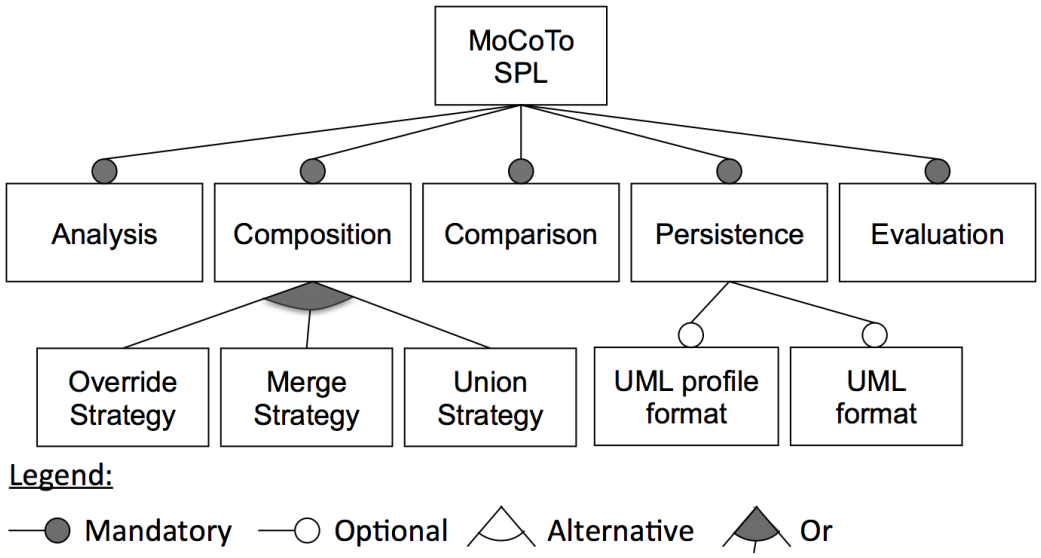
diagramas de classes, componentes, entre outros, de modo a exemplificar seu funcionamento e informar a composição de sua arquitetura interna.

3.1 Projeto e desenvolvimento da ferramenta

A composição de modelos é a definição para um conjunto de atividades, artefatos e regras que atuam sob os modelos de entrada tendo a finalidade de produzir um novo modelo M_{AB}. Essas atividades como descritas anteriormente, são realizadas de forma sequencial, porém, de maneira independente, onde cada atividade funciona separadamente. Contudo, para a execução de uma atividade, um conjunto de informações deve ser indicado, como por exemplo, na fase de composição devem ser informados quais são os elementos equivalentes dentro do modelo e qual deve ser a estratégia de composição aplicada, incluindo *merge*, *override* ou *union*.

A ferramenta está dividida por suas *features* essenciais, isto é, suas atividades principais que incluem *Análise*, *Comparação*, *Composição*, *Persistência* e *Avaliação* (Figura 4), estes são processos obrigatórios, ou seja, os quais são necessariamente executados no fluxo de composição de modelos. Além destes, algumas funcionalidades alternativas compõe a ferramenta, porém estas são executadas somente quando solicitadas pelo usuário.

Figura 4 - *Features* da ferramenta de composição



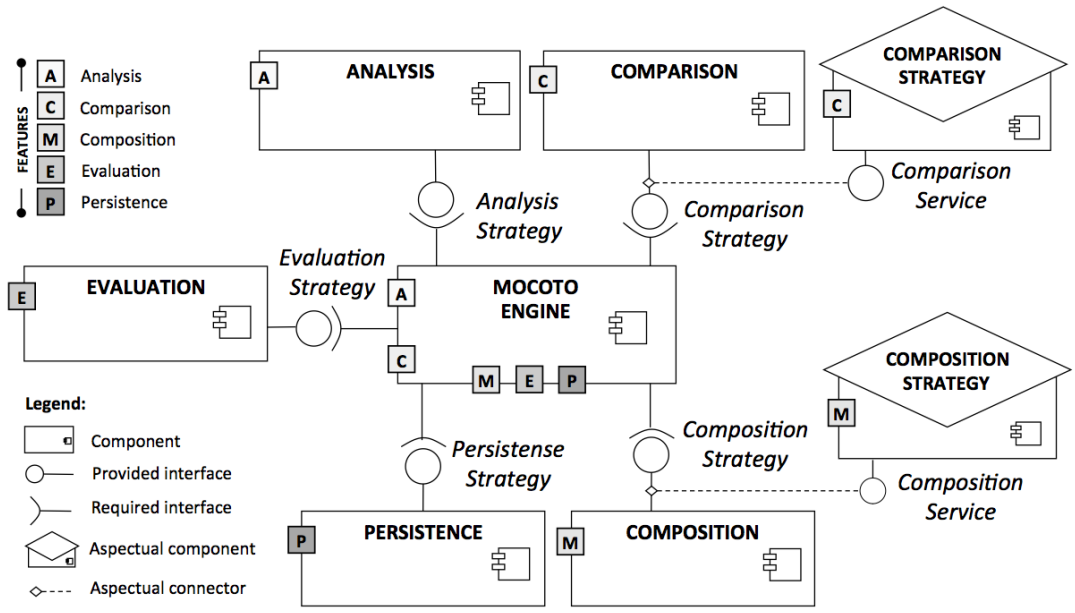
Fonte: (FARIAS, 2014)

3.1.1 Diagrama de componentes da ferramenta

A partir do discernimento das atividades, foram estipulados componentes os quais são responsáveis pela implementação de cada funcionalidade, relacionados diretamente com as atividades apresentadas na Figura 4. Esta forma de composição (Figura 5) permite que os componentes sejam tratados de maneira independente, de modo à modularizar os elementos do projeto. Com isso, promovendo o reuso dos componentes produzidos e permitindo um desenvolvimento e manutenção pontuais na ferramenta.

Cada componente é designado para ser um módulo autônomo que encapsula seu comportamento através de um conjunto de atividades internas. Estes, interagindo com elementos os quais indicam a execução de seu comportamento esperado. Sendo assim, este diagrama de componentes, concentra-se em apresentar os componentes como um grupo de elementos responsáveis por desempenhar um processo flexível a partir de uma configuração pré-estipulada pelo usuário.

Figura 5 - Diagrama de componentes



Fonte: (FARIAS, 2014)

Entre os elementos da ferramenta, a *Engine* é o componente central que administra todo o processo de composição. Este é responsável por disparar a

execução de cada atividade do sistema, sabendo tratar as respostas de cada função.

O componente *Analysis* é responsável por realizar verificações sob os modelos de entrada M_A e M_B, de modo a analisar se ambos são diagramas válidos, verificando se estes não possuem inconsistências, de maneira a informar ao sistema se o processo de composição pode ser aplicado sob estes modelos. Este é o primeiro componente a ser executado através da *Engine*.

Responsável pela segunda atividade do fluxo de composição, o *Comparison* desempenha a funcionalidade de comparação dos modelos, isto é, esta etapa especifica quais elementos entre os modelos são equivalentes. Está atividade pode ser considerada como um dos pontos centrais da ferramenta e de grande importância para o resultado final na geração de M_{CM}. Para isso, pode ser executada uma das três estratégias de comparação definidas: *DefaultMatchStrategy*, *PartialMatchStrategy* ou *CompleteMatchStrategy*.

O *Composition*, assim como o *Comparison*, também depende de uma estratégia envolvida a qual implementa sua funcionalidade. Onde, a partir das informações recebidas da atividade de comparação, este componente é responsável por estabelecer o modelo composto através de uma das estratégias de integração: *override*, *merge* ou *union*.

A etapa de avaliação do modelo composto é realizada pelo componente *Evaluation*. Este verificar se o modelo gerado possui inconsistências.

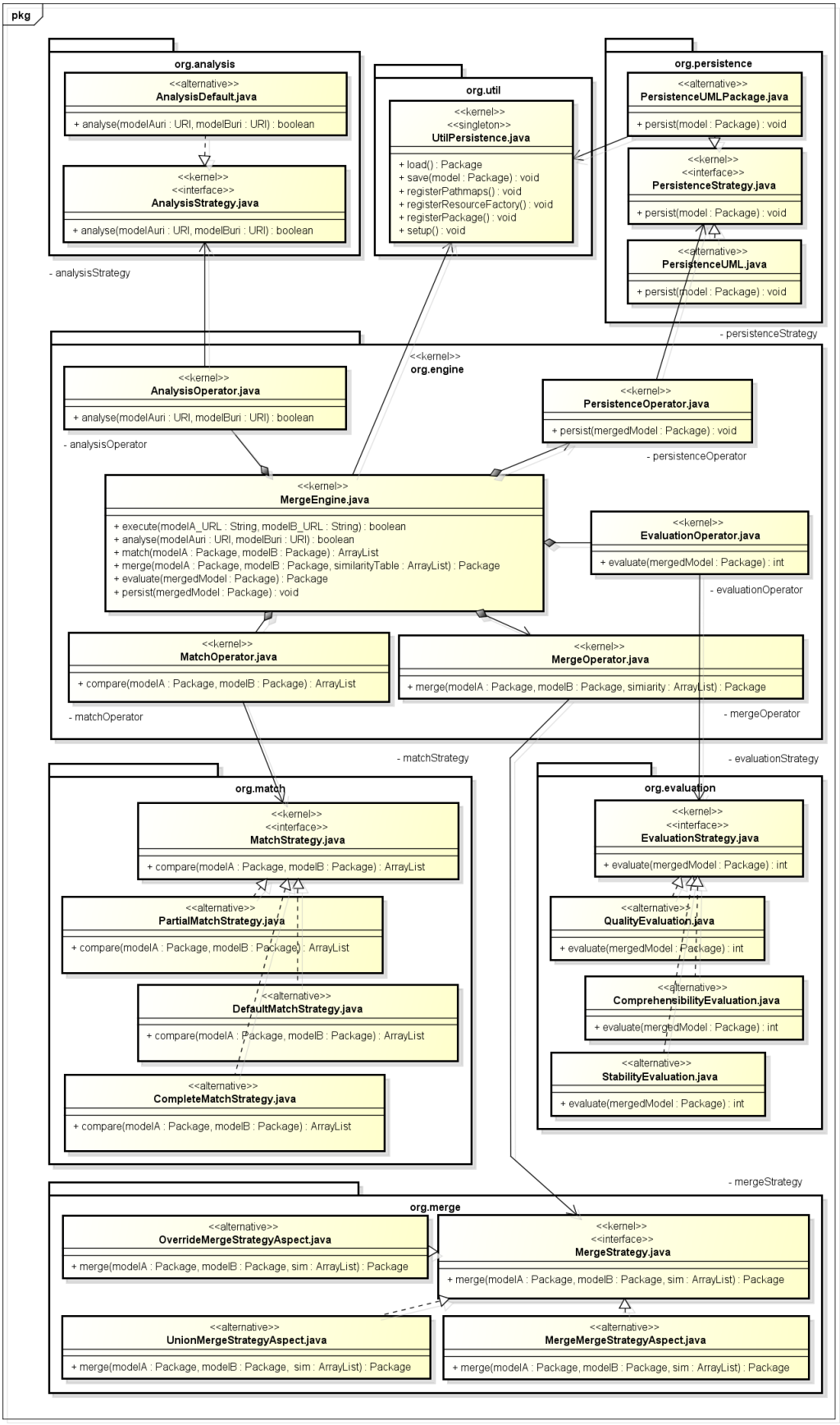
A última funcionalidade do processo de composição é desempenhada pelo componente *Persistence*. Este tem a reponsabilidade de gravar o modelo composto M_{CM} no disco rígido. Essa persistência pode ser gerada em dois formatos diferentes: *UML* ou *UML Package*, a qual é definida pelo usuário no início do processo.

3.1.2 Diagrama de classes da ferramenta

O diagrama de classes exibe de forma mais detalhada as classes que formam o diagrama de componentes, isto é, como as classes estão estruturadas dentro da ferramenta, quais os métodos que elas implementam, assim como o modo que elas se relacionam. Na Figura 6, este diagrama é representado através de sete pacotes separados de acordo com suas funcionalidades. Cada um desses pacotes representa a composição interna de um componente. Pode-se perceber que as

classes que possuem o estereótipo *kernel*, são atividades essenciais, isto significa que serão executadas obrigatoriamente no fluxo do processo. No entanto, as classes com o estereótipo *alternative* implementam funcionalidades específicas para uma determinada estratégia, estas são consideradas como opcionais e devem obedecer as regras de implementação definidas pela *interface*. Características as quais são percebidas na Figura 4.

Figura 6 - Diagrama de classes



Fonte: Elaborado pelo autor

3.1.3 Implementação

Como forma de exemplificar o modo que a ferramenta é executada e a sequência em que as atividades de composição ocorrem, abaixo é exibido um fragmento do código fonte pertencente à classe *MergeEngine.java*. Esta engloba todo o fluxo, ou seja, é o núcleo da ferramenta, responsável por gerir as chamadas e respostas das atividades através da execução dos métodos *analyse*, *compare*, *merge*, *evaluate* e *persist*.

```
1.  public boolean execute(String receivingURL, String meergedURL)  {
2.          Configurations.printConfigurations();
3.
4.          //Analyse
5.          if (!analyse(receivingURL, mergedURL)) {
6.                  return false;
7.          }
8.
9.          this.loadModels(receivingURL, meergedURL);
10.
11.         //Match
12.         ArrayList<SimilarityDegree> matchingResultList =
13.                 match(this.modelA, this.modelB);
14.
15.         //Merge
16.         Package mergedDiagram =
17.                 merge(this.modelA, this.modelB, matchingResultList);
18.
19.         //Evaluate
20.         Package evaluatedDiagram = evaluate(mergedDiagram);
21.
22.         //Persist
23.         persist(evaluatedDiagram);
24.
25.         return true;
26. }
```

Estes métodos, por sua vez, possuem a responsabilidade de instanciar e executar as classes *AnalysisOperator*, *MatchOperator*, *MergeOperator*, *EvaluationOperator* e *PersistenceOperator* presentes no diagrama de classes (Figura 6). Abaixo é apresentado o código referente ao método *match*, o qual realiza a atividade de comparação entre os modelos de entrada e retorna uma lista de similaridade entre seus elementos internos. Esta lista é utilizada posteriormente pelo método *merge*.

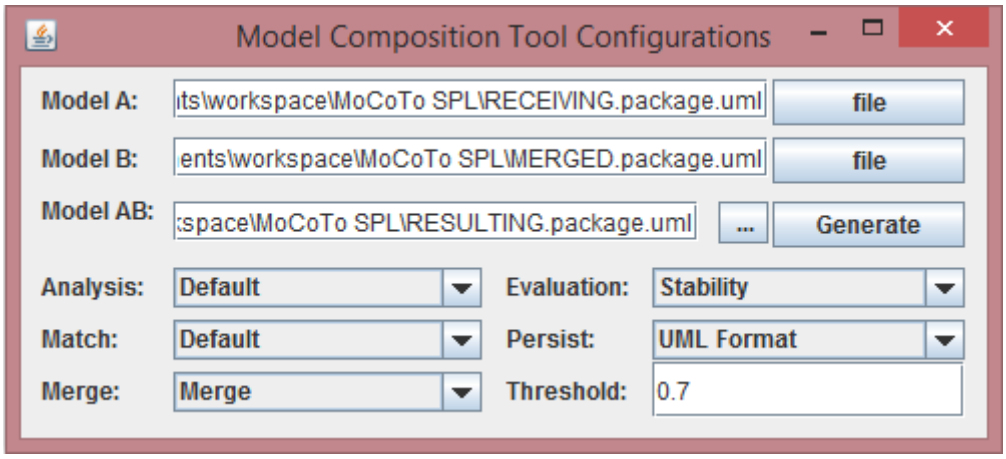
```
1. public ArrayList<SimilarityDegree> match(Package receiving, Package merged) {
2.
3.          matchOperator = new MatchOperator();
4.          SimilarityTable similarityTable = matchOperator.match(receiving, merged);
5.
6.          return similarityTable.getAllElementAboveThreshold(
7.                    Configurations.getThreshold());
8. }
```

3.2 A Ferramenta

A ferramenta de composição de modelos proposta é desenvolvida como um *plugin* da plataforma Eclipse (ECLIPSE, 2014), portanto ao ser executada, esta possui um aspecto como a IDE (*Integrated Development Environment*) do próprio Eclipse.

Para acessar a janela para definições de composição, o usuário deve entrar em seu menu de configurações (Figura 7). Nesta tela é requerido que sejam indicadas uma série de informações necessárias para a execução do processo. Para isso, o usuário especifica o caminho dos modelos de entrada M_A e M_B, qual deve ser a estratégia utilizada para as atividades de: comparação e composição de modelos, o modo de análise e avaliação, o formato de gravação do modelo composto, além do grau mínimo de similaridade entre os componentes internos.

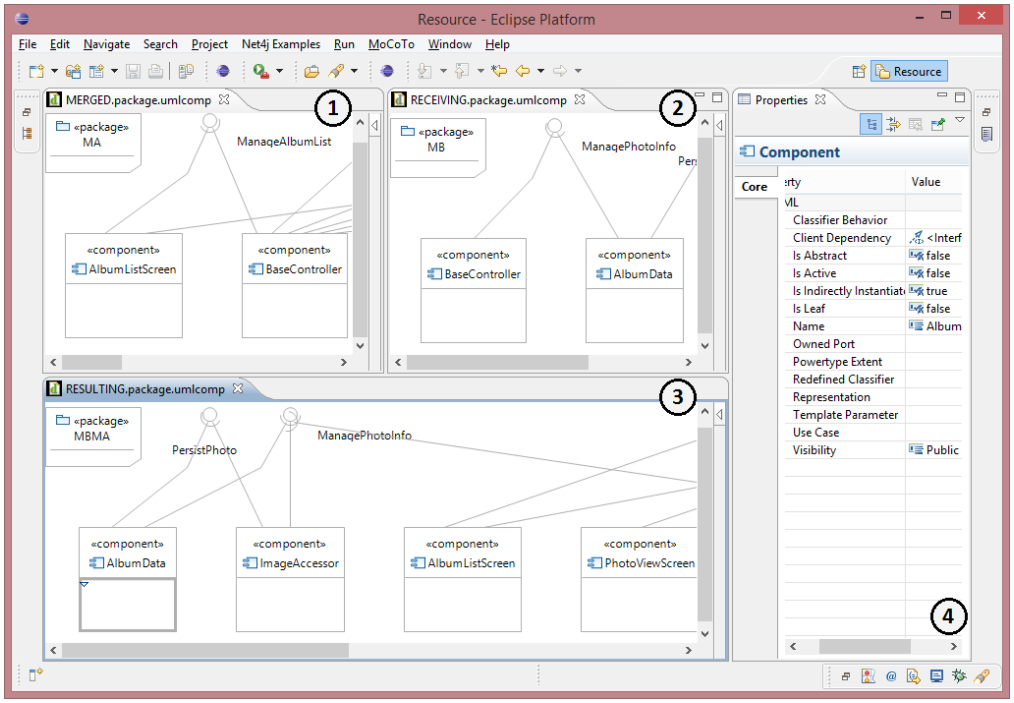
Figura 7 –Tela de Configurações



Fonte: Elaborado pelo autor

Após o usuário preencher estas configurações requeridas, a execução da composição de modelos pode ser inicializada clicando em *generate*. Então, ao finalizar o processo, seu resultado pode ser exibido conforme a Figura 8. Esta tela está dividida em algumas áreas (informadas na imagem). Nas áreas 1 e 2 são projetados os modelos de entrada M_A e M_B, respectivamente. Na região 3 é exibido o diagrama resultante M_{CM}. A área 4 são apresentadas as propriedades de um elemento do diagrama, quando selecionado.

Figura 8 – Resultado da composição de modelos



Fonte: Elaborado pelo autor

4 AVALIAÇÃO DA FERRAMENTA

Como forma de mensurar a efetividade da ferramenta ao produzir os modelos compostos, este estudo utiliza métricas coletadas através da execução do software SDMetrics sobre os diagramas UML gerados. Além disso, são utilizados como modelos de entrada um conjunto de diagramas que simulam a evolução de uma linha de produto de software, chamada de *MobileMedia* (FIGUEIREDO, 2008). Sendo assim, métricas são aplicadas ao modelo composto M_{CM} e ao modelo desejado M_{AB} para avaliar a precisão (*precision*), *recall* e *f-measure* das composições.

4.1 Método de avaliação

Para esta avaliação estão sendo utilizados modelos relativos à simulação para três diferentes etapas do ciclo de desenvolvimento de um software para gerenciamento de imagens. Onde, em cada um destes momentos possui-se o modelo de referência M_A e um novo modelo M_B o qual representa aquilo o que foi desenvolvido naquele ciclo. A partir destes diagramas foram elaborados três experimentos, onde é aplicada a execução da ferramenta de composição de modelos sobre os diagramas de entrada M_A e M_B originando em um novo modelo M_{CM}. Neste estudo é utilizado uma configuração padrão para a composição, isto é, a ferramenta aplica as estratégias Análise *default*, Comparação *default*, Integração *merge*, Avaliação *stability*.

Tendo em base o arquivo UML referente ao diagrama M_{CM} gerado anteriormente e o arquivo UML relativo ao modelo esperado $M_{AB,}$ executa-se o SDMetrics, compara-se suas métricas originadas e então são aplicadas as fórmulas para mensurar a eficácia do modelo composto. Ainda, as métricas que estão sendo consideradas nesse estudo são propriedades direcionadas ao diagrama de componentes, e são apresentadas na tabela abaixo.

Tabela 1. Métricas

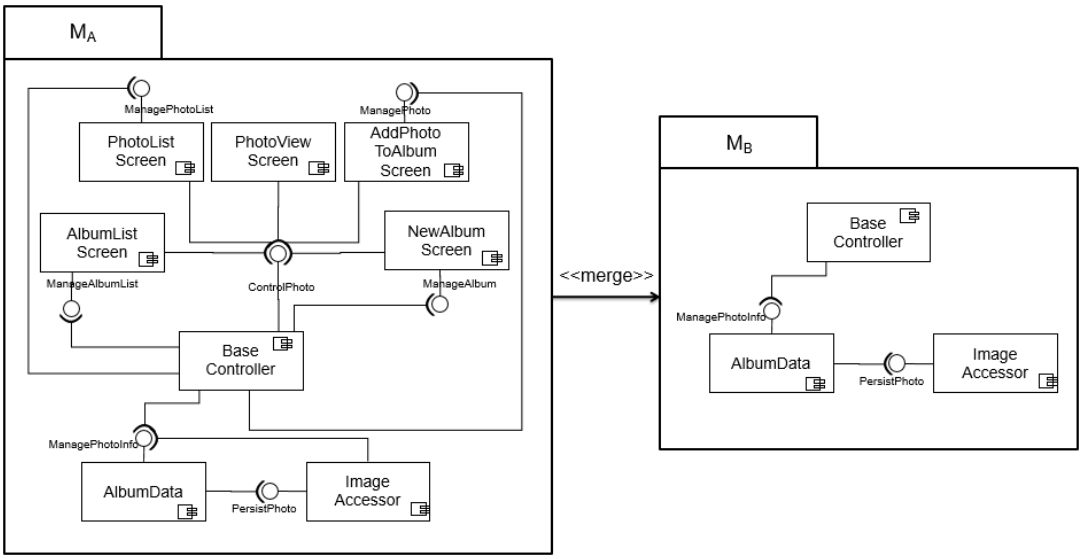
Métrica	Descrição
#Comp	Métrica que contabiliza o número de componentes presentes no modelo
#IntProv	Métrica referente ao número de *interfaces* providas entre todos os componentes
#IntReq	Métrica que calcula o número de *interfaces* requeridas entre os componentes do modelo
#Operat	Métrica responsável por informar o total de operações referentes a todas as *interfaces* do modelo

Abaixo são apresentados os resultados da execução da composição de modelos para cada um dos ciclos de desenvolvimento.

4.1.1 Experimento 1

Para o primeiro experimento utiliza-se os diagramas representados na Figura 9. Pode-se perceber que os elementos presentes em M_B já se encontram em M_A, portanto o modelo composto resultante torna-se igual a M_A. Os resultados podem ser observados na Tabela 2.

Figura 9 – Composição 1



Fonte: (FIGUEIREDO, 2008)

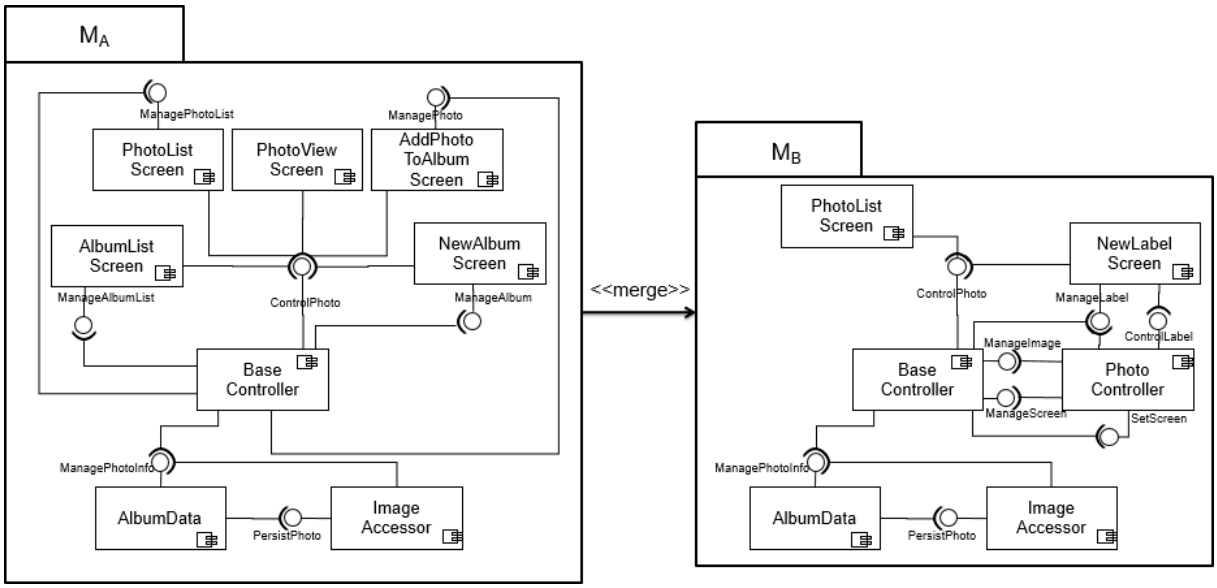
Tabela 2. Avaliação da ferramenta – 1º Experimento

Métrica	M_A	M_B	M_{AB}	M_{CM}	$M_{CM} \cap M_{AB}$	*Precision*	*Recall*	*F-Measure*
#Comp	8	3	8	8	8	1	1	1
#IntProv	7	2	7	7	7	1	1	1
#IntReq	12	2	12	12	12	1	1	1
#Operat	37	20	37	37	37	1	1	1

4.1.2 Experimento 2

O segundo cenário, apresentado na Figura 10, exibe uma grande implementação presente em M_B, podendo notar dois novos componentes *NewLabelScreen* e *PhotoController*, além de novas interfaces providas e requeridas. Além destas alterações, foram adicionadas e removidas algumas operações nesta etapa do desenvolvimento (as quais não são mostradas na ilustração).

Figura 10 – Composição 2



Fonte: (FIGUEIREDO, 2008)

Tabela 3. Avaliação da ferramenta – 2º Experimento

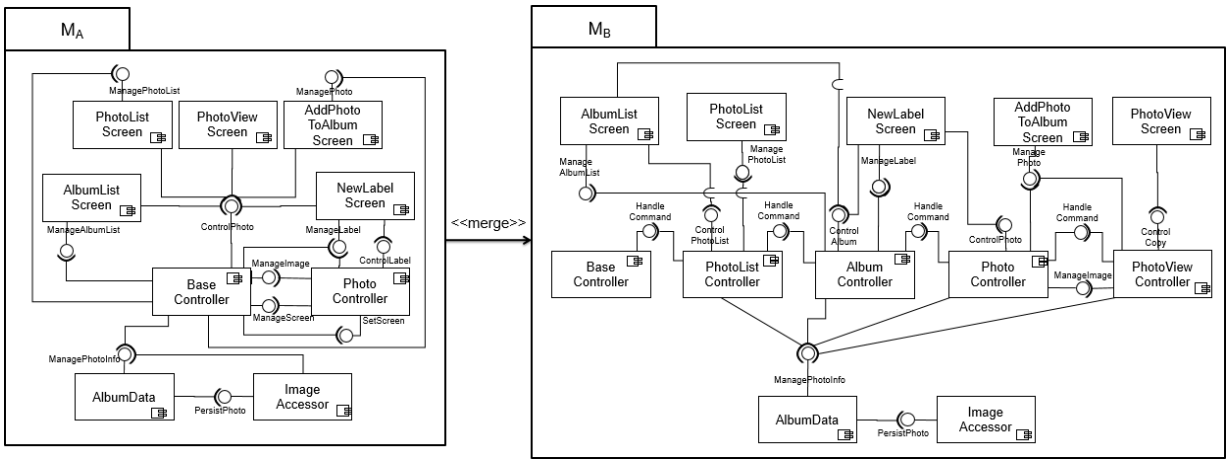
Métrica	M_A	M_B	M_{AB}	M_{CM}	$M_{CM} \cap M_{AB}$	*Precision*	*Recall*	*F-Measure*
#Comp	8	6	9	10	9	0,9	1	0,95
#IntProv	7	8	11	12	11	0,92	1	0,96
#IntReq	12	11	17	19	17	0,89	1	0,94
#Operat	37	41	47	49	47	0,96	1	0,98

Nesse cenário, pode-se perceber uma diferença entre aquilo que era esperado para o resultado gerado. Isso ocorre devido a M_{CM} conter elementos que foram removidos no desenvolvimento, porém estão presentes em M_A. A proposta da ferramenta não prevê a remoção de elementos entre os modelos, apenas procura a similaridade entre estes para adicionar no modelo resultante. Assim, este é observado como uma possível melhoria futura.

4.1.3 Experimento 3

Neste cenário, analisando os resultados (Tabela 4), se percebe outra característica a ser melhorada na ferramenta, porém algo difícil de se resolver. Pode-se observar que em M_A, o componente *BaseController* possui a interface *ControlPhoto*, porém, no ciclo de desenvolvimento esta interface foi movida para o componente *PhotoController*, aparecendo assim em M_B conforme a Figura 11. Com isso, os componentes que utilizavam esta interface em M_A deveriam passar a apontar para a nova interface em *PhotoController*. No entanto, a ferramenta não identifica que a interface *ControlPhoto* em M_A se refere a mesma presente em M_B, já que esta poderia apenas ser uma nova interface com mesmo nome e métodos (assim como em M_B possuímos diversas interfaces *HandleCommand*).

Figura 11 – Composição 3



Fonte: (FIGUEIREDO, 2008)

Tabela 4. Avaliação da ferramenta – 3º Experimento

Métrica	**M_A**	**M_B**	**M_{AB}**	**M_{CM}**	**$M_{CM} \cap M_{AB}$**	***Precision***	***Recall***	***F-Measure***
#Comp	9	12	12	14	10	0,71	0,83	0,77
#IntProv	11	15	15	20	15	0,75	1	0,86
#IntReq	17	20	26	36	23	0,64	0,88	0,74
#Operat	47	56	56	74	56	0,76	1	0,86

Outro fato relevante que ocorreu nessa simulação pode ser visto na primeira métrica da tabela (quantidade de componentes). Onde foram criados 14 componentes, porém apenas 10 conforme o esperado. Isso ocorreu, pois, os elementos *BaseController* e *PhotoController* de M_A e M_B não obtiveram uma similaridade igual ou superior a exigida no cenário de configuração (0.7). Dessa forma, foram gerados *MA_BaseController, MB_BaseController*, *MA_PhotoController*, *MB_PhotoController*.

Dessa forma, em M_{CM} observamos duas situações, a primeira delas é a mesma característica já descrita no cenário 2, onde a ferramenta não remove componentes ou interfaces. A outra particularidade é por não considerar uma similaridade entre interfaces de componentes diferentes, desta forma podendo fazer que uma interface seja provida por outro componente em M_{CM}.

4.2 Resultados

Em um cenário controlado, onde pode-se observar e analisar os resultados gerados sobre aquilo que era esperado, a ferramenta apresentou um bom desempenho ao atingir um alto nível de *precision*, *recall* e *f-measure* nos três experimentos. Além disso, se verificou um comportamento regular para as diferentes necessidades em cada um dos variados cenários.

Porém, foram encontrados alguns pontos a se evoluir no processo. De modo a aperfeiçoar principalmente as atividades de comparação e composição que são *features* que trazem maior impacto sobre o resultado final.

5 TRABALHOS RELACIONADOS

Para a realização do processo de composição de modelos, é necessário empregar alguma ferramenta ou técnica para sua execução. Estas suportam e auxiliam os desenvolvedores a realização das atividades fundamentais para a composição, tais como: comparação de modelos para identificar semelhanças, indicação e auxilio para resolução de conflitos, integração para estabelecer o modelo unificado. Com isso, o número de ferramentas e técnicas comerciais e acadêmicas para composição de modelos tem crescido com o passar dos anos, assim como uma ampla quantidade de trabalhos relacionados à área de composição. Com o intuito de explorar as atuais ferramentas disponíveis para a modelagem UML e principalmente para composição de modelos, este trabalho procurou analisar estas referências no momento do desenvolvimento da ferramenta, tanto quanto para a análise dos resultados.

5.1 Ferramentas para composição de modelos

Apesar de ser uma área que ainda se inicia em estudos aprofundados, já é apresentada uma vasta gama de alternativas para realização do processo de composição. Entre essas, pode-se observar algumas ferramentas de grande importância como a IBM RSA (IBM, 2014), um software de modelagem desenvolvido pela IMB que através da composição de modelos suporta o trabalho colaborativo; Epsilon (EPSILON, 2014), um plug-in para o Eclipse, o qual abrange uma extensa biblioteca para manipulação de modelos; entre muitas outras. Na Tabela 5, são apresentadas algumas ferramentas analisadas no estudo.

Tabela 5. Ferramentas para composição de modelos

Ferramenta	Descrição
IBM Rational Software Architect	Software de modelagem robusto de grande influencia no mercado, o qual suporta a composição de modelos (IBM, 2014).
Epsilon	Plug-in para o Eclipse disponibilizando uma vasta biblioteca para manipulação e gerenciamento de modelos (KOLOVOS, 2014).
EMF	Framework de modelagem da plataforma Eclipse que fornece um conjunto de funcionalidade para manipulação de modelos (EMF, 2014).
Astah	Ferramenta de modelagem que suporta composição de modelos (ASTAH, 2014).
Borland Together	Ferramenta que fornece análise e modelagem do design suportando o trabalho colaborativo entre equipes e desenvolvedores (BORLAND, 2014).
Kompose	Técnica acadêmica que fornece um mecanismo genérico de composição de artefatos de software (KOMPOSE, 2014).
MATA	Abordagem de modelagem Orientada a Aspectos (OA) que suporta o mecanismo de empacotamento definido em AO.
Atlas Model Weaver	Ferramenta que utiliza o plug-in EMF a qual estabelece relações entre modelos. Podendo ser utilizado para rastreabilidade, integração de modelos, comparação do modelo, entre outros (ATLANMOD, 2014).

Porém, apesar do amplo número de ferramentas disponibilizadas, nenhuma técnica tem se mostrado totalmente eficaz para as diferentes necessidades presentes nos modelos, de modo que estas não garantem a ausência de inconsistências, resultando muitas vezes em um modelo M_{CM}. Em consequência, é exigida a interação manual sob os modelos, necessitando a analise e o conhecimento da arquitetura para atuar sob as inconsistências apresentadas, para assim obter o modelo ideal M_{AB}.

5.2 Estudos para composição de modelos

Com essa abordagem descrita no item anterior, fica evidente a necessidade dos avanços nas técnicas para composição de modelos. Assim como, por outro lado, pode-se observar o grande número de trabalhos nesse campo de pesquisa em uma bibliografia crescente. Porém, visto que a técnica de composição de modelos é um processo de grande complexidade, esta pode ser considerada uma área de pouco

aprofundamento teórico até o momento, comparado com outras áreas na computação que estão mais evoluídas devido ao maior tempo e esforços destinados. Em (FARIAS, 2012)(FARIAS, 2010)(FARIAS, 2013), os autores salientam ainda que devido a esta complexidade e por ser uma área de pesquisa que ainda se principia, é inexistente um consenso na comunidade de engenharia de software quanto à definição de atividades e particularidades pertinentes a uma solução quanto à composição de modelos. De modo que é necessário um grande investimento de tempo e esforço para realização de estudos nessa área. Com a finalidade de modificar essa realidade, em seu trabalho foi elaborado um guia para auxiliar o desenvolvimento da composição (OLIVEIRA, 2008). Assim como Oliveira, BÉZIVIN (2006) salienta que há pouco consenso sobre a terminologia na composição do modelo, e ainda menos sobre as principais características de uma solução de composição modelos. Em sua pesquisa, é realizada uma comparação com três frameworks: AMW (*AtlanMod Model Weaver*), EMF (*Epsilon Merging Language*) e *Glue Generator*, procurando características comuns de composição.

Entre os estudos relacionados, pode-se perceber diferentes contextos de pesquisa. FLEUREY (2007) ressalta a importância da modularização na modelagem para grandes sistemas, de modo a diminuir a complexidade e sua manutenabilidade, e com isso a necessidade de se integrar modelos. Em sua pesquisa foi elaborada como solução, uma proposta de framework para composição. Por outro lado, SELONEN (2007) focaliza o estudo na atividade de comparação de modelos presentes nas ferramentas de composição, apresentando um estudo abrangente sob diversas ferramentas no mercado e no meio acadêmico. Ainda, em FARIAS (2010), FARIAS (2012), FARIAS (2013), os autores desenvolvem um trabalho com o enfoque em medir o esforço na realização do processo de composição de modelos.

Apesar das diferentes abordagens entre os estudos, pouco é conhecido sobre a flexibilidade e capacidade das técnicas atuais. De modo que a carências dessas informações dificulta a evolução e o aperfeiçoamento da técnica e estratégias para composição de novos modelos que até então não são suportados com eficácia.

6 CONCLUSÃO

A modelagem de diagramas UML para auxiliar e documentar o desenvolvimento de softwares é uma realidade nas empresas de TI. Permitindo

assim que diversas equipes atuem em um mesmo sistema e realizem um trabalho em paralelo, visando um desenvolvimento ágil com um mínimo de impacto entre estas. Porém, devido ao maior número de profissionais envolvidos para a manutenção e o desenvolvimento dessas ferramentas, são observados recorrentes conflitos ocasionados por alterações simultâneas nas características no modelo de referência, de modo que sejam elaborados diferentes modelos para o mesmo sistema, cada um empregando novas particularidades.

Em vista disso, este estudo apresentou uma necessidade presente em grande parte das empresas de TI e uma tecnologia que vem sendo utilizada para suprir este problema: a integração de modelos a partir da realização da composição. Contudo, torna-se evidente que as técnicas de composição não tem se mostrado totalmente eficazes para as diferentes necessidades presentes nos modelos. Visto que, ferramentas semi-automatizadas exigem um alto conhecimento do usuário para resolução de conflitos, enquanto ferramentas automatizadas realizam essas tomadas de decisões automaticamente, a composição resulta muitas vezes em modelos inadequados, apresentando inconsistências ou falhas estruturais.

Como proposta para evolução dessa realidade, este estudo apresenta a implementação de uma ferramenta para composição de modelos a qual, apesar de ser uma ferramenta automatizada, através da realização de tomadas de decisões automáticas, é flexível por permitir diversas combinações de estratégias em seu fluxo de composição. Desta forma, contribuindo como uma forma alternativa de se projetar ferramentas automáticas, onde pode-se resultar modelos mais próximos do esperado com um menor esforço, o que foi comprovado através da avaliação de seus resultados.

Com base nesse trabalho é perceptível que a composição de modelos é uma área de pesquisa desafiante a qual abrange uma série de atividades complexas que necessitam de evolução e aperfeiçoamento. Desta forma, como trabalhos futuros são observadas possíveis melhorias no que se refere às características da ferramenta de composição proposta, encontradas na seção de avaliação da ferramenta, através de um aperfeiçoamento nas técnicas de *match*, *merge* e *evaluation*. Fazendo com que a ferramenta se torne mais completa e abrangente as necessidades dos modelos, e com isso tornando o modelo resultante mais próximo do modelo esperado.

A Tool for composition of UML component diagrams

Abstract: Models composition plays a central role in many software engineering activities, e.g., evolving design models or reconciling models developed in parallel. Hence, many techniques have been proposed in the last years, e.g., IBM RSA and Epsilon. Nevertheless, several studies in the literature claim these techniques are inaccurate to compose component diagrams, whereas inconsistencies can be inserted into the resulting model. This paper, therefore, presents a tool for composition of component diagrams of UML. The tool was designed as a software product line and developed as an Eclipse plugin. The results indicate the tool was effective to support evolving design models for presenting high rate of precision, recall, and f-measure.

Keywords: Models composition, component diagrams.

REFERÊNCIAS

ASTAH. Disponível em: <http://www.astah.net/>. Acesso em: 28 de Outubro de 2014.

ATLANMOD Model Weaver (AMW). Disponível em: <http://wiki.eclipse.org/AMW>. Acesso em: 28 de Outubro de 2014.

BÉZIVIN, J., Bouzitouna, S., Fabro, M., Gervais, M., Jouault, F., Kolovos, D., Kurtev, I., Paige, R.: **A Canonical Scheme for Model Composition**, 2006.

BORLAND Together. Disponível em: <https://www.borland.com/products/together/>. Acesso em: 28 de Outubro de 2014.

ECLIPSE. Disponível em: <http://www.eclipse.org/>. Acesso em: 28 de Outubro de 2014.

EMF Eclipse Modeling Framework. Disponível em: <http://www.eclipse.org/modeling/emf/>. Acesso em: 28 de Outubro de 2014.

EPSILON. Disponível em: <https://www.eclipse.org/epsilon/>. Acesso em: 28 de Outubro de 2014.

FARIAS, K. **Empirical Evaluation of Effort on Composing Design Models**, In: 32nd ACM/IEEE International Conference on Software Engineering, Doctoral Symposium, Vol. 2, pages 405-408, Cape Town, South Africa, 2010.

FARIAS, K.; GARCIA, A.; WHITTLE, J. **Assessing the Impact of Aspects on Model Composition Effort**, In: 9th International Conference on Aspect-Oriented Software Development (AOSD'10), pages 73-84, Rennes and Saint-Malo, France, 2010.

FARIAS, K. **Empirical Evaluation of Effort on Composing Design Models**, PhD thesis, DI/PUC-Rio, Rio de Janeiro, Brazil. 2012.

FARIAS, K.; GARCIA, A.; LUCENA, C. **Effects of Stability on Model Composition Effort: an Exploratory Study**, Journal on Software and Systems Modeling, pages 1-22, Springer-Verlag, DOI: 0.1007/s10270-012-0308-2, 2013.

FARIAS, K; GARCIA, A; WHITTLE, J; CHAVEZ, C; LUCENA, C. **Evaluating the Effort of Composing Design Models: A Controlled Experiment**, In: Proceedings of the 15th International Conference on Model-Driven Engineering Languages and Systems (MODELS'12), Vol. 7590, pages 676-691, Innsbruck, Austria, 2012.

FARIAS, K.; GARCIA, A.; WHITTLE, J.; LUCENA, C. **Analyzing the Effort of Composing Design Models of Large-Scale Software in Industrial Case Studies**, In: Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS'13), pages 639-655, Miami, USA, September 2013.

FARIAS, K; GONÇALVES, L; SCHOLL, M. **Toward a Software Product Line for Model Composition Techniques**, PIPCA/UNISINOS, São Leopoldo, Brazil, 2014.

FIGUEIREDO, E. *et al*. **Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability**, In proceedings of the 30th International Conference on Software Engineering (ICSE), pp. 261-270. Leipzig, Germany, 10-18 May 2008.

FLEUREY, F.; BAUDRY, B.; FRANCE, R.; GHOSH, S.: **A Generic Approach For Automatic Model Composition**, In: Aspect Oriented Modeling (AOM) Workshop, 2007.

FRAKES W.B.; BAEZA-YATES R.; **Information Retrieval: Data Structures and Algorithms**. Prentice-Hall, 1992.

GUEDES, G.: **UML Uma abordagem Prática**, 2ª Edição, 2006.

GUIMARAES, E.; GARCIA, A.; FARIAS, K.: **On the Impact of Obliviousness and Quantification on Model Composition Effort**, In: Proceedings of the 29th Symposium On Applied Computing (SAC.14), Gyeongju, Korea, March, 2014

IBM Rational Software Architect (IBM RSA). Disponível em: <http://www.ibm.com/developerworks/rational/products/rsa/>. Acesso em: 28 de Outubro de 2014.

KOLOVOS, D.; ROSE, L.; PAIGE, R.: **The Epsilon book**, Disponível em: <http://www.eclipse.org/epsilon/doc/book/>. Acesso em: 28 de Outubro de 2014.

KOMPOSE. Disponível em: <http://www.kermeta.org/kompose/>. Acesso em: 28 de Outubro de 2014.

OLIVEIRA, K.: **Composição de UML Profiles**. Master's thesis, FACIN/PUCRS, Porto Alegre, Brasil (2008).

OMG, Unified Modeling Language: **Superstructure**, Version 2.4.1, 2011.

RUMBAUGH, J., Jacobson, I., Booch, G.: **The Unified Modeling Language Reference Manual**, Addison-Wesley, 2a edition, 1999.

SDMETRICS. Disponível em: <http://www.sdmetrics.com/>. Acesso em: 28 de Outubro de 2014.

SELONEN, P.: **A Review of UML Model Comparison Approaches**, In: Nordic Workshop on Model Driven Engineering, 2007.